UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

TESIS DOCTORAL

Contributions to Non-Iterative Super-Resolution Algorithms Enabling Efficient Real-Time FPGA Implementation for Resolution Enhancement of Video Sequences

Tomasz Szydzik

Instituto Universitario de Microelectrónica Aplicada

Las Palmas de Gran Canaria, Noviembre 2015

D. PEDRO PÉREZ CARBALLO SECRETARIO DEL INSTITUTO UNIVERSITARIO DE MICROELECTRÓNICA APLICADA DE LA UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA,

CERTIFICA,

Que el Consejo de Doctores del Departamento en su sesión de fecha **13 de noviembre de 2015** tomó el acuerdo de dar el consentimiento para su tramitación, a la tesis doctoral titulada "***Contributions to Non-Iterative Super-Resolution Algorithms Enabling Efficient Real-Time FPGA Implementation for Resolution Enhancement of Video Sequences***" presentada por el doctorando Tomasz Szydzik y dirigida por los Doctores D. Gustavo I. Marrero Callicó, y D. Antonio Nuñez Ordoñez.

Y para que así conste, y a efectos de lo previsto en el Artº 6 del Reglamento para la elaboración, defensa, tribunal y evaluación de tesis doctorales de la Universidad de Las Palmas de Gran Canaria, firmo la presente en Las Palmas de Gran Canaria, a **13 de noviembre de 2015**.

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

Instituto: INSTITUTO UNIVERSITARIO DE MICROELECTRÓNICA APLICADA

Programa de doctorado: INGENIERÍA DE TELECOMUNICACIÓN AVANZADA

Título de la Tesis

CONTRIBUTIONS TO NON-ITERATIVE SUPER-RESOLUTION ALGORITHMS ENABLING EFFICIENT REAL-TIME FPGA IMPLEMENTATION FOR RESOLUTION ENHANCEMENT OF VIDEO SEQUENCES

Tesis Doctoral presentada por D. TOMASZ SZYDZIK

Dirigida por Dr. D. GUSTAVO I. MARRERO CALLICÓ

Codirigida por Dr. D. ANTONIO NUÑEZ ORDOÑEZ

El Director,	El Codirector	El Doctorando,
(firma)	(firma)	(firma)

Las Palmas de Gran Canaria, a 13 de noviembre 2015

D. PEDRO PÉREZ CARBALLO SECRETARIO DEL INSTITUTO UNIVERSITARIO DE MICROELECTRÓNICA APLICADA DE LA UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA,

CERTIFICA,

Que el Consejo de Doctores del Departamento en su sesión de fecha **13 de noviembre de 2015** ha acordado que la tesis doctoral titulada "***Contributions to Non-Iterative Super-Resolution Algorithms Enabling Efficient Real-Time FPGA Implementation for Resolution Enhancement of Video Sequences***" presentada por el doctorando Tomasz Szydzik y dirigida por los Doctores D. Gustavo I. Marrero Callicó, y D. Antonio Nuñez Ordoñez **reúne todo los requisitos para optar a la acreditación de DOCTORADO EUROPEO**.

Y para que así conste, firmo la presente en Las Palmas de Gran Canaria, a **13 de noviembre de 2015**.

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

CONTRIBUTIONS TO NON-ITERATIVE SUPER-RESOLUTION ALGORITHMS ENABLING EFFICIENT REAL-TIME FPGA IMPLEMENTATION FOR RESOLUTION ENHANCEMENT OF VIDEO SEQUENCES

Tomasz Szydzik
Institute for Applied Microelectronics
University of Las Palmas of Gran Canaria

This dissertation is submitted for the degree of
Doctor of Philosophy

Las Palmas de Gran Canaria, November 2015

To my loving family

Abstract

We live in a reality dominated by visual content, more specifically, by high resolution visual content. High resolution images are of crucial importance in a number of areas centered on two main applications, namely the improvement of pictorial information for human interpretation, and robust automatic machine vision. In order to fit with the new level of expected visual quality, the low resolution contents quality must be augmented in a process called resolution enhancement. The technology-based solutions that increase the spatial resolution by either reducing the pixel size and increasing the number of pixels per area unit or increasing the chip size to accommodate more pixels in practice degrade the image quality and/or increase the implementation costs to such a point that they are no longer considered to be efficient.

With the rise of easily accessible computation power, spatial resolution enhancement using soft-processing became a promising alternative. From all the techniques of carrying out the algorithmic spatial enhancement super-resolution algorithms distinguish themselves by the unparalleled potential of improvement in the resulting image quality they can achieve. Super-resolution algorithms take advantage of the fact that aliasing, arising in digital images due to limitations of the system sensors and/or optics, contains extra high-frequency information with additional details about the scene. Super-resolution algorithms extract this information and use it to reconstruct a higher-resolution image. This process is computationally expensive requiring hardware implementations in order to reach real-time execution. However, most of the contributions presented over the last 40 years address only the algorithmic level and software implementations without taking into consideration the challenges of hardware implementation. Even the state-of-the-art implementations have their performance limited by memory storage requirements or memory access latency resulting in sub-optimal output quality.

The herein presented doctoral dissertation has tackled the key contemporary challenges faced at the time of developing hardware implementations of the SR techniques for video sequences, namely: the delivery of real-time execution capability, provision of high implementation efficiency, and preservation of software-level quality of the super-resolved image

observed at the output. The ultimate goal of this thesis has been to provide a hardware implementation that is characterized by the above-mentioned properties.

The base software implementation did not consider hardware implementation challenges and was characterized by high use of memory resources precluding its direct implementation in FPGA technology using only on-device memory. With the goal of overcoming these weaknesses, we have proposed a modified execution flow that operates using finer-grain elements, applying super-resolution using a single macroblock execution context. The results of the proposed modifications lead to significant memory occupancy reduction at the expense of increased memory traffic. The computed minimal and maximal value of the expected factor of reduction in memory occupancy associated with the switch to the macroblock-level flow was within the range of 3.5 to 16, depending on the algorithm parameter values. The computed minimal and maximal value of the expected factor of increase in memory traffic associated with the change was within the range of 1.1 to 16.9.

For the needs of the planned hardware implementation a high-level implementation methodology has been developed. The established methodology defines a hierarchy of levels of abstractions, in which each of the models in the hierarchy is obtained based on the model being one level higher in the hierarchy. This renders the models tightly-coupled, facilitating inter-model propagation of modifications. The use of intermediate representation coded using generic SystemC increases design portability by allowing high-level synthesis to target a range of HDL languages and technologies from the same higher-level description. The level of details of the provided methodology description facilitates the reuse of the established flow in implementations of other similar algorithms.

The final architecture has met the targeted performance of 24 fps with operating frequency of 109 MHz using the *xc5vf70t-1* FPGA device (Xilinx Virtex5 technology). The carried out comparison with the state-of-the-art has yielded satisfactory results as the observed logical resources occupancy for the proposed system was up to 5 times lower than the one reported for the state-of-the-art mapped using the same target FPGA technology. The hardware implementation synthesis results: (i) have demonstrated the capability of reaching real-time performance in FPGA technology, while preserving the quality of the super-resolved images at the level offered by the software implementation, and (ii) have proved the correctness of the presented algorithmic level changes and the established implementation methodology. The observed synthesis results justify the claim that the proposed implementation contributes to the advancement of the state-of-the-art by implementing the MB-level processing flow (typical flow of image/video compression algorithms) and by delivering higher implementation efficiency.

Resumen

Vivimos en una realidad dominada por los contenidos visuales, y más específicamente, por la alta resolución de los contenidos visuales. Las imágenes de alta resolución son de una importancia crucial en varios campos de investigación centrados en torno a dos aplicaciones principales: la mejora de la información visual para la interpretación humana, y el incremento de la robustez de la visión artificial automática. Con el fin de alcanzar los nuevos niveles de calidad visual demandados, la calidad de los contenidos de baja resolución puede ser aumentada mediante un proceso denominado mejora de la resolución de la imagen. Las soluciones de fuerte base tecnológica que aumentan la resolución espacial, ya sea reduciendo el tamaño de píxel y aumentando el número de píxeles por unidad de área, o bien aumentando el tamaño del chip para dar cabida a más píxeles, en la práctica degradan la calidad de la imagen y/o aumentan los costes de la aplicación hasta el punto de no considerarse ya eficiente.

Con el aumento de la potencia actual de cálculo y de su fácil acceso, la mejora de la resolución espacial usando procesamiento software se convierte en una alternativa prometedora. De todas las técnicas que llevan a cabo la mejora de la resolución espacial, caben destacar los algoritmos de súper-resolución por el potencial incomparable de mejora en la calidad de la imagen resultante que pueden lograr. Los algoritmos de súper-resolución aprovechan el hecho de que el aliasing existente en las imágenes digitales, que surge debido a las limitaciones espectrales de los sensores y/o a las limitaciones de la óptica del sistema, contiene información de alta frecuencia que permite incorporar detalles adicionales sobre la escena. Los algoritmos de súper-resolución extraen dicha información y la utilizan para reconstruir una imagen de mayor resolución. Este proceso es computacionalmente costoso y requiere de implementaciones hardware con el fin de alcanzar velocidades de ejecución en tiempo real. Sin embargo, la mayor parte de las contribuciones presentadas en los últimos 40 años sólo abordan el nivel algorítmico y las implementaciones software sin tener en cuenta los retos de una implementación hardware. Incluso las implementaciones del estado del arte encuentran su rendimiento limitado por los elevados requisitos de almacenamiento de memoria y/o por la elevada latencia de los accesos a memoria, lo que resulta en una

calidad de salida sub-óptima. La presente tesis doctoral aborda los principales desafíos encontrados actualmente a la hora de desarrollar implementaciones hardware de las técnicas de SR para secuencias de vídeo, dónde cabe destacar las siguientes: la capacidad de ejecución en tiempo real, una alta eficiencia de la implementación y la preservación de la misma calidad de salida de la imagen súper-resuelta que la obtenida con el software. El objetivo final de esta tesis ha sido proporcionar una implementación hardware caracterizada por las propiedades antes mencionadas.

La implementación del algoritmo base en software no consideró en ningún momento desafíos cruciales de la implementación hardware, caracterizándose por una alta utilización de recursos de memoria que impiden su aplicación directa en tecnologías basadas en FPGAs utilizando solamente la memoria disponible en el dispositivo. Con el objetivo de superar estas limitaciones, se ha propuesto un flujo de ejecución modificado que funciona con elementos de grano más fino, aplicando la súper-resolución en el contexto de una ejecución centrada sólo en macro-bloques. Los resultados de las modificaciones propuestas conducen a una reducción significativa de la ocupación de memoria a expensas de un aumento del tráfico de memoria. El valor mínimo y máximo calculado del factor de reducción en la ocupación de la memoria asociada con el cambio del flujo de ejecución a nivel de macro-bloques está entre 3,5 y 16, dependiendo de los valores de los parámetros del algoritmo. Al mismo tiempo, el valor mínimo y máximo calculado del factor de aumento en el tráfico de memoria asociado con dicho cambio de flujo está entre 1,1 y 16,9.

Para cubrir las necesidades de una implementación hardware planificada se ha desarrollado una metodología de diseño de alto nivel. La metodología establecida define una jerarquía de niveles de abstracción, en la que cada uno de los modelos de la jerarquía se obtiene basándose en el modelo de nivel superior de dicha jerarquía. Esto hace que los modelos estén fuertemente ligados entre sí, lo que facilita la propagación de las modificaciones entre modelos de la jerarquía. El uso de una representación intermedia codificada usando SystemC genérico aumenta la portabilidad del diseño al permitir la síntesis de alto nivel dirigida a una serie de lenguajes HDL y de tecnologías a partir de la misma descripción de alto nivel. El nivel de detalles proporcionado por la descripción usando esta metodología facilita la reutilización de otros flujos establecido en las implementaciones de algoritmos similares.

La arquitectura final alcanzó las prestaciones deseadas de 24 fps con una frecuencia de operación de 109 MHz utilizando el dispositivo FPGA xc5vj70t-l (Xilinx: tecnología Virtex5). La comparación realizada con el estado del arte ha resultado satisfactoria, dado que la ocupación de los recursos lógicos del sistema propuesto es hasta 5 veces menor que

la reportada para el estado del arte usando el mismo tipo de tecnología FPGA. Los resultados de síntesis de la implementación hardware: (i) han demostrado la capacidad de alcanzar prestaciones en tiempo real usando tecnología FPGA, preservando al mismo tiempo la calidad de las imágenes de salida súper-resueltas al mismo nivel que el ofrecido por la referencia software, y (ii) han demostrado la fidelidad de los cambios realizados a nivel algorítmico y la validez de la metodología de implementación establecida. Los resultados de síntesis obtenidos suponen una contribución al estado del arte gracias a la implementación del flujo de procesamiento a nivel de MB (flujo típico de los algoritmos de compresión de imagen/vídeo) lo que supone una mayor eficiencia de la implementación.

Acknowledgements

I would not have been able to complete this journey without the aid and support of countless people over the past four years. Foremost, I would like to express my gratitude to my supervisors, Prof. Gustavo I. Marrero Callicó and Prof. Antonio Nuñez Ordoñez, who have been greatly supportive and have guided me during this research and while writing this dissertation, offering constructive comments and warm encouragement.

Over the years, I have received funding from several entities, which have supported me while I completed my research. I would like to thank Barco Electronic Systems S.A., Movidius Ltd., the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC) and the Institute for Applied Microelectronics (IUMA) for their financial support. In particular I would like to thank David Moloney and Luigi Albani for their generous support and for sharing their immense knowledge. I'd like to thank also my fellow labmates, for all the stimulating discussions, the fun we have had, the coffees and their patience.

Last, but not least, I am deeply grateful to my parents for generously offering me support and the education that has made it possible for me to get here. Thanks also to my sister, Alina, and the rest of my family and for believing in me and instilling confidence in me.

Contents

List of Figures

List of Tables

List of Acronyms

ANSI American National Standards Institute

ASIC Application Specific Integrated Circuit

AT Approximately Timed

BCA Bus Cycle Accurate

BFM Bus Functional Model

BM Block Matching

BRAM Block Random Access Memory

BSynth Behavioral Synthesizable

CA Cycle Accurate

CCF Cross-Correlation Function

CFT Continuous Fourier Transform

CGI Computer-Generated Imagery

CIF Common Intermediate Format

CMOS Complementary Metal Oxide Semiconductor

COTS Commercial Off-The-Shelf

CPU Central Processing Unit

CT Computed Tomography

CUDA Computer Unified Device Architecture

DFT Discrete Fourier Transform

DoF Degree of Freedom

DSP Digital Signal Processor

DUT Design Under Test

EDIF Electronic Design Interchange Format

ESA European Space Agency

FL Frame-Level

FLM Functional Level Model

FPGA Field Programmable Gate Array

FR FRame

GPU Graphics Processing Unit

HD High Definition

HDL Hardware Description Language

HDTV High Definition TeleVision

HEVC High Efficiency Video Coding

HL High Level

HLA High Level (of) Abstraction

HLS High Level Synthesis

HR High Resolution

I/O Input/Output

IBP Iterative Back Projection

IP Intellectual Property

IR Image Registration

IUMA Institute for Applied Microelectronics

LR Low Resolution

LT Loosely Timed

LUT Look-Up Table

MAD Mean Absolute Difference

MAD Multiply and ADd

MAE Mean Absolute Error

MAP Maximum A Posteriori

MB Macroblock

MBL Macroblock-Level

ME Motion Estimation

ML Maximum Likelihood

MPSoC MultiProcessor System on Chip

MRI Magnetic Resonance Imaging

MSE Mean Square Error

MSSIM Multi-scale Structural SIMilarity

N/A Not Available

NTSC National Television System Committee

NUGP Non-Uniform Grid Projection

NUGPA Non-Uniform Grid Projection Algorithm

OSCI Open SystemC Initiative

PCA Pin Cycle Accurate

PDC Pixel Difference Classification

PLD Programmable Logic Device

POCS Projection Onto Convex Sets

PSNR Peak Signal-to-Noise Ratio

QCIF Quarter Common Intermediate Format

QVGA Quarter Video Graphics Array

RAM Random Access Memory

RB Reorder Buffer

RF Reference Frame

RGB Red, Green, Blue

ROI Region Of Interest

ROM Read Only Memory

ROV Remotely Operated Vehicle

RS Reference Structure

RT Register Transfer (accurate)

RTL Register Transfer Level

SA Search Area

SAM System Architectural Model

SAR Search Area Radius

SFW Sliding Frame Window

SLM System Level Model

SoC System-On-Chip

SPM System Performance Model

SR Super-Resolution

SRIR Super-Resolution Image Reconstruction

SSDA Sequential Similarity Detection Algorithm

SSIM Structural SIMilarity

STORM Stochastic Optical Reconstruction Microscopy

SVC Scalable Video Coding

SXGA Super Extended Graphics Array

TF Timed Functional

TLM Transaction Level Modeling

TMAC Total Memory Access Count

TMR Total Memory storage Requirements

TVM Transaction Verification Model

ULPGC University of Las Palmas de Gran Canaria

UT UnTimed

VGA Video Graphics Array

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

VLIW Very Long Instruction Word

WVGA Wide Video Graphics Array

WXGA Wide Extended Graphics Array

XGA Extended Graphics Array

XST Xilinx Synthesis Technology

Chapter 1

Introduction

1.1 Challenges

We live in a reality dominated by visual content, more specifically, by *High Resolution* (HR) visual content. High resolution images are of crucial importance in a number of areas centered on two main applications, namely the improvement of pictorial information for human interpretation, and robust automatic machine vision. In order to fit with the new level of expected visual quality, the *Low Resolution* (LR) contents quality must be augmented in a process called resolution enhancement. Image resolution describes the details contained in an image: the higher the resolution of an image, the higher the amount of details it contains. The resolution of a digital image can refer to: pixel resolution, spatial resolution, spectral resolution, temporal resolution, and/or radiometric resolution. In this context, this PhD Thesis focuses on contributing to the improvement of spatial resolution.

(A) Considering a digital image as composed of *picture elements* (*pels*), spatial resolution of digital images is expressed (or measured) as the total number of these elements and their density understood as the number of the elements per area unit. In the context of digital images, a singular discrete pel is usually referenced as a *pixel*. In practice, the spatial resolution of most of the imaging systems is limited by the properties of the used sensor rather than the optical diffraction. The most intuitive solution to increase spatial resolution is to reduce the pixel size (i.e. increase the number of pixels per area unit) by sensor manufacturing techniques. However, as the pixel size decreases so does the number of photons captured and the amount of light being integrated by each pixel per time unit. In consequence the sensor is more susceptible to shot noise, that can degrade the image quality severely. The acceptable value of the shot noise

imposes a technology dependent limitation on the minimal pixel size. In case of 0.35 μm CMOS process the optimal pixel size is estimated at about 40 μm^2. This size has been already reached by the current image sensor technology. On the other hand, increasing the chip size to accommodate more pixels, and thus augment the spatial resolution, has its own limitations, being the most important the corresponding increase in sensor area and capacitance. Increased capacitance results in slower transfer rates limiting the temporal resolution and system applications. Increasing area size results in higher fabrication costs and higher power dissipation. Along with the arising requirement of higher precision optics, these properties significantly increase the production costs.

The above-mentioned technology-based solutions are considered to be not cost-efficient. Moreover, their application does not provide solutions for legacy content already captured with old technology. Therefore, a new approach toward increasing spatial resolution is required to overcome the limitations of the sensors and manufacturing technology. With the rise of easily accessible computation power, spatial resolution enhancement using soft-(post)processing became a promising alternative, especially if the computational resources can be shared with other processes.

(B) The goal of algorithmic spatial enhancement is to produce a HR approximation from the known LR input. There are many methods of carrying out this process that differ in the way they use the LR pels to estimate the values of the HR ones. The pels that make up the HR representations can be: (ii) a direct copy of a pel from the LR image or (ii) an estimated value. There is a clear trade-off between the computational cost of the estimation step and the possibility of providing better results (one that would approximate more closely the tentative 'real' HR representation of the LR image). In most cases, upscaling is carried out by means of interpolation, where the HR pels values are estimated based on the local neighborhood pels values (non-adaptive interpolation) and some characteristics of the region that the pel forms part of (adaptive interpolation). Such approaches prefer inexpensive implementation over the quality of results.

A smart alternative is to use *Super-Resolution* (SR) algorithms which analyze the LR content (image or a sequence) in search of clues (in-image patterns and similarities) on how to perform a better estimation. The mathematical basis of this kind of algorithms is a generalization of the Nyquist sampling criterion. This generalization establishes that it is possible to reconstruct a signal from several sampling sets in

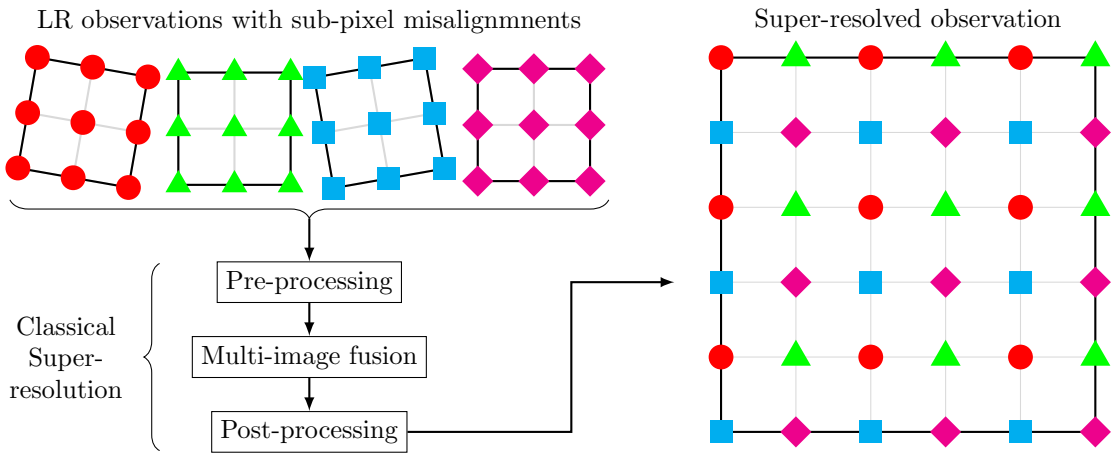


Fig. 1.1 Execution flow of the multi-image super-resolution image reconstruction based on sub-pixel misalignments exploitation.

presence of aliasing, if the sampling periods prove to be different for every sampling set. In case of a video sequence or a successive acquisition of several photographic pictures, it is also possible to widen the space of the search of the 'clues' to include other images and to take advantage of the information on how they relate among one another. In this way, a significant improvement in the resulting image quality can be achieved over the sensor resolution used to acquire the images. The SR approaches that are of most relevance to this thesis are the fusion-based multi-frame methods. These methods enhance the spatial resolution and the observed quality by gathering 'unique' data from a set of images in the spatial-temporal domain and integrating the collected information (when possible) in a new quality-improved super-resolved image. In this process, presented in Fig. 1.1, the quality of results heavily depends on the availability of 'unique' data. Typically, it is sufficient to provide several images that have been acquired with small spatial sub-pixel shifts between the captures. This can be achieved for instance by recording a video sequence at high frame rates with a hand-held camera.

In the context of limited spatial resolution, super-resolution comprises the methods that allow increasing the spatial resolution beyond the native resolution of the sensor and the enhancements offered by interpolation methods. However, the computational complexity introduced by additional processing required for the SR process to provide results is very high. In fact, it is so high that, for the time being and to the best of our knowledge, none of the available software implementations can carry out the

SR enhancement in real-time for frame formats usually found in consumer electronics applications (QCIF and higher). As many data processing algorithms, also most SR methods offer great parallelization opportunities. Custom and many-core hardware implementations of SR methods are expected to be able to exploit the existing parallelism and greatly improve the observable performance. Nevertheless, hardware implementations face problems and limitations of their own. Not all algorithms can be efficiently mapped onto hardware, leading to inefficient use of resources that results in limited performance. Even the implementation of more hardware-friendly algorithms is likely to require significant trade-offs in order to make the implementation feasible. A common case is the increasing of the observable execution performance at the cost of sub-optimal outcome quality. Therefore, the ability of providing hardware implementations of most of the SR algorithms capable of producing satisfactory output quality with real-time performance still remains a challenge.

(C) In a typical design flow the reference algorithm is first analyzed, architecturally constrained, and then used to create a register-level and pin-accurate description in hardware design language. In most hardware implementations the whole system functionality is first prototyped in software. This somewhat intermediate step is used to test the algorithm functionality and provide a high-level of abstraction reference to be used in validation and testing. A common case is that the required register-level/pin-accurate description is created from scratch. The designer manually codes the system directly from the provided functional specification and algorithmic description. The available base code is used only as a source of information on design functionality and the reference in the process of verification. This approach leads to a highly optimized, low-level implementation that requires a significant amount of time for system creation and validation. Due to non-propagative relationship between the software and hardware descriptions, implementation of changes introduced to the base software, as well as evaluation of their impact on the system performance, is complicated and requires significant amount of effort and time.

The well-known drawbacks of direct manual implementation are becoming more and more evident and hard to deal with as the contemporary electronic systems are shifting towards *Systems-on-Chip* (SoCs) and in particular *Multi-Processor SoCs* (MPSoCs). These systems are not only more complex but also call for solutions of additional issues, especially in the context of the very competitive industry of consumer electronics. Among most important challenges faced by electronic systems design methodolo-

gies, the *European Space Agency* (*ESA*) [Age08] lists the following ones: (i) reduced time-to-market, (ii) increased verification complexity that scales with the system components and the need to use vendor-specific tools, (iii) huge design space to explore, (iv) highly complex software development, and (v) the need for methodologies enabling concurrent design of hardware/software. All these issues can be alleviated by using a set of tiered *High-Level Abstractions* (HLA) during system specification and the initial phases of electronic system design. Thus, recently, an emerging trend in the design and verification of electronic systems is to provide not one, but a set of progressively more complex system models. In order to bridge the gap between the algorithmic and *Hardware-Description Language* (HDL) descriptions, the automated *High-Level Synthesis* (HLS) operates on an intermediate representation that imposes limitations on the set of software language constructs that are allowed and introduces additional ones to represent the missing HDL constructs. The intermediate representation can be simulated at much higher speeds than the low-level high-accuracy models. The execution flow and most of the software code remain unchanged during the transformation to the intermediate representation. The coherency between the representations, allows rapid system modeling, and facilitates propagation of changes introduced to the base code onto the hardware implementation, as the HDL is generated automatically from the intermediate representation by an HLS tool.

Although the multi-tiered approach alleviates many of the issues associated with the manual conversion, it faces some issues of its own, most importantly:

(i) The creation of the intermediate representation requires additional effort. The learning curve is somewhat steep.

(ii) The HDL created by the synthesis tool is human illegible, this significantly limiting the possibility of manual introduction of post-synthesis changes and tweaks.

(iii) Even the smallest change made to the intermediate description requires a synthesis re-spin.

(iv) The relation between the input and the output of the synthesis is somewhat non-deterministic. That is, small changes in the input may result in significant changes in the created HDL. The observed changes are not always intuitive.

(v) The quality of HDL is somewhat inferior to the manually written one.

Thus, designing the implementation flow so that it provides a hierarchy of tightly-coupled models and enhances the probability of a successful implementation of func-

tionality in a cost-effective way while assuring high-level quality of the results continues to be a challenge.

1.2 Research motivation

Super-resolution image reconstruction (SRIR) comprises algorithms that attempt to reconstruct the high resolution image corrupted due to the limitations of the imaging system. Typically, these limitations arise in the system sensor and/or the optics, causing aliasing to occur as the sampling process does not meet the requirements defined by the Nyquist sampling theorem. Aliasing in digital images is often considered as a nuisance and (both optical and digital) filters are designed to avoid aliasing in digital cameras. However, aliasing also contains extra high-frequency information with additional details about the scene. Super-resolution algorithms extract the information present in the aliasing effect to reconstruct a higher-resolution image [Mil10].

In the context of limited spatial resolution, super-resolution allows to increase the spatial resolution beyond the native resolution of the sensor and the enhancements offered by interpolation methods. The ability of additional detail extraction offered by the SR algorithms, when compared with single frame interpolation (the usual case), greatly improves the results of the process of spatial image augmentation, leading, where possible, to significant objective image quality enhancement expressed in the increase of *peak-signal-to-noise ratio* (PSNR). The above strengths of SR techniques lead to a wide range of possible applications. Among others, SR techniques are used in the following applications:

- Surveillance [RRYB13, ZH12, ATB12]: in this scenario use of SR provides savings in system and storage costs. A typical use case consists of image loading ('freezing') and then applying resolution enhancement to a selected *region of interest* (ROI). Most common use case is automated recognition (e.g. vehicle license plate checking, face recognition, etc.).

- Geographic or space or underwater imaging: uses SR for resolution and quality enhancement of images or acoustic signals acquired in environments that intrinsically exhibit unfavorable capture conditions or limited bandwidth: e.g. underwater imaging carried out using *remotely operated vehicle* (ROV) [YU14, QdLCC+14, CY13, CYX+11], geo-satellite images of a determined geographic area [XZZ14, SKC+13, VCB+10, ESHEK12, GSP86] or open space images captured by space rovers or probes.

- Medical imaging [OII+14, vABVG+14, AEH+14, VCT+10, Nik10]: there is a wide scope of uses as these systems deployment set-ups allow for the captures to be easily taken with (forced) sub-pixel misalignments and do not demand real-time requirements. Used in a multitude of applications, most importantly in *Computer Tomography* (CT), *Magnetic Resonance Imaging* (MRI), medical ultrasound (a.k.a. ultrasonography) and super-resolution microscopy (e.g. *Stochastic Optical Reconstruction Microscopy*, STORM).

- Consumer electronics: there is a wide range of applications mainly built around upscaling and conversions (legacy content). A typical application is the conversion between different video standards [Mal06, NEC09, Goh13, KSS+12, GSH+11] e.g. NTSC (National Television System Committee) and HDTV (High Definition TeleVision). Another one is the SRIR setup with sensor-shifting to capture high resolution photos. By capturing and combining multiple captures with fixed (thus known) subpixel movement it is possible to extend the resolution beyond the sensor limits, provide *true color* (interpolation-free) demosaicing and moire free capture (e.g. Hasselblad Multi-Shot Cameras series [Has14b, Has14a]). Early adopters of this technology in digital cameras include Olympus [Oly14], Ricoh/Pentax [Ric14] and Apple [App14].

The majority of the above presented applications do not pose real-time constraints on the execution speed. Thus, typically these applications requirements can be satisfied with provision of a software implementations of SR that perform processing 'off-line'. Nevertheless, when introduced, real-time capabilities are expected to broaden even further the range of possible applications and result in enhanced usability and user experience in the cases that do not intrinsically require them.

Due to its vast applications and ill-posed nature, super-resolution has been a very active area of research since its introduction in 1974. The most important contributions that are of relevance to this work are presented in Fig. 1.2. Most of the contributions presented over the last 40 years address only the algorithmic level and software implementations without taking into consideration the challenges of hardware implementation. Lack of contemplation of hardware considerations in the process of algorithm design complicates its hardware implementation. This results in designed algorithms having high resource requirements precluding their efficient implementation without prior algorithmic-level modifications. Additionally, hardware implementation has to consider the existing trade-off between the execution speed and the resources required to obtain it. In order to take advantage of the available

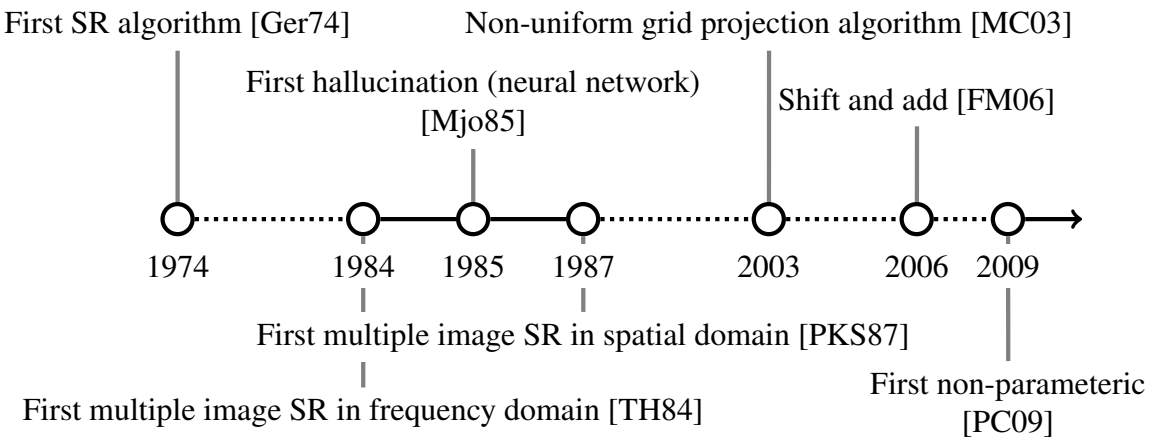


Fig. 1.2 Excerpt of the time line of proposals of SR algorithms.

parallelism all of the data would be required to be readily available from memories, the execution units have to be replicated/pipelined, etc. On one hand the processing has to be parallelized, on the other hand achievable parallelism is limited by the memory required for its exploitation. All of the above has led to a point where it is not guaranteed that the proposed contributions can be easily propagated onto hardware. Hence, there is a growing need of provision of know-how or procedures and recommendations on how to consider the implications of possible hardware implementations at the time of algorithm definition in order to help to close the ever-growing gap between software and hardware implementations.

As pointed out, software based implementations are not capable of providing HR real-time performance when mapped on available technology. The currently available hardware implementations with real-time performance provide an output quality lower than their software implementation counterparts. Even the state-of-the-art implementations using *Field-Programmable Gate Array* (FPGA) technology have their performance limited by memory storage requirements and/or memory access latency. Thus, only a limited number of iterations or reference frames are implemented, leading to sub-optimal output quality. Based on our experience with compression algorithms we find much similarity in the nature of processing carried out by SRIR and compression algorithms and the problems faced by their implementations. Both types of processing deploy complex algorithms that work with large amounts of data and require hardware implementations to process them in real-time. Moreover, many of the compression algorithms pose hard real-time constraints on the execution time. Hardware implementations of compression algorithms can provide solutions for how to implement algorithms that process images so that their hardware implementations can achieve real-time performance.

This doctoral thesis provides an example of the required modifications that alleviate the aforementioned problems and therefore could result in the future hardware implementations of super-resolution based resolution enhancement, becoming efficient enough to allow their use in mainstream consumer electronics. In this line, we propose an SRIR implementation based on a non-iterative version of the *Non-Uniform Grid Projection Algorithm* (hereafter the *NUGP algorithm* or *NUGPA*). Direct implementation of the reference version of this algorithm is considered unviable due to high memory requirements. In this work we evaluate and implement modifications to the NUGPA execution flow so that its processing can be applied in a way that only requires a singular macroblock execution context to be readily available from memories. This approach has achieved so much success in the context of compression algorithms implementation that we hope to replicate it for the needs of SRIR. This change has some significant implications on the overall algorithm execution flow that have to be considered at the time of implementation. Thus, the algorithm is modified in such ways that not only provide real-time performance and observed quality of the super-resolved output that matches the level of the software state-of-the-art, but also facilitate efficient hardware implementation. Only by providing all of these characteristics the high fidelity resolution enhancement can find its way out of research facilities to everyone's home. The lessons learned during the process of NUGPA implementation are expected to contribute to the understanding of the hardware implementation challenges and limitations knowledge by the software/algorithmic designers. In the long term, this could contribute to the process of providing more hardware-aware implementations, simplifying the process of hardware implementation and most likely leading to increased efficiency and performance.

1.3 Objectives

The ultimate goal of this thesis is to provide solutions at the algorithmic and architectural level to the fusion-based SR algorithm developed by the *Institute for Applied Microelectronics* (IUMA) of the *University of Las Palmas de Gran Canaria* (ULPGC) that would allow reaching real-time execution when implemented in FPGA devices. In order to achieve this goal, this doctoral thesis is focused on reaching the following objectives:

1. Contextualize this work by exploring the state-of-the-art of super-resolution methods and their hardware implementations. In this process, the state-of-the-art of SR algorithms and a complete classification of the available methods will be described. The study is carried out in order to: (i) highlight strengths and weaknesses of the NUGPA approach in comparison with other approaches, and (ii) justify the decision

of using NUGPA as the base algorithm for hardware implementation capable of real-time execution. Following, the known approaches to hardware implementations of SR in FPGAs will be presented. At this point it is important to pay attention: (i) to the parallelization strategies exploited, (ii) techniques applied for improving the final performance of the system, (iii) avoidance of system bottlenecks, and (iv) required trade-offs. Based on the obtained knowledge, an adequate development strategy will be developed.

2. Provide algorithmic contributions to facilitate hardware implementation of the NUGPA super-resolution kernel (SRK) using only on-chip memory. The available version of the NUPGA does not contemplate hardware implementation in programmable logic devices. Similarly, the available baseline software was not implemented with the view of limiting the use of resources. Thus, in order to increase the chances of a successful and efficient implementation using FPGA devices the algorithm data flow should be revised and modified if needed.

3. Create a design flow and implementation methodology that increases the chances of obtaining an efficient implementation. The methodology should consider the fact that the SR algorithm and its software implementation are constantly under development by facilitating rapid and robust propagation of the modifications made at the algorithmic level to the hardware implementation. The created design and implementation methodology should be general enough to be able to be applied to other similar algorithms. At the same time, the description should be specific enough to remove any doubts on the steps required to provide high quality results. All these requirements lead to a combination of *Electronic System Level* (*ESL*) and *Register Transfer Level* (RTL) synthesis flow.

4. Demonstrate the viability of reaching real-time performance by providing a baseline implementation of the NUGPA SR kernel in a FPGA device. Apart from the performance goal, the implemented design must be able to fulfill additional expectations of which the most relevant are listed below:

   (a) Efficiency: in this specific case understood as minimization of resources (logical and memory) occupancy while providing a solution that meets the targeted performance in terms of execution speed. The hardware implementation is expected to be successful in preserving the observed super-resolved image quality at the level offered by software implementations.

(b) Portability: this characteristic is related to the fact that we expect the implementation to implement solutions and mechanisms that facilitate migration to a different technology (different FPGA family/vendor) without having to redesign it completely from scratch. Having in mind that FPGA implementations are frequently a required prototyping step for custom hardware, this migration should also help in the road towards ASIC and SoC implementation targets.

1.4 Organization of this thesis

The present document is structured into seven chapters that somewhat follow the order of the objectives given above. The first three chapters (chapters 1–3) are dedicated to presenting the problematic, the state of the art and the motivations for developing this Doctoral thesis. These chapters provide information required to fully understand the issues being raised in this thesis by readers who might be not familiar with the topic of super-resolution, its current state-of-the-art, and the NUGP algorithm in particular. The introductory nature of these chapters serves to set the scene for presenting the main contributions of this work. Each of the chapters of this work is considered to be self-confined, that is, understanding of the main contributions presented in a chapter does not require previous reading of any of the chapters preceding it. Thus, readers interested in particular objectives of this thesis (modeling, methodology, or hardware implementation) may start reading directly from the chapter of their choosing.

Chapter 2: IMAGE ENHANCEMENT USING SUPER-RESOLUTION

This chapter introduces the basic concepts of the super-resolution process and the state-of-the-art of the SR methods. The presented data allow contextualizing the non-iterative non-uniform grid projection algorithm within the state-of-the-art, show its weaknesses and strengths, and justify its choice as a base for hardware implementation. The final objective of this chapter is to describe the available approaches to hardware implementations of SR in FPGAs, exposing their weaknesses, highlighting the scarce number of such implementations and the need of higher efficiency in terms of resources occupancy in order to overcome the observed limitations.

Chapter 3: THE NON-UNIFORM GRID PROJECTION ALGORITHM (NUGPA)

Provision of efficient hardware implementation of any algorithm requires in-detail knowledge on the algorithm control and data flow, as well as their dependency on the

algorithm parameters configuration. This chapter focuses on presentation and evaluation of the base SR algorithm and its baseline software implementation. The description emphasizes the importance of using a single-pass (non-iterative) approach in order to facilitate hardware implementation and identifies memory occupancy as the likely factor precluding hardware implementation of the reference implementation. This chapter finalizes presenting results of a study on the quality of the super-resolved image produced by the software implementation. The presented quantitative data proves that, when appropriately configured, the base implementation is capable of providing output whose quality of results is not only better than interpolation but also on-par or better than the one provided by state-of-the-art hardware SR implementations.

Chapter 4: PROPOSED SUPER-RESOLUTION ALGORITHM

The NUGPA algorithm was expected to pose memory requirements that were prohibitively high for a hardware implementation meeting the proposed objectives. In order to tackle this problem the algorithm dependency on *Frame-Level* (FL) buffers had to be eliminated. The focus of this chapter is the set of modifications of the NUGPA execution flow proposed in order to facilitate its hardware implementation. The chapter presents models and quantitatively evaluates memory occupancy and traffic of both the reference and the proposed execution flows. The obtained data show that the proposed modifications are expected to lead to reduction in memory occupancy at the cost of increased memory traffic. The chapter concludes presenting an example of system bottleneck identification data and then algorithmic-level transformation in order to optimize the system.

Chapter 5: IMPLEMENTATION METHODOLOGY

Most of the contributions to the state-of-the-art of SR methods focus on introducing algorithmic-level contributions that are aimed at improving the observable output quality. The effectiveness of these proposals is usually proved mathematically or by means of software implementations. The latter, in most cases, are not capable of applying the processing in real-time. Meeting real-time constraints usually requires the algorithm to be mapped onto hardware. With the ever-growing complexity of the algorithms the organization of the process of mapping becomes a challenge itself. In our implementation we have opted for using a tiered hierarchy of abstraction models. Each of the defined tiers of abstraction is accompanied by a verification set-up. The use of multiple models of abstraction that are progressively more accurate allows re-

ducing the gap separating the created system descriptions and greatly facilitates the propagation of changes from one tier to another. This chapter focuses on the presentation of the methodology established to carry out the process of mapping the NUGP algorithm onto FPGA technology. The used models and refinement steps carried out at each stage of mapping are organized and presented in the order that follows the design and implementation flow stages. The presented description provides details on how to set-up the verification environment in a way that leverages mixed-language simulation allowing the same environment to be used for both ESL and RTL simulations.

Chapter 6: HARDWARE IMPLEMENTATION

Definition of an implementation methodology is the first step required for successful implementation. A well-defined methodology is the stepping stone of any hardware implementation and significantly increases the probability of successful mapping. Even if provided with a well-defined implementation methodology, meeting the implementation goals is not guaranteed as the designer has yet to tackle architecture-level implementation challenges. Solutions for these challenges are constrained by the targeted system characteristics (performance, output quality, efficiency, etc.) whose implications have to be taken into consideration at the moment of balancing the implementation trade-offs. Delivery of a hardware implementation that reaches the targeted performance while maximizing the efficiency of resources utilization is not a trivial task. This chapter presents the process and the results of implementation of the NUGP algorithm operating at MB-level in FPGA technology. The description of the implementation process provides information on the system architecture model evolution, final organization, and the used abstraction levels. The tackled implementation challenges are presented in great detail along with the implemented solutions. The observed synthesis results prove that the MB-level processing contributes towards real-time implementations of the NUGPA by significantly reducing the implementations memory occupancy while preserving software-level output quality. The comparison with the state-of-the-art yields satisfactory results in terms of implementation efficiency considered as the observed gain in the quality of the super-resolved output (vs interpolation) per resource utilized.

Chapter 7: CONCLUSIONS AND FUTURE WORK

This chapter presents a recapitulation of the contributions provided in this doctoral dissertation and highlights their relevance to the super-resolution field. This document

concludes with a presentation of further research lines, which might complement and enhance some of the aspects developed in this thesis.

Chapter 2

Image enhancement using super-resolution

2.1 Introduction

Super-resolution comprises algorithms that attempt to reconstruct the high resolution image corrupted due to the limitations of the imaging system. Typically these limitations arise in the system's sensor resolution and/or the optics causing aliasing to occur as the sampling process does not meet the requirements defined by the Nyquist sampling theorem. Aliasing in digital images is often considered as a nuisance and (both optical and digital) filters are designed to avoid aliasing in digital cameras. However, aliasing also contains extra high-frequency information with additional details about the scene. Super-resolution algorithms extract the information present in the aliasing to reconstruct a higher-resolution image [Mil10].

Due to its ill-posed nature and vast applications, super-resolution has been a very active area of research since its introduction in 1974. Over the last 40 years several methods have been proposed under the umbrella of super-resolution. Most super-resolution algorithms consist of two main stages: (i) meta data registration, where the images are precisely aligned and analyzed, and (ii) image reconstruction, where the input data and meta data are used to estimate a higher resolution image. The most recent and complete taxonomy of super-resolution methods, presented in [NM14], classifies the introduced SR techniques into four main families: (i) frequency based approaches, (ii) interpolation-based approaches, (iii) regularization-based approaches, (iv) example-based approaches. The first three categories reconstruct (or super-resolve) the image from a set of lower resolution input images (multiple-image context), while the last example-based approaches are capable of achiev-

ing the same objective by exploiting the information provided by an image and a database with *a prori* attained knowledge. Recently, super-resolution has gain a lot of attention from industry which resulted in multiple software and hardware super-resolution-based solutions destined for the medical, military and even consumer electronics market.

2.2 Super-resolution basics

The general definition of the super-resolution process given in the introduction will now be refined. In particular: (i) the main characteristics of the super-resolution process will be described, and (ii) a classification of the super-resolution methods will be introduced. The *Image Registration* (IR) stage that forms a critical part of the multi-image approaches is presented in details in Section 2.3.

2.2.1 Applications of super-resolution

In this work we define the objective of super-resolution in reference to the limitations of the imaging system that the SR technique is designed to overcome. The limitations tackled by the SR methods can be classified into the following three categories: *optical*, *geometric*, and *temporal*.

2.2.1.1 Optical limitations

The diffraction-limited resolution theory was formulated by Ernst Abbe in 1873 and later refined by Lord Rayleigh in 1896. These researchers contemplated the separation necessary between two airy patterns in order to distinguish them as separate entities. They have observed and specified an upper limit —the *cut-off* spatial-frequency— below which the two point sources are considered to be *resolved* and can readily be distinguished. Beyond the cut-off spatial-frequency structural details fail to be correctly transferred into the optical image and cannot be resolved. In order to transcend the limitations of optical imaging systems a sequence of images which meet the Rayleigh criterion can be used. By manipulating the imaging conditions between the subsequent captures (i.e. using spatially-variant excitation light, gathering light over a larger set of angles around the specimen, swapping spatial frequency bands beyond the cut-off spatial frequency for one inside it (activators), changing the excitation light, etc.) it is possible to generate and transfer complementary subsets of information to different observations. Given that these observations are stochastically unique, then the extracted information can be fused, using an SR method, in order to form the final

higher-resolution image. This method allows two particles that appear to be merged, i.e. form a single point in a particular image, to be discriminated into two independent particles. Even in the case of flawless fabrication, glass-based systems resolution is still hampered by an ultimate limit in optical resolution that is imposed by the diffraction of visible light wavefronts. The use of complementary information from the context is considered to determine the exact position of the two adjacent particles even below the Rayleigh limit, effectively breaking the diffraction barrier limiting the optical imaging systems. Hence, SR methods from this category have found great success in glass-based microscopy.

2.2.1.2 Geometric limitations

In practice, the spatial resolution of most of the imaging systems is limited by the properties of the used sensor rather than the optical diffraction. The most intuitive solution to increase spatial resolution is to reduce the pixel size (i.e. increase the number of pixels per unit area) by sensor manufacturing techniques. However, as the pixel size decreases so does the number of photons captured and the amount of light being integrated by each pixel per time unit. In consequence the sensor is more susceptible to shot noise that can degrade the image quality severely. The acceptable value of the shot noise imposes a technology dependent limitation on the minimal pixel size. In case of 0.35 μm CMOS process the optimal pixel size is estimated at about 40 μm^2. This size has been already reached by the current image sensor technology. On the other hand, increasing the chip size to accommodate more pixels, and thus augment the spatial resolution, has its own limitations, being the most important the corresponding increase in sensor area and capacitance. Increased capacitance results in slower transfer rates limiting the temporal resolution and system's applications. Increasing area size results in higher fabrication costs. Along with the arising requirement of higher precision optics, these properties significantly increase the production costs. The above-mentioned 'technology' based solutions are considered to be not cost-efficient. Therefore, a new approach toward increasing spatial resolution is required to overcome these limitations of the sensors and manufacturing technology. With the rise of easily accessible computation power, spatial resolution enhancement using soft-processing became a promising alternative. In the context of limited spatial resolution, super-resolution comprises the methods that allow increasing spatial resolution beyond the native resolution of the sensor.

2.2.1.3 Temporal limitations

As aforementioned, a camera captures an image by integrating and quantizing the light coming from the scene during the exposure time. In presence of rapid movement, an objects' reflected light can be integrated over multiple sensor elements, reaching beyond its actual size. Such a case will manifest itself as a blur along movement trajectory, corrupting the quality of the image and reducing the effective resolution (capability to distinguish fine details) in the affected area. The noticed blur is stronger the faster the object moves (or the longer the exposure time is). In case of video sequences, another problem arises. Variations that occur faster than the frame rate will either not be visible or captured incorrectly. The latter case leads to false visual illusions caused by (motion-based) aliasing. A well-known example of this phenomenon is the 'wagon wheel effect' [FDC84, FD87]. This visual illusion manifests itself by the change of direction of a spinning wheel beyond a certain speed. In the temporal context, super-resolution encapsulates methods that enhance the resolution in terms of reduction of motion blur and motion aliasing using information from multiple observations. In practice, spatial SR is carried out alongside temporal SR. The enhancement in the joint space-time SR framework is carried out by using multiple sequences which have been captured at different quantization times, with different frame rates and/or spatial resolutions. Algorithms that enhance both of the resolutions are referenced as *spatio-temporal SR*.

Major advantage of the super resolution approach is that it may cost less than other hardware approaches and the existing LR imaging systems (usually cheaper) can be still utilized. This is even more important in the context of low-cost implementations, where the fundamental limits of the imaging system and the SR methods need to be understood in order to balance between expensive optical imaging system hardware and image reconstruction algorithms to meet the implementation constraints.

2.2.2 Describing the super-resolution process

Super-resolution image reconstruction algorithms aim at reconstructing the high-fidelity representation from one or more low-fidelity observations that are assumed to be corrupted by the limitations of the imaging system data. The SR methods can be classified based on many criteria. The most important criteria, in the context of video super-resolution, are presented in Fig. 2.1.

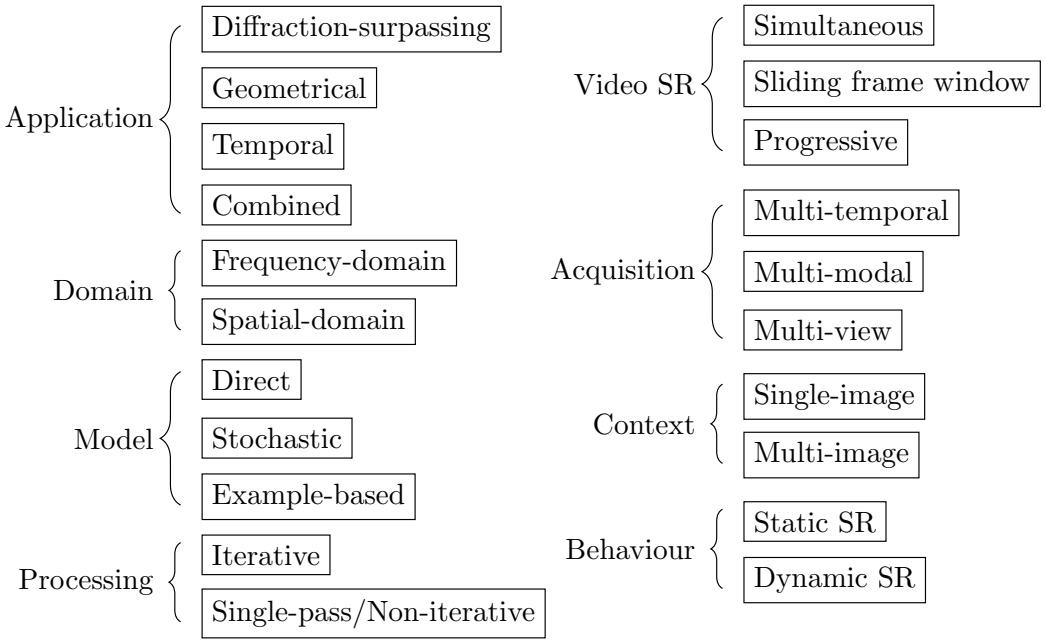


Fig. 2.1 Criteria used to classify super-resolution methods.

2.2.2.1 Processing domain

As aforementioned, the SR takes the LR observation(s) as the input data. These data are usually provided as spatially ordered direct luminance values. Depending on the domain in which the SR is carried out the SR algorithms can be divided into two families: (i) the *frequency* and (ii) *spatial* domain ones.

The SR algorithms of the former group carry out the data processing on a frequency domain representation of the input LR observation, explicitly using the aliasing that exists in each LR observation to reconstruct an HR image. These algorithms begin by transforming the LR images to the frequency domain before the super-resolved observation is estimated. Once reconstructed, the observation is transformed back to the spatial domain. Depending on the transformation employed for transforming the spatial data to the frequency domain, these algorithms are generally divided into two groups: *Fourier*-based and *Wavelet*-based methods. Wavelet-based methods are more recent but they tend to present difficulties in efficient implementation of degraded convolution filters. The Fourier-based approaches are based on the following three principles [PPK03]: (i) the shifting property of the Fourier transform, (ii) the aliasing relationship between the *Continuous Fourier Transform* (*CFT*) of an original HR image and the *Discrete Fourier Transform* (*DFT*) of LR observation, and (iii) the assumption that the original HR image is band limited. Based on these properties

TABLE 2.1 *Frequency vs. spatial domain SR [BM11, Table 1.].*

Observation model	***Frequency* domain**	***Spatial* domain**
Motion models	Global translation	Almost unlimited
SR Mechanism	Limited, linear space-invariant	Linear space-variant/-invariant
Noise model	Limited, space-invariant	Very Flexible
Degradation model	De-aliasing	De-aliasing A-priori information
Computation req.	Low	High
A-priori info	Limited	Excellent
Regularization	Poor	Excellent
App. performance	Limited	Wide
Applicability	Good	Almost unlimited

it is possible to design the system equation relating the aliased DFT coefficients of the observed LR images to a sample of the CFT of an unknown input image.

A brief review of differences for the most important features of the frequency- and spatial-domain-based approaches are tabulated in Table 2.1. The frequency-domain-based SR approaches have a number of advantages over the spatial-domain ones. First of all, they present an intuitive way to enhance the details of images by extrapolating the high-frequency information presented in the low resolution images. Moreover, the frequency-domain-based approaches have low computational complexity and offer high parallelization opportunities. Nevertheless, although being very sound and interesting from a theoretical point of view, the frequency domain methods present many constraints that limit their application in real world scenarios. The two most important factors limiting their applications are (i) the limited capability to model motion and (ii) the limited possibility of incorporate spatial *a priori* knowledge in the regularization stage. The motion models used by the frequency-domain-based methods are limited to global displacements (shifts and rotations) between the observed images. Also, the blur during the image acquisition process is modeled as linear and space-invariant. Spatial domain methods have better evolved for coping with real world problems.

2.2.2.2 Execution context

From the above-presented criteria, the most important one is the *context*, that is, the number of images that are *required* to carry out the enhancement process. Based on this criterion we divide the SR algorithms into two groups: (i) *multi-image* (or *classical*) and (ii) *single-image* (or *learning-based*) SR methods. In learning-based SR, the missing high-resolution

information is assumed to be available in the form of an *a priori* acquired knowledge that has been learned from the low-resolution/high-resolution pairs of examples.

2.2.2.2.1 Multi-image super-resolution. The classical SR operates based on the assumption that the high-frequency information is split across multiple low resolution images, implicitly found there in aliased form. Each of the low resolution observations is investigated in search of complementary information. If such information is found, the observation is considered as a linear constraint on the unknown high resolution intensity values. If enough *unique* low resolution observations are available, then the set of equations becomes determined and can be solved leading to recovery of the high resolution image. Only observations that cannot be obtained from the others are considered to have the property of uniqueness. In practice, at least $\geq scale^2$ unique representations are needed for the inversion problem to become determined. Given a sufficiently large set of observations, the increase in resolution can be of arbitrary value.

In order to work, multi-images methods require that the observations represent a unique state of the scene. Thus, in classical SR the identification and provision of unique observation is of crucial importance. In the case of using one camera, uniqueness of the observations is more probable when the observations are captured with sub-pixel misalignments. If the LR observations are shifted by integer units, then each one contains the same information, and thus they do not convey complementary details that can be used to reconstruct an HR image. However, if the LR captures have different sub-pixel misalignments from each other (and if aliasing is present), then each observation cannot be obtained from the others and thus is unique. In this case, the complementary data contained in each LR observations can be exploited to obtain an HR one.

The concept of sub-pixel misalignments exploitation in the process of unique LR observations creation is illustrated in Fig. 2.2. The origin of the sub-pixel misalignments can be either caused by controlled (e.g. orbiting satellites, jitter camera) or uncontrolled motion (i.e. vibrating imaging systems (hand-held) or local motion in the scene). Multiple views of the scene can be obtained either by moving the camera to different positions (multi-view acquisition) or by using multiple imaging systems (multi-modal acquisition). One of the most promising acquisition approaches in terms of SR is to use sensors with different pixel sizes. By using different pixel sizes the fact that pixel values are created by integrating over a different amount of light, most likely resulting in different quantization, can be exploited. Knowledge of the acquisition set-up can be used to simplify the registration stage and maximize the amount of new information.

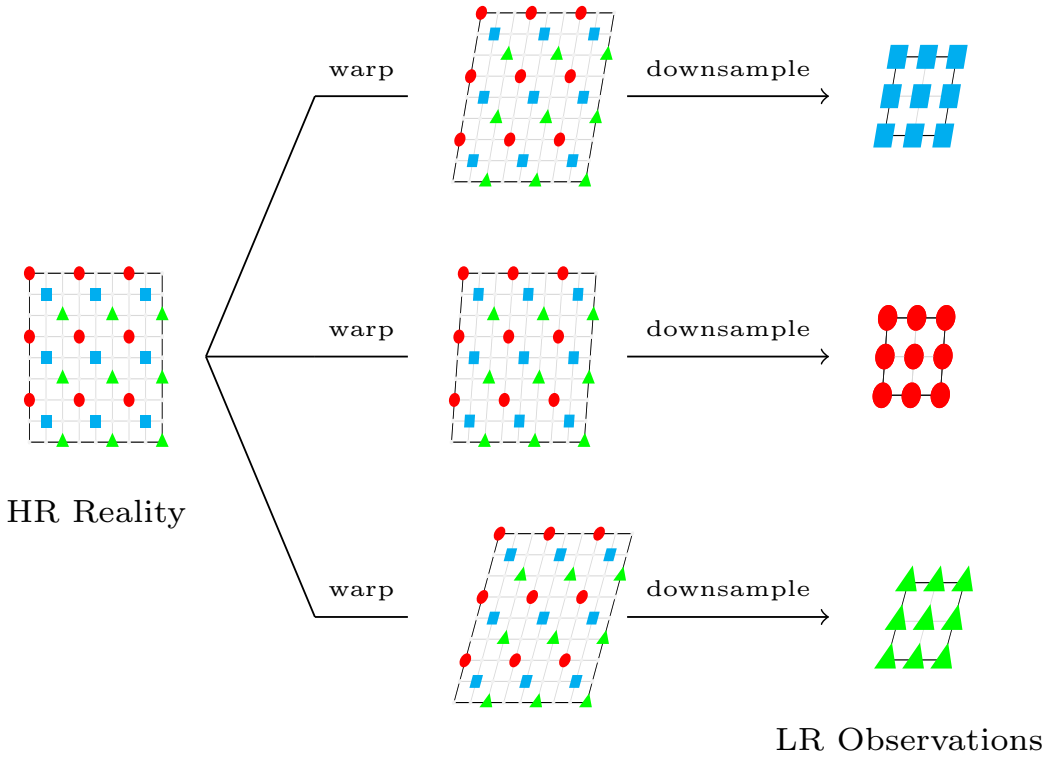


Fig. 2.2 Creation of unique observations creation with sub-pixel misalignments.

Given that the set of used images contains observations that are unique, the classical SR image reconstruction can be carried out following the flow already presented in Fig. 1.1 on page 3. The classical multiple-image SR flow comprises three main steps: (i) the *pre-processing*, (ii) the actual *reconstruction* (multi-image fusion), and (iii) the *post-processing*. It is most likely that each of the observations represents the scene under different capturing conditions of the imaging system. Depending on the observation model the set of parameters that are considered variable can include motion, blur (optical, atmospheric, and/or motion blur), zoom, multiple aperture, multiple images from different sensors, and different channels of a color image. Thus, unless the scene motions are known a prori, before being combined the images have to be registered to determine variations between one another. The two most common types of compensation for the changes between LR images are geometric registration and blur estimation. As long as the images to be used represent the same scene and can be registered in a satisfactory way, super-resolution flow allows any acquisition method to be used. The post-processing depends on the used kernel and on the

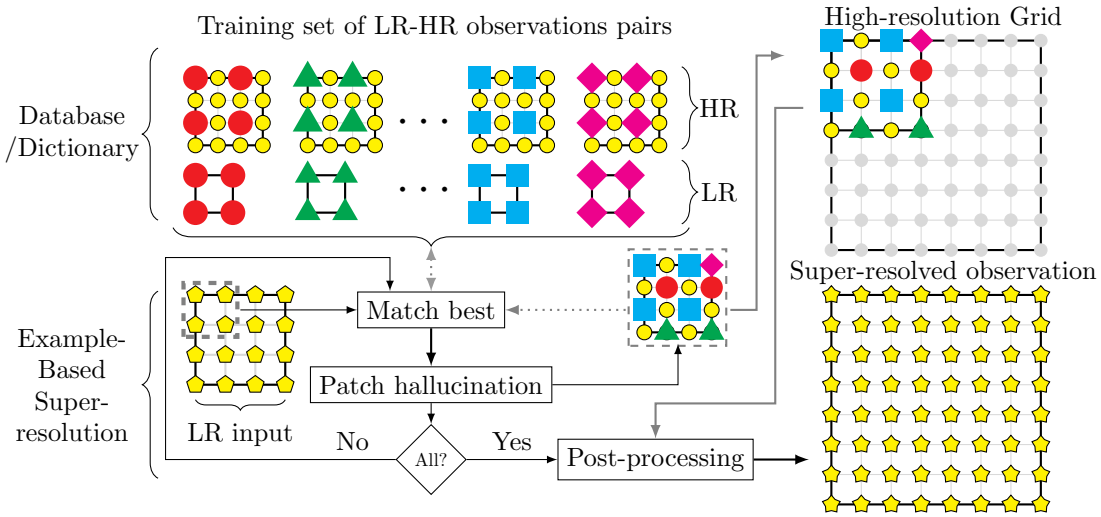


Fig. 2.3 Execution flow of the single-image super-resolution image reconstruction.

super-resolution process application. Most state-of-the art algorithms carry out post-fusion regularization and restoration.

2.2.2.2.2 Single-image super-resolution. The known drawback of the aforementioned classical SR flow is the fact that it fails to provide satisfactory results when the available observations do not fulfill the uniqueness requirement or only a single image is available. In these cases, no complementary information is available to be extracted from other observations and fused. Therefore, the missing high-resolution details that are not present in the available observations have to be estimated. Single-image methods do not look for the complementary information. Rather, they *learn* the correlations between the low and high resolution representations. Once learned, these correlations are represented by a dictionary (or a database). The characteristics of the dictionary (e.g. size, structure, search method, scalability, generalization capabilities, etc.) are of critical importance to the performance of the SR process in terms of speed and output quality.

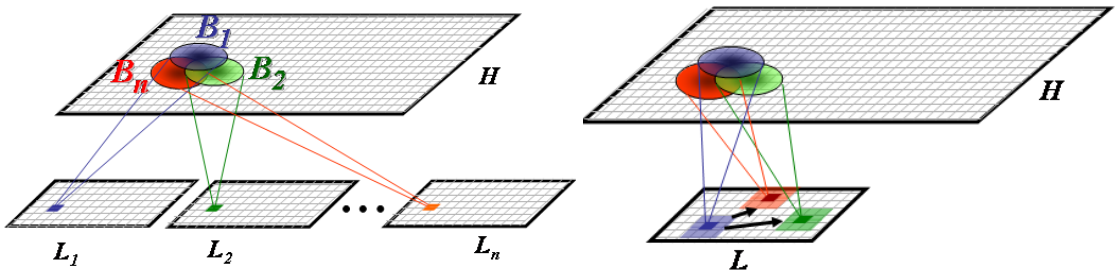
The typical single-image SR algorithm execution flow is presented in Fig. 2.3. The processing is carried out in two stages: (i) dictionary creation and (ii) image hallucination. In the former stage, a set of training images is used to learn the best ways of mapping of the local geometry of the low resolution patch space onto the high resolution patch space. This process takes a significant amount of time and hence is carried out off-line. During execution, the super-resolution stage begins with a piece-wise search in the created dictionary for an entry associated with the features recognized in the LR input. Following,

the best match(es) are identified using the method's match criteria (e.g. similarity, spatial compatibility, smoothness, etc.). One or more of these matches are used to compute the reconstruction weights that minimize the reconstruction error for the input LR patch. The weights are used in combinations with the HR data associated with the matched entries to hallucinate the HR patch. The estimated HR representation is projected onto the HR grid and optionally back-projected to feed the error estimation loop and adapt (if necessary) the processing of the subsequent patches. The initial HR representation of the entire frame is further regularized and refined using the reconstruction constraint. The reconstruction constraint's penalty terms are also derived from the training set.

2.2.2.2.3 Combined or multi-patch approaches. The advantages of the described approaches can be combined. The combined framework is based on an observation (justified statistically) that patches in a single natural image tend to redundantly recur many times inside the image, both within the same scale, as well as across different scales. Given an input image, the combined framework creates a pyramid of its scaled representations and looks for recurrence of patches within and across the scales. Patches recurring within the same image scale (with sub-pixel misalignments) can be regarded as if extracted from different observations of the same scene and used to impose constraints to the HR information. This forms the basis for applying the multi-patch counterpart of the classical SR approach as illustrated in Fig. 2.4. Recurrence of patches across different image scales, as illustrated in Fig. 2.5, implicitly provides examples of low-resolution/high-resolution pairs of patches, thus, giving rise to learning-based super-resolution from a single image. The great benefits of the combined framework are the capability to obtain super-resolved observations from as little as a single low resolution image, the eliminations of the external *a priori*-created database and the off-line training phase.

2.2.3 Super-resolution in spatial domain

Spatial domain approaches offer powerful methods to deal with the ill-posed problem of image restoration. The most recent and complete taxonomy of SR methods has been proposed in [NM14] and is presented in Fig. 2.6. This figure uses the execution context as the main classification criteria. The single- and multiple-images approaches are further sub-divided into families based on the used SR technique.



(a) Classical multiple image SR. (b) Single-image multi-patch SR.

Fig. 2.4 Contributions of patches (B_i) recurring in LR images (L_i) to the process of HR image (H) reconstruction [GBI09].

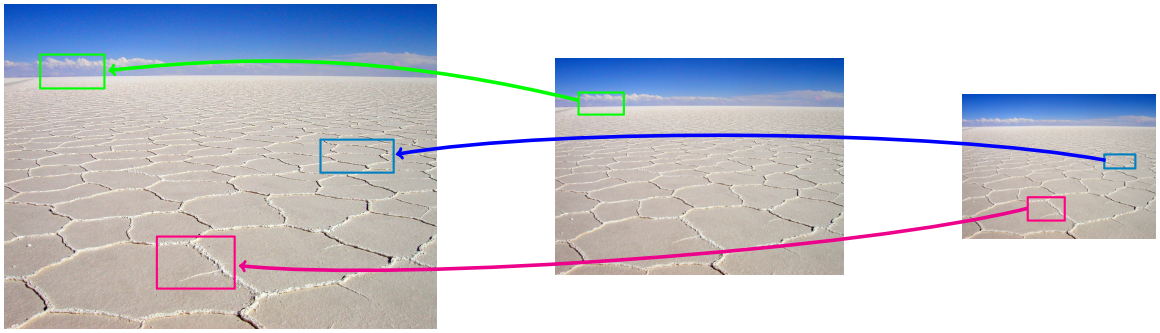


Fig. 2.5 Example of patch recurrence across scales resulting in implicit LR-HR pairs identification.

2.2.3.1 (A) Multiple-images super-resolution in spatial domain

The multiple-images-based approaches comprise five families of techniques:

(A.1) Direct. The theoretical basis of these techniques is the non-uniform sampling theory which allows for the reconstruction of functions from samples taken at non-uniformly distributed locations.These methods follow the pure classical SR approach presented in Fig. 1.1. Given a set of LR observations, one of the LR images is chosen as the target image and the others are registered against it. Following, the target image is scaled up by a specific scaling factor and the other LR images are projected (that is scaled and warped or shifted) into the reference grid using the data obtained from the registration stage. Then, the HR image is generated by fusing all the images together. The missing data are created by means of non-uniform interpolation. Finally, an optional deblurring/refinement kernel may be applied to the result. The fusing process is usually implemented

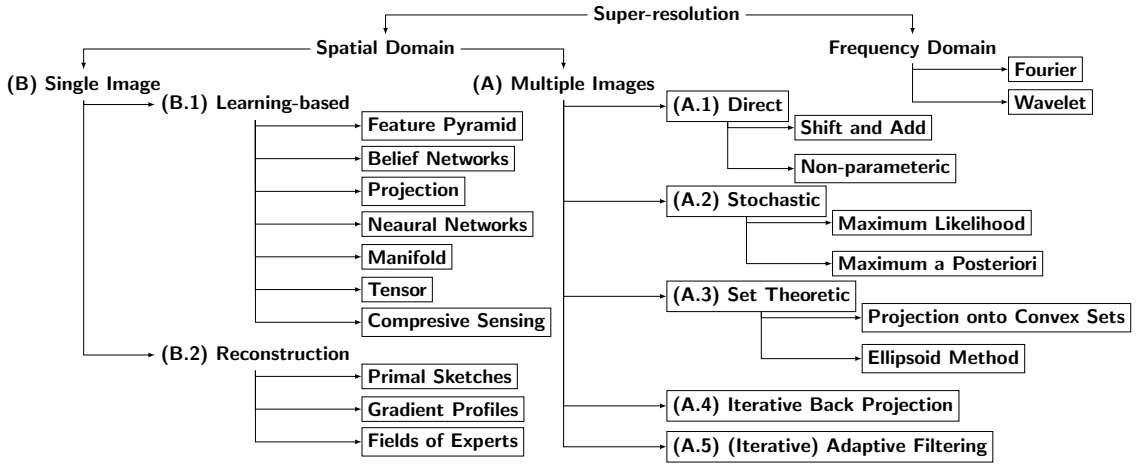


Fig. 2.6 Excerpt from the SR taxonomy proposed recently in [NM14].

as a weighted sum, thus, this approach is usually referenced to as *shift and add* [FREM03, FREM04b, FREM04a, FM06, AIAOM13].

More recent direct methods carry out the registration and fusion at patch-level. That is, the LR images are first divided into patches. Then every patch of the reference image is compared to a set of patches, including the corresponding patch and some patches in its vicinity, in the remaining LR images. Based on the similarity of these patches and their distances from the current patch, a weight is associated to every patch indicating the contribution of this patch in the process of producing the output HR patch. This approach helps to handle occlusion, local and complex motions (e.g. facial expressions change). The family of methods that deploy this type of processing is referenced to as *non-parametric* direct SR [PE09, PETM09, TMPE09, BBV10, CCL10, HS12, ZLRG12].

(A.2) Stochastic. Methods from this family consider HR image and motions among low resolution observations as stochastic variables. These variables are used to construct a Bayesian framework which is used to estimate the unknown values by minimization of a cost function. The statistical approaches vary in the ways: they treat the degradation process, the LR/HR observations *priors* (defined as our beliefs about the sensitivity and specificity of a process/variable) they use and the inference method used by the framework. The two most widely used statistical approaches to SR are the *maximal likelihood* (*ML*) [CKK$^+$96, SS94, SS96, LSZ97, EHO01, HYCC07] and the *maximum a posteriori* (*MAP*) [GH97, HBA97, BS99, RC01, TM10, CNYA12, KI12, YZS12, ZZLH12]. The ML ap-

proach only considers the relationship among the low resolution observations and the original high resolution. In contrast, the MAP approach incorporates also the prior image model to reflect the expectation of the unknown high resolution image. In most cases, the use of MAP-based approaches is encouraged as the preferred one.

In mathematics, a process of introducing additional information in order to solve an ill-posed problem is referred to as regularization. Thus, many of the stochastic approaches are also referred to as *regularization-based* SR. More on stochastic approaches to SR can be found in [Mil10].

(A.3) Set theoretic reconstruction. Set theoretic reconstruction assumes the SR process to be a feasible linear optimization problem with rational data that can be solved by generating a sequence of sets of possible solutions whose cardinality uniformly decreases. The subsequent sets are subsets of the previous set containing the desired images, that is the images that fulfill (or intersect with) a subsequent constraint convex set. Defining constraints in terms of convex sets is flexible and allows incorporating even non-linear and non-parametric constraints or priors.

In the context of SR, the most important set theoretic reconstruction algorithms have been the *ellipsoid method* and the *projections onto convex sets* (*POCS*) [SO89, EF97, HKK97, PA98, BS00, CTH03]. In the POCS method, the subsequent sets are determined by finding the intersection points of the constraint and the current solution set. In the ellipsoid method the subsets are ellipsoid bound by the constraints set. More on set-theoretic approaches to SR can be found in [Mil10].

(A.4) Iterative back projection (IBP). IBP is similar in its approach to the back-projection used in computer tomography image reconstruction. This method creates an initial guess of the high resolution image which is refined by iteratively projecting the difference (the residual) between the observed low resolution images and the simulated (back-projected) low resolution images [IP92, CD00, ZP00, ZRAP01, JWB03, BEZN04, BEZN05, YB06, DHWG07, GDAG09].

(A.5) (Iterative) adaptive filtering. This SR approach is most appropriate when the motion model and the point spread function model commute. The filters usually deployed for SR are the Kalman and Wiener adaptive filters. These methods are in effect linear minimum mean square error estimators and do not allow inclusion of non-linear *a priori* constraints. Although limited in terms of performance

understood as the output quality, these approaches offer very fast execution and have been mainly used in SR of video sequences [EF99b, EF99a, EF99c, CB07, CB06, CB08, KKC$^+$10].

Each of the above-presented approaches has its advantages and drawbacks. The direct approach is basic and intuitive method of super-resolution with its implementations having relatively low computational complexity. On the other hand, the direct approach assumes that the blur and noise characteristics are identical across all low resolution images. These methods rely on very accurate registration between images and thus are very sensitive to any errors produced in the image registration stage which can be easily propagated to the SR kernel. Moreover, the step-by-step forward approach with separation of the motion estimation and image fusion steps does not guarantee optimality of the restoration. The IBP is also understood intuitively and easily. However, this approach preserves the ill-posed nature of the inverse problem. Thus, there are multiple possible solutions and the algorithm either converges to one or it may oscillate between them. Although, the latter can be dealt with by incorporating *a priori* knowledge about the solution, the inclusion of the *a priori* constraints is not easily achieved in the IBP method and usually requires additional regularization terms. The possibility to conveniently include the *a priori* information, along with their theoretical simplicity, is the biggest advantage of the set theoretic methods. Nevertheless, these methods have high computational cost, slow convergence and lead to a solution that is not guaranteed to be optimal. Moreover, the use of the POCS method has the disadvantage of producing non-unique solution that depends on the initial guess. Practical implementations of POCS also have problems in dealing with the reconstruction of discontinuities of edges and details. Compared with other approaches, stochastic SR offer better robustness, flexibility in modeling noise characteristics and easiness of including the *a priori* knowledge of the solution in the probability priors. In certain cases (e.g. the noise process is white Gaussian, and MAP estimation with convex energy functions), this method ensures the uniqueness of the solution. The uniqueness of the solution allows the use of efficient gradient descent methods and is necessary for optimal reconstruction. Recently, approaches looking to combine the properties of the stochastic approaches with other families have been undertaken. The ML/MAP-POCS joint SR estimates the HR image by minimizing the ML/MAP cost functional while enforcing the containment of the solution within the intersection of the convex constraint. The advantage of the combined approach is that it ensures a single optimal solution, which is not the typical case when POCS approach is used on its own, while allowing any kinds of constraints and priors, even the ones that may present as impossible for purely stochastic approaches.

2.2.3.2 (B) Single-image super-resolution in spatial domain

Single-image SR is based on interpolation and hallucination of the high-fidelity details (usually high frequency part of the image) of the HR image. Single-image SR approaches comprise two families of techniques:

(B.1) Learning based. Learning-based approaches [BK00a, CZ01, BK00b, PRZ03, WT03, JG05, ZPJ08, KCR09, ZH12, ZHD12] use pairs of LR and HR training inputs to create a dictionary that relates the low-resolution data (or its features) with their corresponding high-resolution representation. During run-time, the given LR test input is used to traverse the dictionary in order to find the closest match (or a set of closest matches) and retrieve the associated HR-data to be used in the hallucination. The learning-based approaches are further divided based on the way the learning is carried out, the structure of the dictionary, the way the search is carried out and how the matches contribute to the hallucination process. The most recent work [NM14] defines seven sub-families of example-based SR approaches, namely: features pyramids, belief networks, projection, neural networks, manifold, tensors, and compressive sensing. Details on each of these approaches can be found in [NM14].

(B.2) Reconstruction based. The *a priori* knowledge is generalized in order to encapsulate techniques for reconstruction of primitives (and/or shapes), or statistics that can be used to carry out the reconstruction. The SR is carried out by applying the adequate reconstruction technique to each of the identified primitives (like edges, ridges, corners, etc.) or using the provided statistics to form a gradient based constraint to be applied to the reconstruction process [SZTS03, RB05, Fat07, SLJT08, SSXS08, XSW09, HFL10, XSW10, SSXS11].

Some of the single-image methods allow their execution in a multi-images context. These methods jointly exploit the information learned from a given high resolution training data set, as well as that provided by multiple low resolution observations. Also, the statistical approach can be incorporated in the example-based approach where the found patches are used as the prior image model and then merged into a MAP framework cost function in order to arrive at the closed form solution of the desired high-resolution image.

2.2.4 Super-resolution of video sequences

The SR processing of video sequences is usually required to be dynamic, that is to produce an output sequence that (at least) maintains the number of frames and the frame rate of the input sequence. The scenario of *dynamic SR* imposes high constraints on the execution time. In practice, the choice of algorithms for application of SR on video sequences is limited to approaches that offer a favorable trade-off between performance, execution time and resources requirements.

The existing approaches to SR of video sequences can be classified into the following four categories: (i) the sliding-window-based approach [SKR07, NHBS07, NSLZ07, PJ07], (ii) the sequential approach [EF99c, EF99b, FEM06, CB07], (iii) the simultaneous approach [BS99, ZM07, AMK03], and (iv) the learning-based approach [BBM03, DKA04, KHX$^+$06]. The first three approaches require multi-images/frames context, while the learning-based SR is capable of executing in a single-image/frame context. The learning-based approach to SR of video sequences is straightforward: subsequently apply learning-based SR of still images on each of the frames of the sequence. In the case of execution in multiple-images context, this straightforward approach is not the preferred one. Its use is not efficient as it leads to reloading of data and does not (even try to) take advantage of the relationships between the subsequent frames of the sequence. As for now, software implementations offer performance that is not sufficient for real-time execution, even though the single-image context shows inherent parallelism that allows simultaneous multi-frame SR. The available implementations of learning-based SR of video sequences have not been implemented in hardware. Considering that in the case of video sequences availability of multiple LR images/frames is assured, the multi-image SR techniques constitute the preferred approach, especially in the context of hardware implementation.

2.2.4.1 Multi-frame approaches to SR of video sequences

As aforementioned, the multi-image approaches require a number of frames to execute. This set of frames, called the *frame window* (or the *working window*), forms the execution *context*. The frame window comprises two types of images: the *target image* that is being super-resolved and a number of *reference images* (or *reference frames* (*RF* or RF_{nr})). Frame window usually comprises the immediate neighbors of the to-be-processed frame as illustrated in Fig. 2.7.

A consequence of using a multi-image context of execution is the requirement of registering the frames belonging to the current working window. In the context of SR of video

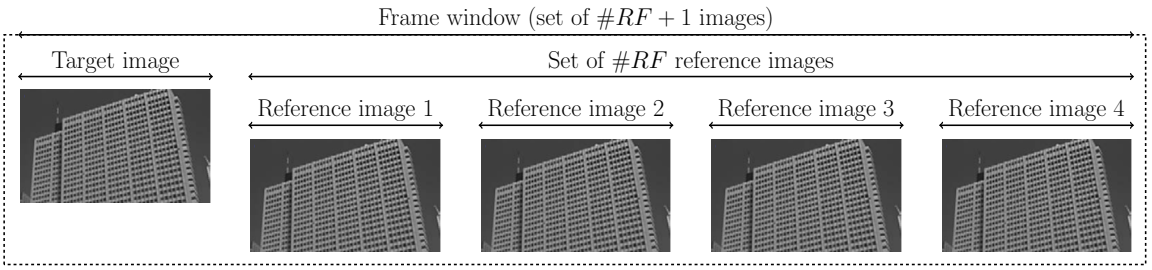


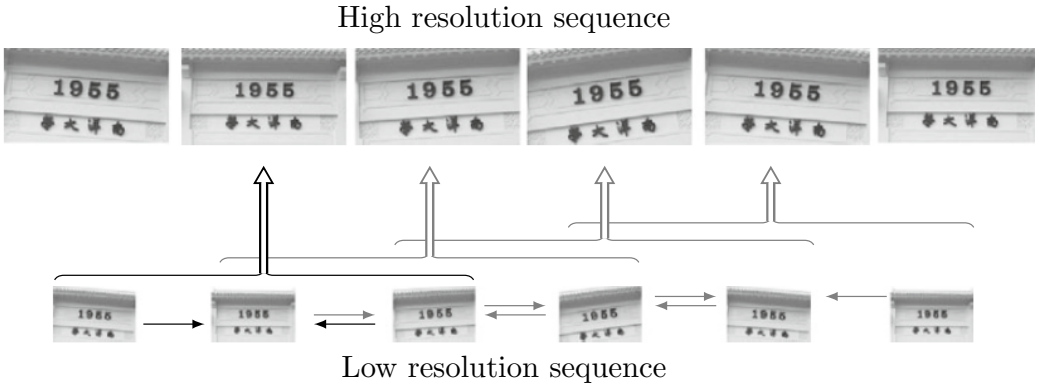
Fig. 2.7 Example of frame window comprising 5 images (4 reference images or reference frames (RF); $RF = 4$) and one target image).

sequences there are two main approaches to image registration and (effectively) context updating being deployed, namely: the *anchored* and the *progressive* approach. In the anchored approach, one of the frames of the context is chosen as the target frame and the other frames misalignments are registered in relation to this frame. In case of the progressive registration, the current frame is registered in relational to its immediate temporal neighborhood made up by the frames that precede it.

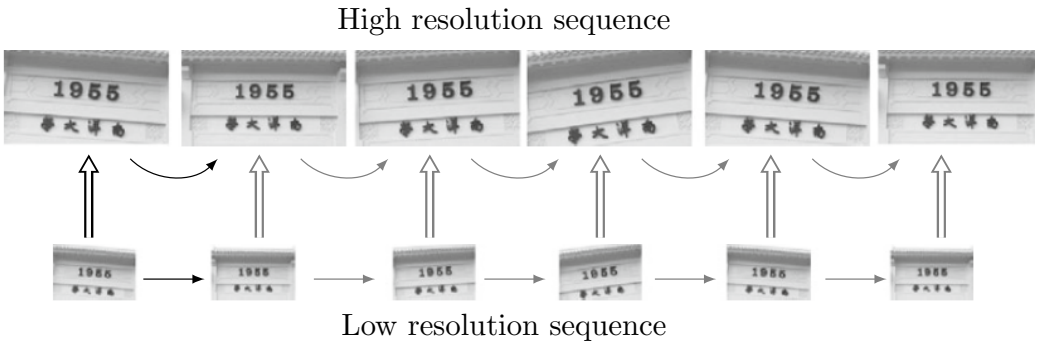
In practice, multi-image SR for video sequences is implemented using one of the three following approaches illustrated in Fig. 2.8:

Sliding-window-based SR approach. Deploys anchored registration over a set of consecutive low resolution frames which are later combined producing one high resolution image frame. The execution context is moved across the input frames to produce successive high resolution frames sequentially. The major drawback of this approach is that the temporal correlations among the consecutively reconstructed high resolution images are largely unexploited. Also, memory occupancy associated with this approach is high as all required data from the context need to be readily available from the memory.

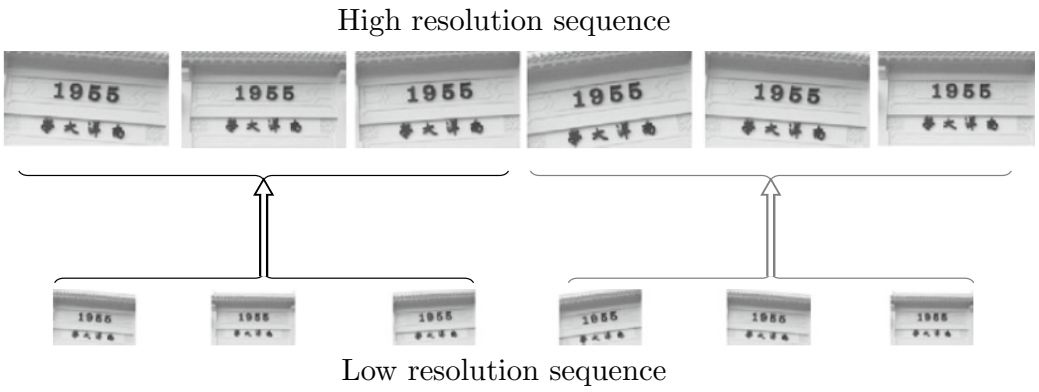
Sequential SR approach. The sequential SR approach tries to exploit the temporally correlated information provided by the established high resolution images. Correct identification and exploitation of temporal correlation of HR images using their LR observations is a challenging problem. In an effort to do so, progressive registration and previous HR frame can be used in tandem during the process of estimating the current HR frame. This allows to reduce the cardinality of the execution context while still including (by means of propagation) information from multiple frames. Reduction of the execution context usually leads to significant reduction of memory occupancy and computational complexity. On the other hand, the propagation may lead to er-



(a) Sliding frame window approach.



(b) Sequential approach.



(c) Simultaneous approach.

Fig. 2.8 Approaches to super-resolution of video sequences executing in multiframe context. Based on [TM11].

rors escalation and drift if robust fusion and outliers elimination mechanisms are not deployed.

Simultaneous SR video approach. This approach tackles the problem of simultaneous reconstruction of multiple high resolution images. The typical approach is to impose temporal smoothness constraint on the prior image model and produce multiple images from the context at once. Alternatively, multiple contexts could be processed in parallel i.e. multiple sliding windows could be selected and used to reconstruct multiple high resolution frames simultaneously. In both cases, simultaneous SR poses very high memory occupancy requirements, since high resolution and low resolution images from multiple contexts are required to be readily available from memory at the same time throughout the reconstruction process.

It should be noticed that on-line video SR is significantly more demanding in terms of performance, execution speed and memory occupancy than SR of still images [Goh13]. The highest requirements are presented by the simultaneous SR where frames from multiple contexts need to be stored and/or hardware resources have to be duplicated (or pipelined) if the computations are to be parallelized. On the other end of the requirements spectrum is the sequential approach, whose requirements in terms of resources and computational power seem to be the lowest. Nevertheless, in order to be successful, this approach requires algorithms that allow inclusion of *a priori* knowledge to regularize the SR process. Thus, stochastic/regularization-based approaches are preferred over direct/IBP-based ones. The former are known to be mathematically involved and suffer from high execution times. In the context of real-time on-line dynamic SR of video sequences less involved (and thus faster) algorithms are preferred. The interpolation-based, namely, the direct and IBP methods have found most success in meeting real-time constraints.

2.2.4.2 Dynamic super-resolution using sliding frame window

Dynamic SR using the sliding-frame-window approach is provided by moving the frame window across the video sequence and performing the SR process once for each frame of the sequence. After the processing of one frame is terminated, the least recently loaded frame is discarded, the remaining frames are shifted, and a temporarily subsequent (next) frame is loaded. For a sequence composed of n frames, processing of the whole sequence is defined as carrying out the super-resolution process for each frame $t : t \leq n$ with the set κ of frames contributing in the fusion process being limited to a size $\kappa \leq n$ (in practice $3 \leq \kappa \leq 17$) and updated for each of the frames being processed. The set κ represents the

sliding frame window with each of the frames in κ whose index is $\neq t$ acting as a reference frame. Using this set-up, the process of sliding window updates, for an example where $n = 5$, $\kappa = 3$ (containing up to one preceding and one succeeding frame) is illustrated in Figure 2.9.

2.3 Image registration

In this work we consider images as a somehow limited 2D projection onto the image/sensor plane of the real 3D representation of environment (natural/synthetic scene). Having a series of captures of the same scene we consider them a spatio-temporal representation of the variations of that scene. The differences present between images are introduced due to different imaging conditions. Some of these variations may manifest themselves as aliasing or exclusive data which can be utilized in the process of image enhancement by the super-resolution process. In order to use these data they need to be localized first. Thus, it is of most interest to find the correspondence between the images in order to be able to fuse the details from different images that represent the same piece of the scene. This correspondence describes the spatio-temporal evolution of the images and due to its nature can be characterized in terms of the motion of the contents between the captures caused by the contents movement and/or the relative motion between an observer (an eye or a sensor) and the scene. Thus, it is of crucial importance for the super-resolution process to be able to accurately quantify the evolution of apparent movement between two (or more) captures.

In the context of computer vision, the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene is called the *optical flow* or *optic flow* [Gib50]. Optical flow is used to obtain the unknown vector field of the apparent motion encountered in the 2D projection, called *motion field*. Motion field relates the points in one image to their corresponding points in the other. In order to detect and model the variations between two images, the images have to be geometrically aligned with each other. *Image registration* is the process of overlaying two or more images of the same scene taken at different times from different viewpoints and/or by different sensors [ESHEK12]. Due to the optical flow mechanics, image registration is an inverse problem in imaging that looks for a transformation that relates the points in one image to their corresponding points in the other image. The problem of image registration is an example of the more general problem of estimating motion in an image sequence. In Fig. 2.10 we can see an example of a motion field produced by image registration of two frames of the *mobile* sequence.

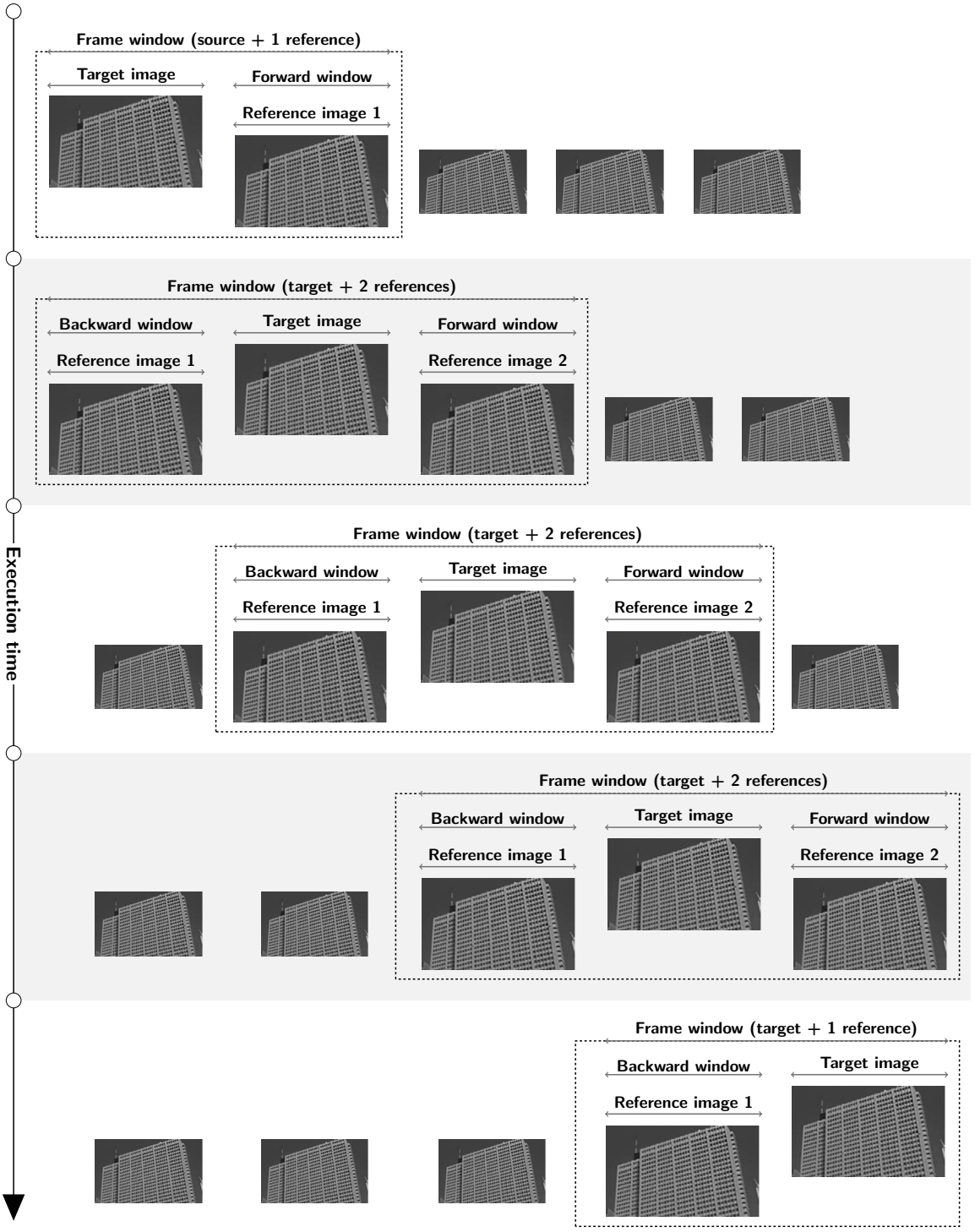


Fig. 2.9 Replacement scheme for a sliding frame window comprising one succeeding and one preceding frame.

(a) Input image a. *(b) Input image b.* *(c) Computed motion field.*

Fig. 2.10 Results of image registration: (a) and (b) input images, (c) computed motion field correlating images (a) and (b). Source [Boc09].

Image registration is of practical importance in many fields, not only in computer vision [Bro92, ZF03]. Variants of image registration technique developed by Lucas and Kanade are used in almost all motion-compensated video compression schemes such as MPEG and H.263/4 [Ric04, Boc09]. More sophisticated image registration algorithms have also been developed for medical imaging and remote sensing. Common applications include, among others, objects tracking, correction of camera jitter (stabilization), image stitching (mosaic) and 3D shape reconstruction from motion.

Before going into details, some naming conversions must be defined. In this work, we will denote the images that are used in the registration process as, either, the *target* or the *sensed* (also *source* or *subject*) [Bro92, ZF03]. The former image is kept unchanged and is used as a basis for the transformation. The latter refers to images that are geometrically transformed to be aligned with the target image. Finally, a *transformation* (or warping) is the function used to modify the sensed image with the objective of approximating the target image.

2.3.1 Classification of image registration techniques

As aforementioned, image registration carries out alignment of images of the same scene taken at different times, from different viewpoints, and/or by different sensors. Hence, the principal way to classify image registration methods is carried out based on the set-up used for image acquisition. Based on this criterion we can divide image registration methods into the following four groups [ESHEK12].

Multi-views analysis. These methods use images acquired from different viewpoints in

order to gain a larger two dimensional view or a three dimensional representation of the captured scene. Examples of such applications are image stitching (mosaicking or panoramas [Sze10]) or SR reconstruction of remotely sensed images and shape recovery in computer vision.

Multi-temporal analysis. These methods analyze images of the same scene taken at different times, and possibly under different conditions, in order to evaluate variations appearing between the subsequent captures. Examples of such applications are automatic change detection, monitoring tissues changes in medical images and SR reconstruction of remotely sensed images.

Multi-modal analysis. These methods use images of the same scene obtained using different sensors with the aim of integrating them into a more complex and detailed scene representation. Examples of such applications are processing of magnetic resonance images (MRI), enhancement of spectral resolution of images (hyperspectral imaging) and fusion of remotely sensed images.

Scene-to-model registration. Acquired image is registered with a template image (natural or synthetic). Examples of such applications are comparison of patient's image with a digital anatomical atlas (so-called *atlas* mapping), specimen classification, real-time template matching, automatic quality inspection and registration of satellite images into maps.

2.3.2 Fundamental components and steps of image registration

The presented definition of image registration as a process of image alignment is very general and allows several acquisition scenarios. Thus, even though, the basic idea of carrying out image registration can be summarized as composed of two stages [Bro92]: (i) determination of the points that correspond between the images and the nature of the correspondence, (ii) determination of the transformation (and fine-tuning of its parameters) that 'best' models the establish relation, the ways in which it is implemented vary significantly. Moreover, regardless of its implementation methodology, the image registration processes is in general computationally demanding. To tackle the problem of computational intensity, most of image registration algorithms are specifically designed for a specific application and are not suitable for all types of problems or data. The observed broad spectrum of available methodologies for image registration implementation lead to several possible classifications. However, all registration techniques involve searching over the space of transformations of

certain type (e.g. affine, polynomial or elastic) to find the optimal transformation for a particular problem. These two processes can be viewed as combinations of choices for the following four components [Bro92].

Feature space. Feature space extracts the information in the images that will be used for matching, According to their nature, feature spaces can be classified as feature-based and area-based. Feature-based methods establish a correspondence between a number of especially distinct points in images (i.e. edges or corners). In contrary, area-based methods consider predefined size areas of the image as features and look to register their correspondence either in spatial (*direct* methods) or frequency domain.

Similarity metric. Also called proximity or cost function. It is a measure that interconnects the transformation and data to be transformed. It defines the 'test' carried out in order to determine the relative accuracy of each transformation. The criteria used by the similarity measure determines what types of matches are optimal and is usually a trade-off on the required accuracy, acceptable speed, and complexity of the data.

Search space. It defines the class of transformations (and its range of values) that the system is capable of performing in order to align the images. Search space define the cardinality and the structure of the set of possible tests (so-called candidates) for which similarity measures will be computed in search for the 'best' match.

Search strategy. It is a pattern that decides how to choose the next transformation from the search space, to be tested in the search for the optimal transformation. Search continues according to the search strategy until a transformation is found whose similarity measure is satisfactory. The choice of the feature space requires a suitable error metric as well as a suitable search technique to be chosen.

Combinations of the above described four components determine the flow of the image registration process. For most image registration methods this leads to an execution flow that can be generalized as consisting of the following four steps [ZF03, ESHEK12].

1. *Feature detection*. For feature-based algorithms salient and distinctive objects are detected and represented. In area-based methods a predefined subsets of the image are used instead of 'features'. Tessellation, which could be seen as the counterpart of the future detection, forms a part of the feature matching process and does not form a separate step.

2. *Feature matching*. The correspondence between the features detected in the sensed image and those detected in the target is established.

3. *Transform model estimation*. The type and parameters of the so-called mapping functions, aligning the images, are estimated based on the established feature correspondence. In order to do so, the transformations from the search space are tested using search strategy.

4. Image *re-sampling and transformation*. The sensed image is transformed by means of the selected mapping functions. In general, the result of the coordinates transform can be fractional. Image values in the non-integer coordinates are computed by appropriate interpolation technique.

The choice of the appropriate image registration technique cannot be understated and should be adjusted to suit the specific problem. The selection ought to consider the acquisition manner, the relationship between the variations among the images and the choices for the four components of image registration, all of which are established and presented in [Bro92].

2.3.3 Feature domain

The first step in registering two images is to decide on the feature space to use for matching. The feature space is the representation of the data that will be used for registration. This may be the raw pixel values, i.e., the intensities, but other common feature spaces include: edges, contours, surfaces, etc. The choice of feature space determines what is matched. The type of features determines possible similarity metrics that can be used. The similarity metric determines how matches are rated. Together the feature space and similarity metric can ignore many types of variations which are not relevant to the proper registration and optimize matching for features which are important.

Based on the feature space selection we distinguish two classes of methods: the so-called *area-based* and *feature-based* image registration. Each of these classes can be further divided based on the similarity metrics used to carry out the matching and/or transformation step. A detailed description of these sub-classes is considered out of the scope of this work and can be found in [Bro92, ZF03, ESHEK12]. A classification of the image registration methods based on these criteria is presented in Fig. 2.11.

The choice of the feature space should consider the application and acquisition conditions in order to reduce some of the unwanted (volumetric) variations while, where needed,

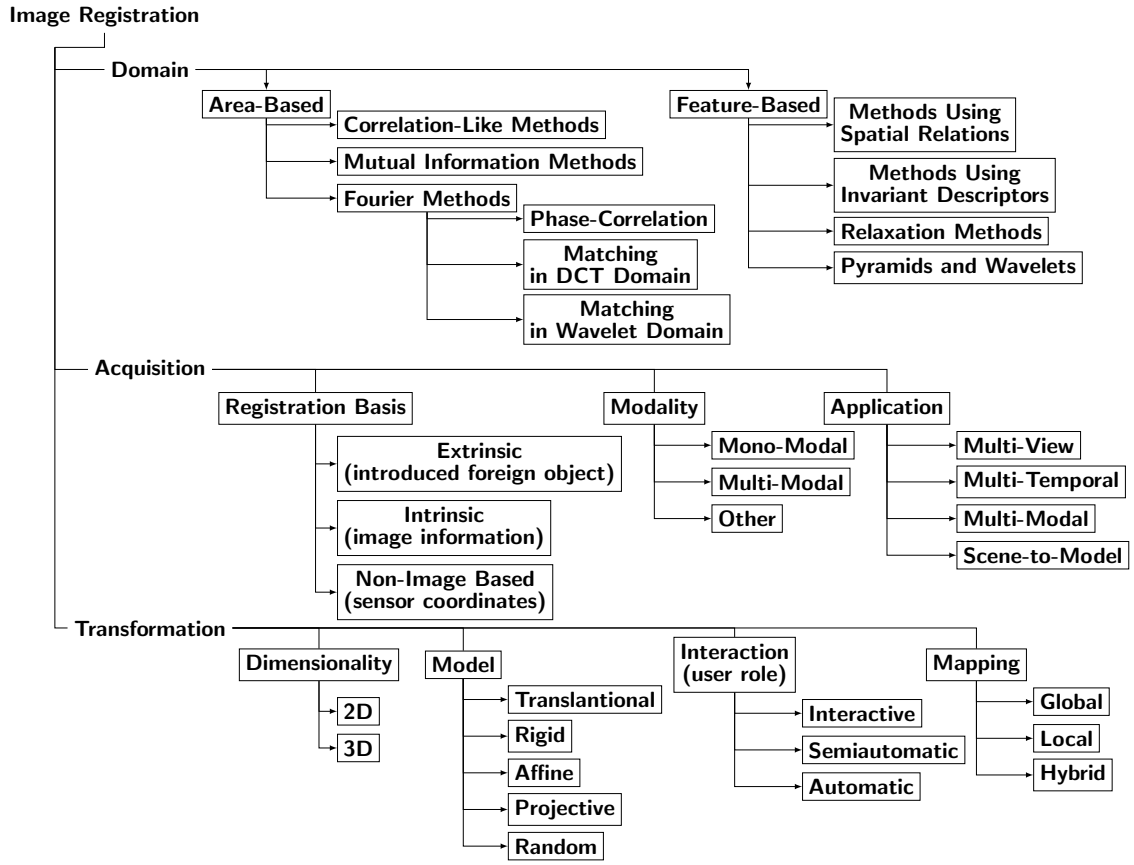


Fig. 2.11 Classification of the most popular approaches to image registration.

allowing efficient implementation. If uncorrected variations have not been eliminated by the feature space and similarity metric, then the search for the optimal match is also made more difficult, since there are more likely to be several local optima and a less monotonic space [Bro92]. This problem is solved to some extent by feature detection step and similarity metric.

Feature-based class encapsulates the algorithms that work by extracting a sparse set of features in the images which are then matched against each other. To provide efficient registration these methods rely on the existence of salient points in the input. The problem with feature-based matching is that, typically, good, matchable features such as corner points are sparse while poor easily mismatched features such as edges, are denser. Even when reasonably unique features are available, establishing the correct correspondences can be problematic, especially for cases when occlusion occurs. Image registration of images with features occlusion is known to be the most challenging scenario for feature-based methods, and is likely to lead to matching errors [BB95].

Area-based approaches, sometimes referenced as correlation-like or template matching, are less sensitive to these problems. These techniques do not rely on the presence of salient points (called *control points*), rather, they consider areas of the image as features used in matching—shifting the emphasis from feature detection to the feature matching step. The matching step is carried out by directly minimizing point-to-point dissimilarities for predefined sized regions/patches. The variable window sizes can be used near occlusion boundaries to handle multiple motions. This matching and minimization in most cases are carried out in the spatial domain using directly the illumination values (so-called *direct methods*). However, it is possible to carry out the process using representations in the frequency domain which offers advantages in noise sensitivity and computational complexity.

The limitations of the area-based methods originate from their basic idea: the predefined size/shape window (rectangular windows are most often used) suits the registration of images which locally differ only by a translation. If images are deformed by more complex transformations, this type of window is not able to cover the same parts of the scene in the target and sensed images. Feature detection is not implemented by the area-based methods rending them more sensitive to acquisition conditions (and noise). Hence, a more robust and noise tolerant similarity metric are required to be used in these algorithms.

Some of the aforementioned limitations are alleviated by carrying out the processing in the frequency domain. Frequency domain is less susceptible to differing conditions of illumination since illumination changes are usually slow varying and therefore concentrated at low-spatial frequencies. Similarly, the techniques using frequency domain are relatively scene independent and useful for images acquired from different sensors since it is insensitive to changes in spectral energy. The scene independence is further strengthened in the case of phase-correlation methods that use only the phase information making the correlation measure invariant to linear changes in brightness. By using the frequency domain, the *Fourier methods* achieve excellent robustness against correlated and frequency-dependent noise. On the other hand, if the images have significant white noise, noise which is spread across all frequencies, then the location of the peak will be inaccurate since the phase difference at each frequency is corrupted. In this case, use of the spatial cross-correlation is better. Also the Fourier methods are applicable only for images which have been at most rigidly misaligned.

To summarize, the use of feature-based methods is recommended if the images contain enough distinctive and easily detectable objects. This is usually the case of applications in remote sensing and computer vision. Moreover, feature-based approaches have also the advantage of being more robust against scene movement, and are potentially faster, if imple-

mented in the right way. It should be noted that registration methods using simultaneously both area-based and feature-based approaches have recently started to appear [HZ07].

2.3.4 Transformations

In the context of image registration a transformation can be defined as the mathematical model that maps pixel coordinates from one image to another. Most registration techniques involve searching over the space of transformations of a certain type to find the optimal transformation for a particular problem. Thus, before the registration and alignment of images happens, it is necessary to establish the basis of the possible search spaces and applicability of these transformations. These transformations can be expressed by a variety of models ranging from simple 2D translations to complex 3D movements of elastic body models. This section will focus only on the models and mappings which are of relevance for the task of 2D image registration.

2.3.4.1 Transformations primitives describing movement in 2 dimensions

Let x_{2d} denote a geometric primitive of a point in 2-D space x_{2d} defined using a pair of values (x,y) as $x_{2d} = (x,y) \in \mathbb{R}^2$. In computer graphics a point in 2-D space is usually represented in *homogeneous coordinates* as $\tilde{x}_{2d} = (\tilde{x},\tilde{y},\tilde{w}) \in \mathbb{P}^2$, where $\mathbb{P}^2 = \mathbb{R}^3 - (0,0,0)$ is called the 2D projective space. The conversion between the Cartesian and homogeneous coordinates is described as

$$\tilde{x}_{2d} = (\tilde{x},\tilde{y},\tilde{w}) = \tilde{w} \cdot (x,y,1) = \tilde{w} \cdot \bar{x}, \tag{2.1}$$

where $\bar{x} = (x,y,1)$, created by extension of the coordinates number with third coordinate equal to '1', is called the *augmented vector*. Homogeneous coordinates have the advantage of allowing all of the transformations to be expressed using multi-dimensional matrices with mapping parameters.

Having established the representations of the 2D point, it is necessary to define and describe the basic set of primitives that carry out the transformations listed in Fig. 2.12. This description is based on the contents from [Sze10] covering only the transformations that are used most extensively. A graphical illustration of these primitives is presented in Fig. 2.12. Basic properties of the introduced below primitives are presented in Table 2.2.

Translation. Translation is a pure shift in 2D space which preserves the orientation (size,

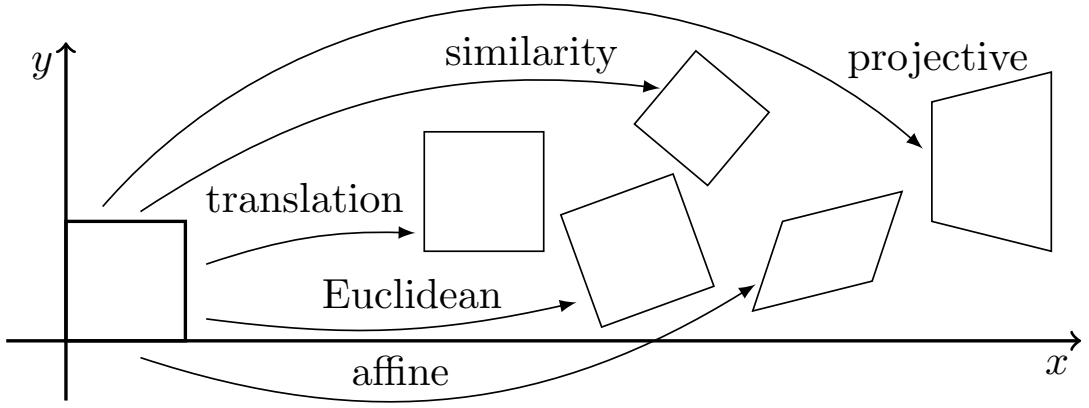


Fig. 2.12 Basic set of 2-D planar transformation [Sze10].

scale and shape) of the object. Having δ_{2d} represent the translation distance, translation can be written as $x'_{2d} = x_{2d} + \delta_{2d}$.

Euclidean. Also known as 2D *rigid body motion*. It can be seen as a sequence of two transformations: rotation and translation. It preserves the inter-pel distances, shape and scales but allows changes of its orientation. Denoting rotation as R, rigid body motion is modeled as $x'_{2d} = R \cdot x_{2d} + \delta_{2d}$.

Similarity. Extends the Euclidean model by allowing the scale of the object to be changed. This process is modeled as $x'_{2d} = s \cdot R \cdot x_{2d} + \delta_{2d}$, where s is an arbitrary scale factor. One thing to note is that the similarity transform still preserves angles between lines. This translation is sometimes referenced as *scaled rotation*.

Affine. Affine transformation is modeled as $x'_{2d} = A_{2\times 3} \cdot \bar{x}_{2d}$, where $\bar{x}_{2d}$ is such a representation of x_{2d} in the homogeneous coordinates space that $\bar{x}_{2d} = (x, y, 1)$ and $A_{2\times 3}$ is an arbitrary 2×3 matrix with mapping parameters. Affine transformation preserves the parallelism of lines.

Projective. Also known as a *perspective transform* or *homography*. Allows even higher *degree of freedom* (DoF) than the affine transformation. Denoting $\tilde{H}_{3\times 3}$ an arbitrary 3×3 matrix with mapping parameters, the projective transformation is modeled as $\tilde{x}'_{2d} = \tilde{H}_{3\times 3} \cdot \tilde{x}_{2d}$. Straight lines remain straight after the transformation.

TABLE 2.2 *Hierarchy and properties of 2D coordinate transformations. An extract from [Sze10].*

Transformation	Matrix dim	# DoF	Preserves
Translation	2×3	2	orientation
Euclidean(rigid)	2×3	3	lengths
Similarity	2×3	4	angles
Affine	2×3	6	parallelism
Projective	3×3	8	straight lines

2.3.4.2 Global and local transformations mapping

Each of the presented transformations representing motion models can be constructed or mapped by considering different extent of the available support (control points (CP; feature-based methods) or pels (area-based)). The extent of support considered in the process of mapping parameters estimate determines whether each individual technique is classified as *local* or *global*. Global models use all available support for estimating one set of the mapping function parameters which accounts for all the variations between the images. In other words, a single equation is considered valid for the entire image, and it is used to describe the motion over the entire visual field. As these variances become more local, it will become progressively more difficult for a global point-mapping method to model all of the changes using one global transformation. In presence of significant and multiple local geometric variances or local 3D features observed from different viewpoints (resulting in different 3D-to-2D projections) global methods may fail to account for the misalignment between the images in a satisfactory way. In these cases, more than one transformation, which limit their support by using only a local neighbourhood, would be preferable. To accomplish that, local methods use not one, but a set of mapping parameters that vary across the different pieces of the support in order to account for different models of (local) variations. In other words, the mapping transformation is no longer a single mapping with one set of parameters independent of position. This allows local methods to be more powerful and handle many (combinations of) distortions that global methods cannot. Several authors have shown the superiority of the local or at least locally sensitive registration methods above the global ones in situations where local variations are present [YMB11, BAA05, CLT+08].

In all cases, there is a trade-off between the power of these methods and their corresponding computational and implementation cost. As illustrated in Fig. 2.13 and Fig. 2.14 the choice of mapping of the transformations directly influences the size and complexity of the search space. In practice, this manifests itself as execution time and memory requirements. Local methods are well-known to have the largest and most complex search

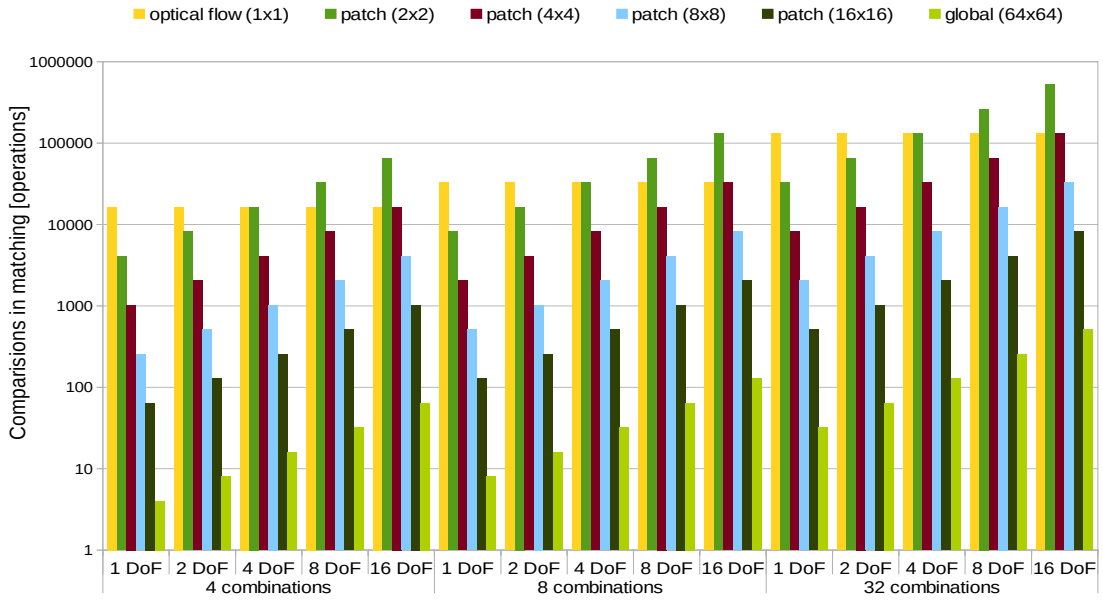


Fig. 2.13 Number of matchings as a function of template size, degrees of freedom and candidate combinations. Optical flow uses only 1 DoF.

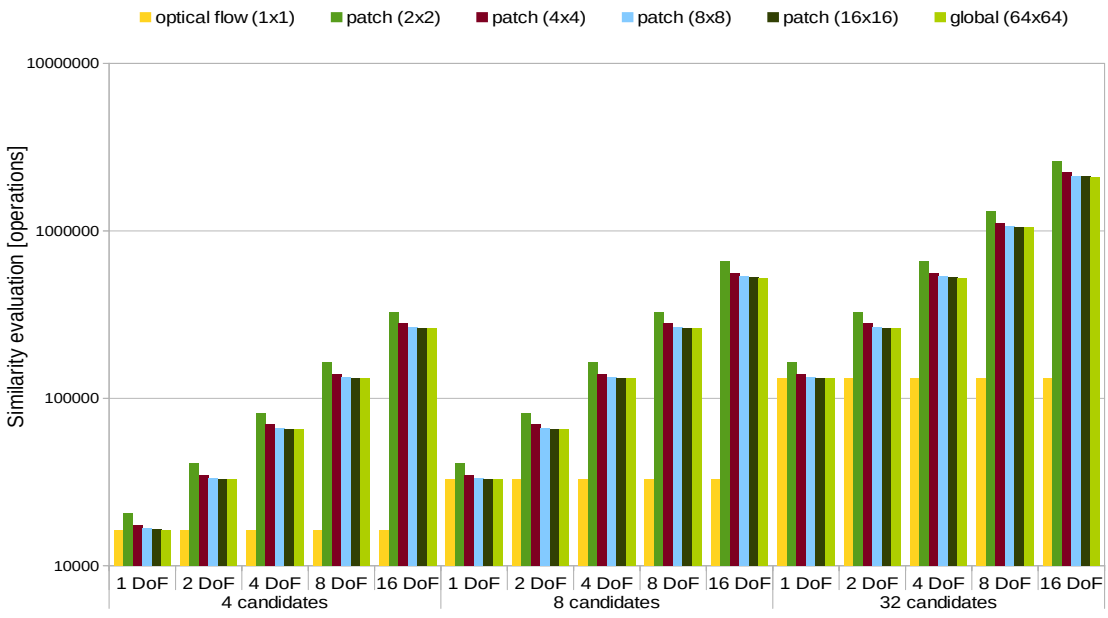


Fig. 2.14 Number of similarity index evaluations as a function of template size, degrees of freedom and candidates number. Optical flow uses only 1 DoF.

spaces. On the other hand, is many cases local registration allow pieces of the input to be registered without influencing other portions which have already been matched leaving more room for parallelization. Moreover, for many registration problems, both local and global distortions exist, and it is sometimes useful to take a hierarchical approach in finding the optimal transformation. In this case, the regions with local variations are registered using local methods with the rest of the support being described using one global transformation mapping. This may lead to significant reduction in computational complexity and implementation cost [BAA05, CLT+08].

2.3.5 Search space

Let *search space* denote the set of created by superposition of all allowable transformations with all their respective allowable parameters values in the defined feature space. Then, the aim of feature matching is to find the combination of transformation and its parameters from the search space which carries out the registration in an optimal way. To this end it is necessary to devise methods that (i) identify the transformation (and its parameters), and (ii) measure the quality of results on the preselected features in a quantitative way. The former and the latter methods are called the *search strategy* and the *similarity criterion*, respectively.

The matching operation in spatial domain that traverses the set of possible transformations computing the corresponding similarity test results for a set of allowed templates is known as 'template matching' [TK08]. A search iteration is carried out by placing the template over the reference at the test location and computing the similarity metric. The similarity metric is constructed in a way that it returns higher values for templates placed over a reference that is more similar (has smaller differences between the corresponding intensities). The optimal transformation is the one for which the similarity measure reaches its global optimum (maximum in case of similarities). Finding the global optimum of similarity index is a multi-dimensional optimization problem, where the number of dimensions corresponds to the degrees of freedom of the class of the expected geometrical transformation.

For example, in the case when the only allowable transformation is translation, then the search space is the set of all translations over the range of parameters (in this case displacement/horizontal and vertical shifts). Under the assumption of global mapping, the matching process would be carried out by iteratively translating the sensed image by a value from the defined range and quantifying how the template and the reference pels correlate.

The only way to assure that the displacement that results in the optimal transformation can be found, that is the global maximum of correlation, is for the matching to be carried out exhaustively over all possible displacements.

2.3.5.1 Computational complexity as a function of search space

The computational intensity associated with template matching is determined by the size of the search space, type of transformations mapping and the computational intensity of the similarity measure estimation. The size of the search space is determined by the number of degrees of freedom of the (expected) transformation model and the range and grain of the allowable values of the parameters. The simplest solution of trying out all allowed combinations from the search space results in the template being transformed (translated, rotated, scaled, etc.) for each possible transformation of interest. This affects the number of combinations that need to be tested, skyrocketing with the increase of the number of degrees of freedom and/or search range.

The type of mapping determines the number of sets of parameters that have to be computed to align the images being registered. This translates to the number of templates for which the matching process has to be carried out. In global mapping only one template is used. The number of templates used by local methods corresponds to the number of regions that the target image has been divided into. In case of area-based correlation-like image registration three types of templates can be defined based on the number of pels that they contain: a singular pel, a full frame and a block of pels.

The choice of granularity of the template has a significant impact not only on the dimensionality of the search space but also on the quality of the results. Using a coarser-grain template reduces the number of templates (towards the limit of one) at the cost of potential results quality degradation should local distortions be present. The use of a finer-grain template has the disadvantage of increasing the number of templates used in matching introducing additional operations in the similarity measure computation. Piece-wise (local) search of transformations may result in a *blocking effect*, characterized by the appearance of artifacts at patch boundaries. Effective treatment of these variations would requires additional processing [Ric04]. Consequently, methods using fine-grain templates tend to have the largest and most complex search spaces. An exception to that rule is the case of using only one pel. This finest-grain template is an interesting case as it allows to reduce the complexity of the motion model to purely translational. Significant reduction in computational complexity, relatively low memory requirements, and intrinsic parallelism have made optical flow one of the preferred ways for hardware implementations in low cost systems.

On the other hand, accurate estimation of the true transformations becomes more and more challenging for fine-grain templates. The change from coarse-to-fine template is carried out by limiting the extent of support used in matching. A well known trade-off of reduction in support is the loss of ability to accurately represent structural information. This makes fine-grain templates more susceptible to producing false motion estimation, especially when applied on homogeneous regions of the image [CLT+08]. To some extent this problem can be mitigated by using more robust (and more computationally intensive) similarity measures.

2.3.6 Search strategies

In most cases, trying out all of the combinations from search space with high cardinality is too time-consuming to be practical. Due to its high computational intensity, the exhaustive search is rarely implemented for registration of images using models with more than 2 degrees of freedom. In practice, exhaustive search is used as the baseline to evaluate the quality of results of other algorithms with reduced computational intensity. The approaches to reduce the computational complexity of template matching aim at lowering the number of tests or making the test less expensive. The latter is implemented by using a mathematically less involved similarity metric and/or allowing early termination at various stages of computation should a threshold value be reached. The former can be enforced by limiting the set of allowable transformations (arbitrarily or based on an *a priori* knowledge) and/or using *heuristic* methods for matching. Reduction of transformations model limits the allowable transformations to those with lower number of degrees of freedom. Heuristic algorithms encapsulate a set of methods that limit the number of carried out tests by constraining the search range of parameters for which the matching is carried out. In case of the area-based matching, the space described by the set of combinations of parameter values of the search range is referenced as the search area. The search area in spatial domain is described by its (relative) span and the size of the template. An example of such a search area used in piecewise area-based correlation-like registration (block-matching) is presented in Fig. 2.15. This particular case assumes translational model over a range of (search radius$_x$,search radius$_y$) displacements and a square template composed of template size$_x$ $\times$ template size$_y$ pels.

The pattern used to traverse the search range/space is known as the *search strategy*. Heuristic algorithms can be classified based on the deployed search strategy. It is difficult to present a complete classification of search strategies without sacrificing presentation clarity. Each search strategy has its advantages, disadvantages, sometimes limited domains

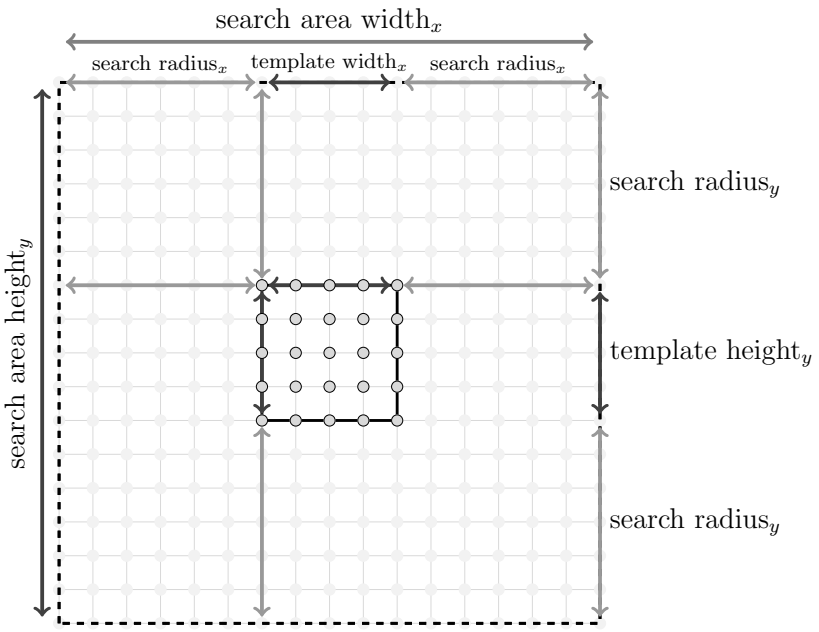


Fig. 2.15 Search area structure used in block matching of a template assuming translational model over a range of displacements.

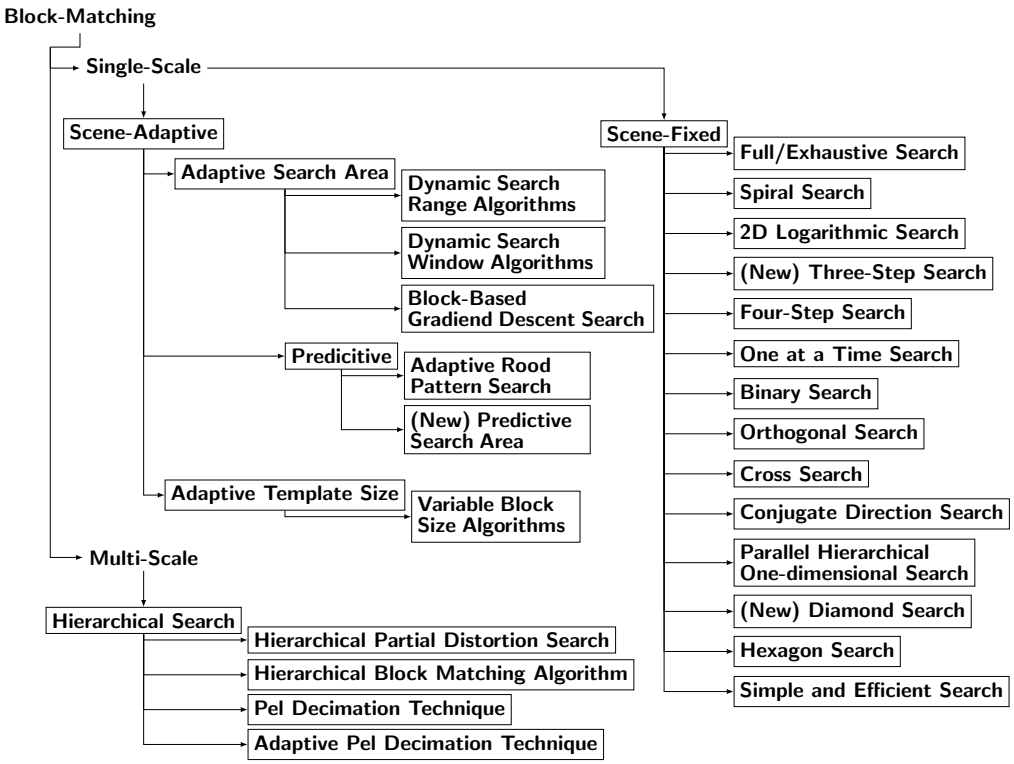


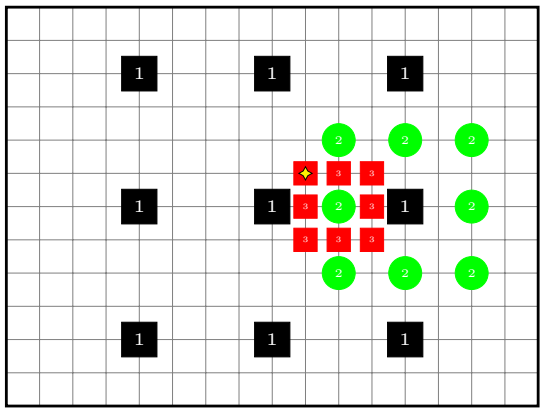
Fig. 2.16 Classification of block-matching implementations based on the used search strategy.

and usually offers a different grade of the accuracy/complexity trade-off. In general, the selection of the search strategy is done per particular application and is determined by the assumed constraints on the search space, required accuracy, available time/power budget and the difficulty of finding the optimum. A simplified classification of search strategies most widely deployed in implementations of block-matching (for spatial domain) is presented in Fig. 2.16. Rather than describing the presented implementations, it is more useful to provide a brief presentation of the main ideas on which most of heuristic block-matching algorithms are based on.

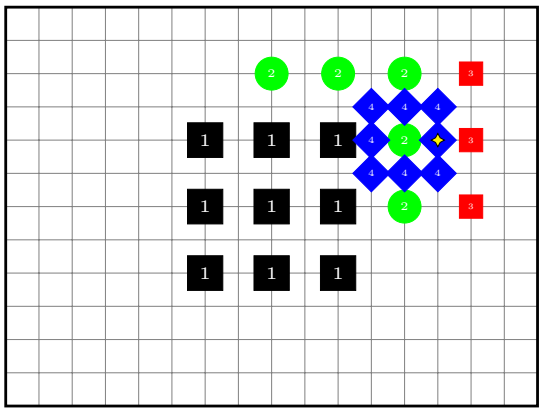
Sub-sampling. Sub-sampling is the simplest and most straight-forward way of reducing the computational intensity of correlation-based piece-wise template matching. Sub-sampling reduces the support used to calculate the similarity index. In case of area-based block-matching this is implemented by estimating the match criteria across a sub(sampled)-set of pels instead of doing so over all pels from a patch.

Multi-path/point search. Many block-matching techniques rely on the assumption that the correlation surface within the defined search range is relatively smooth. This leads to an expectation that once a 'good' match has been found, an even 'better' match is assumed to be nearby. This is generally correct in many sequences with the exception of sequences with periodic patterns such as the windows of the building shown in Fig. 2.18(e). Multi-point strategies exploit these notions by repeatedly carrying out the matching for a relatively small number of test points scattered across the search range in a regular pattern. After finding the iteration's 'best' match, the pattern is refined (usually points are scattered over a smaller area), and the next iteration of the search continues in the vicinity of the 'best' match over a much narrower range of displacements. The test carried out can vary (change) from stage to stage, usually becoming more strict for later stages. If the test varies from stage to stage, it usually becomes more accurate for later stages. This process can be repeated several times until the desired motion vector accuracy is achieved.

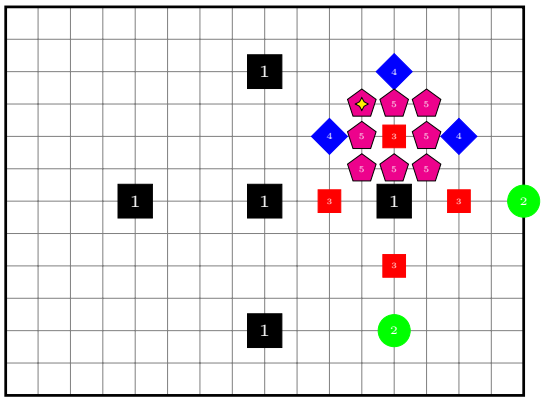
The literature provides many examples of such processing. The existing implementations differ in terms of the used update policy, the (spatial) pattern of the candidate subset and the conditions defining the transition between the steps. In the context of block matching, the most commonly used search strategies are the: (new) three-step-search [KIH+81, LZL94], four-step-search [PM96], diamond search [ZM00], two dimensional logarithmic search [JJ81], cross search [Gha90], and orthogonal search [PHS87]. Examples of particular paths for these algorithms are presented in Fig. 2.17.



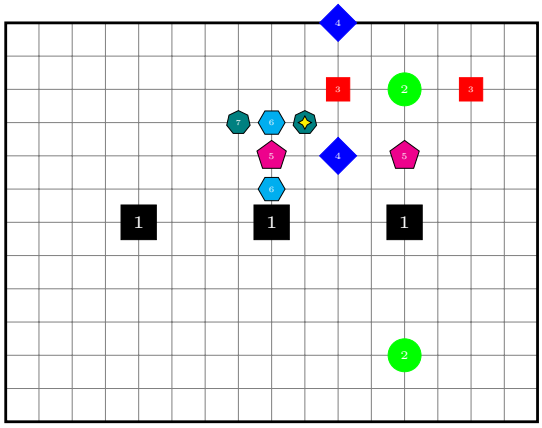
(a) Three step search [KIH+81].



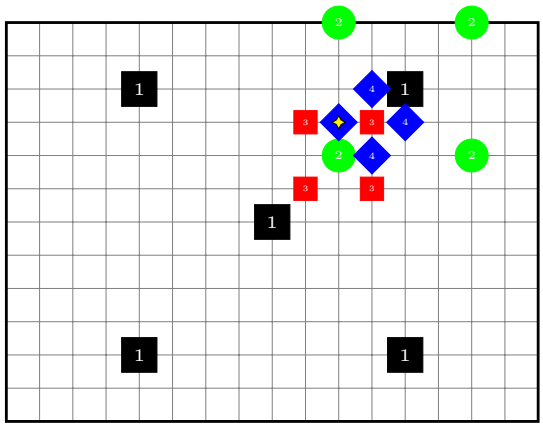
(b) Four step search [PM96].



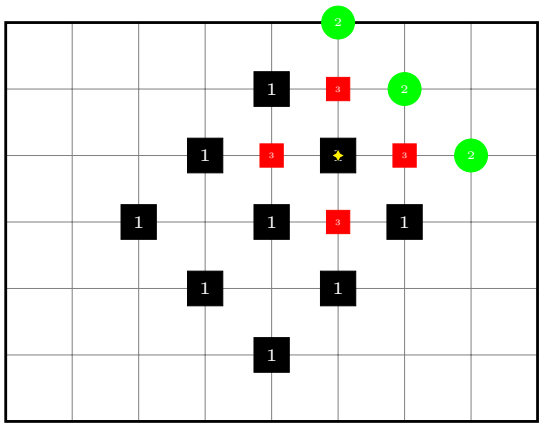
(c) Two dimensional logarithmic search [JJ81].



(d) Orthogonal search [PHS87].



(e) Cross search [Gha90].



(f) Diamond search [ZM00].

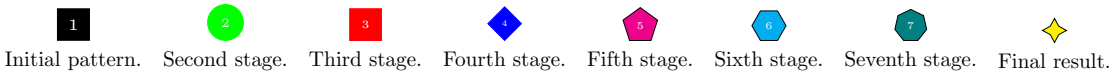


Fig. 2.17 Example of particular paths for some multi-path search strategies.

Early termination. These techniques assume that once a certain match criterion has been achieved a further search is no longer necessary. The match criterion itself can be changed as the search progresses. The most well-known implementation of early termination techniques is the *spiral search* [KDM+05]. This technique assumes that the matches close to zero displacement present the biggest probability of fulfilling the criteria. Spiral search starts at the zero-displacement position and spirals out to cover larger and larger displacement sizes. As soon as the match criterion has been achieved, the search is terminated.

Early termination can also be used to lower the cost of similarity metric computation. In this case, once the threshold of a cost value has been reached the computations are terminated with candidates being rejected. An example of using a threshold for early rejection are the *Sequential Similarity Detection Algorithms* (SSDAs) proposed in [BS72].

The drawback of early termination is that, although it reduces the average execution time (vs the exhaustive search), its total execution time is not deterministic. Moreover, early termination is satisfied with finding a 'good enough' solution. Better solutions may never be found if the threshold is triggered before they are even considered. Sequences with slow motion are searched quickly whereas the search of fast and/or complex motion takes considerably longer. In practice, spiral search requires a time limit to be set. This severely limits its use for real-time processing of sequences with fast movement.

Sub-block registration and adaptive block size. This process allows to adapt the template size to better match characteristics of the actual images. Adaptive template size leads to more accurate registration and in some cases reduces bandwidth and memory requirements. This technique is extensively used in the state-of-the-art video coding/-compression where adaptive block size can be used to provide a better fit within the targeted bandwidth [CJ06, Ric04, Boc09]. The adaptability of the processing increases the irregularity of the control flow and comes at the cost of higher complexity of hardware implementation.

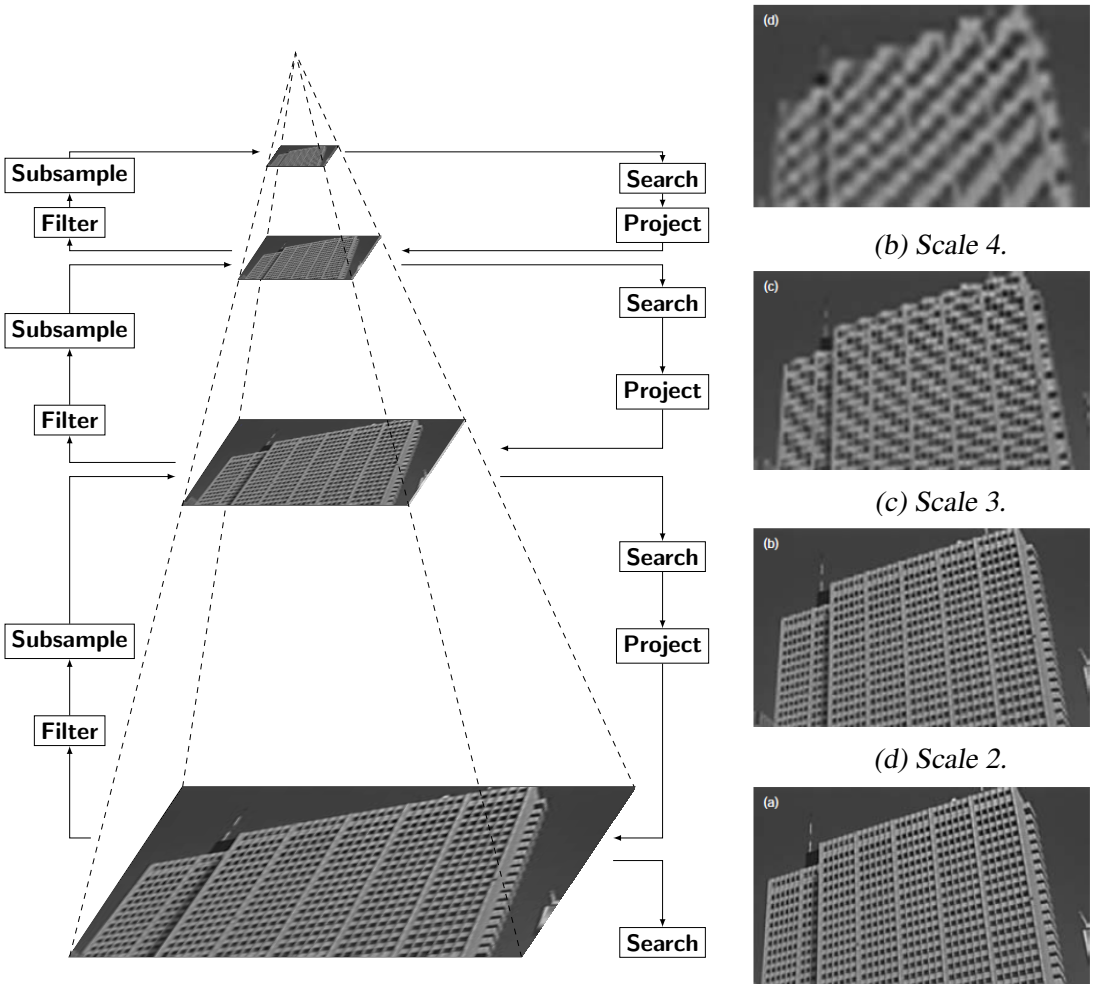
Predictive or spatially dependent registration. These methods assume that the motion patterns of neighboring patches are correlated. These methods look at the results of already carried out matches and form a prediction on the match for the current patch. In most cases the prediction is based on the results of few immediate neighbors. In most cases the predicted match forms an additional test candidate or a special

case to provide earlier termination when favorable conditions are met. An example of the latter use case is presented in [WLBR97]. Spatially dependent registration is extensively used by modern compression codes i.e. H.264 [Ric04] and *High Efficiency Video Coding* (*HEVC*) [SOHW12].

Multi-scale search. Known also as *hierarchical search* or *coarse-to-fine/pyramids* methods. These methods exploit the fact that a displacement observed at a finer-grain representation can be described as a smaller displacement at a coarser level. This allows the search at finer level to be substituted by a carrying out the search at a coarser level that is performed over a smaller set of discrete pels and then projecting the results onto the finer level representation. An optional accuracy refinement step is possible.

In order to carry out the multi-scale estimation, a set of scales —representations of the images being registered— are required. These representations form a hierarchical structures of images referenced as *pyramids*. An example of such hierarchy is shown in Fig. 2.18. Denoting the original images as scale 1 representation, the representations at different scales are created by iterative decimations (filtering and subsampling) of the representation at previous scale. The search is cascaded over all levels, starting from the coarsest one. Any search strategy can be used at any level. However, in most cases full search is carried out at the coarsest level. The results are used to estimate the initial displacement for the next finer level of the pyramid. The search is then repeated. It is a common case that the search at finer level is executed over a much narrower range of displacements.

While carrying out the search over a hierarchy of down-scaled representations is not guaranteed to produce the same result as full search over the source, it usually works almost as well and is much faster. In most cases, the coarse-to-fine estimation due to the search being guided by results at coarser levels increases convergence whose lack is known to be the biggest disadvantage of direct techniques. Performing the search at a coarser level makes the process more robust against fine-grain variations characterized by abrupt movement at different speeds and in different directions, which are very likely to lead to false estimation at finer-scales. However, the search can be potentially misguided, due apparent to false patterns created by distortions originating from the increased amount of aliasing in each layer of the hierarchy. In practice it is hard to use more than two or three levels of a pyramid before important details start to be blurred away. An example of aliasing introduced by the inter-scale filter manifested as moiré patterns can be observed in Fig. 2.18(b) through 2.18(c).



(a) Coarse-to-fine pyramid and execution flow.

(b) Scale 4.

(c) Scale 3.

(d) Scale 2.

(e) Original; Scale 1.

Fig. 2.18 Example of multi-scale template matching.

Since its initial deployment in [VR77], multi-scale processing image registration has gain much popularity. Nowadays, usually referenced as hierarchical motions estimation, it forms one of the key components of the current state-of-the-art image registration techniques [BAHH92, HZ07, WLBR97, LG00, CJ06].

2.4 Super-resolution hardware implementations

As pointed out, software based implementations are not capable of providing SR real-time performance when mapped on the available technology. When introduced, real-time capabilities are expected to broaden even further the range of possible applications and result in enhanced usability and user experience in the cases that do not intrinsically require them.

Hardware implementation has to consider the existing trade-off between the execution speed and the resources required to obtain it. On one hand the processing has to be parallelized to reach satisfactory performance, on the other hand achievable parallelism is limited by the memory required for its exploitation. This section briefly describes the most important approaches to hardware implementation of SR of images. In the case of commercial devices information on the used algorithm are speculations deduced from the (scarce) available data. The characteristics of the hardware implementations executing in the multi-frame context introduced over the last 15 years are shortly summarized in Table 2.3.

2.4.1 Multi-frame SRIR implementations

In [CLL+06] Callico et al. present hardware implementation of two super-resolution image reconstruction algorithms based on non-uniform interpolation referenced as, respectively, *Iterative Super-Resolution* (ISR) and *Non-Iterative Super-Resolution* (NISR). Both of these algorithms execute in a multi-frame context and fall into the family of fusion-based *direct* SR methods working in the spatial domain. The presented implementation is mapped onto a hybrid software/hardware system based on *Commercial Off-The-Shelf* (COTS) Picasso system developed by Philips Research. The Picasso system is composed of a general-purpose embedded ARM microprocessor and four *Very Long Instruction Word* (VLIW) processors. The idea is to take advantage of software/hardware partitioning and output computationally expensive tasks to the VLIW processors acting as hardware accelerators, while memory access and flow control is managed by the ARM processor. In order to carry out SRIR the platform had to be modified by addition of memory, arithmetic unit and implementation of quarter-pixel motion estimation. Mapping results showed that NISR outperforms the iterative approach in terms of execution time, and, if more than four reference frames are used, also in super-resolved image quality measured as peak noise to signal ratio (PSNR). The gain in performance comes at the price of higher memory requirements, which authors consider the architecture bottleneck. Both algorithms implementations were far from meeting real-time execution. The NISR is described as more suitable for systems which aspire to meet real-time requirements.

In [CN07] Callico et al. implement a modified version of NISR on platform comprising an ARM process and *Programmable Logic Device* (PLD). The implemented algorithm, called *eXtended Super-Resolution* (XSR) introduces a new down-sampling method, called *smart down-sampling*. During down sampling image is divided into three regions: motion and textured (MT), flat (F), and no motion but textured (T). Pixels belonging to T regions in

consecutive LR frames are assigned different HR samples values. XSR applied on T regions is able to undo smart down-sampling, as it knows the inverse of sample selection scheme. Flat regions are upscaled using interpolation and MT regions are super-resolved in the regular manner. The algorithm was mapped onto the Integrator CM922T-XA10 development board. The board includes an Excalibur XA10 device and off-chip memory. The former device comprises a programmable logic device, used to accelerate *inverse discrete cosine transform* (IDCT) and handle off-board communication, and an ARM9 embedded processor that manages the rest of the tasks. The described system was capable of super-resolving QCIF (176x144 pixels) resolution frames to CIF (352x288 pixels) resolution at the rate of 15 frames per second. The memory access latency and bus contention have been identified as the system bottlenecks.

In [ABCC09] Angelopoulou et al. present a FPGA implementation of a super-resolution image reconstruction based on iterative back projection that executes in a multi-frame context. In this approach, additional details are reconstructed based on exploitation of sub-pixel shifts caused by warping. In order to facilitate parallelization and minimize the execution context memory occupancy, the processing is done at pixel level by means of weighted mean optical flow, referred to as *weight based* (picture elements) *merging*. Weights estimation is based on the inter-frames motion estimations for pel matching. The implementation reaches an operating frequency of 80 MHz, which the authors claim is sufficient to allow real-time execution outputting 25 VGA 2x super resolved frames. Nevertheless, the output quality is compromised due to the low number of implemented iterations (up to 10) limited by the available resources. The main weaknesses of this approach are its high memory requirements and the fact that, in order to output a super resolved image, multiple passes through the hardware are required. The design bottleneck was identified to be the triple buffering memory access scheme.

In [BB08] Bowen et al. present a similar implementation of IBP algorithm targeting a development board hosting a FPGA device. The architecture onto which the algorithm was mapped is shown in Fig. 2.19. Motion estimation data, pixel values, and weights of aforementioned LR frames are loaded and used to fill-in the HR grids. Those two grids are merged with previous frame initial approximation to form initial approximation for the current frame. Missing pixels are reconstructed by means of modified nearest neighbor interpolation forming an initial approximation of the HR image. This approximation is later fed to the first iterative stage modules. Iterative stage modules carry out the refinement process using values from previous iterations, original pixel, and weight related parameter in similar fashion to the aforementioned weight based merging. Output of the last iterative

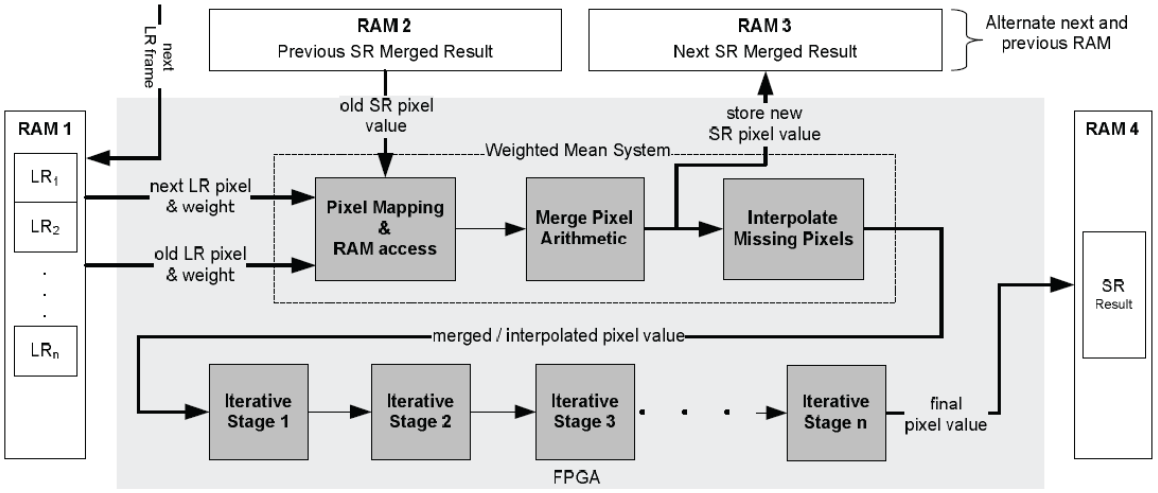


Fig. 2.19 System architecture deployed by [BB08]; source [BB08].

stage is considered the super-resolved pixel. The authors claim, that, with a loaded pipeline, their approach is capable of producing one super-resolved pel per cycle. The described architecture, implementing 10 iteration stages, was mapped onto a Xilinx XC2V6000 FPGA device reaching a frequency of 58 MHz. In this configuration the system was capable of super-resolving 61 CIF formatted LR images to 1280x720 pels per second. Nevertheless, a satisfying quality level requires at least 20 iteration stages, being out of reach for the target device. The number of implemented iteration stages is said to be limited only by the available on-chip memory.

In [STdA13] Singla et al. implement a modified version of the NISR on a low cost NoC-based MPSoC platform comprising up to 4 Xilinx MicroBlaze soft-core processors. The processors are equipped with a 64 kB of local memory and access to DDR3 controller over an Arteris FlexNoc 2D-Mesh NoC. Each core executes its own copy of the algorithm that applies SR processing on a (statically assigned) chunk of the input frame loaded from off-device *random access memory* (RAM). Mapping onto Xilinx Spartan-6 LX45T FPGA results in utilizing 88% of the available slices ($\sim$6003 slices or $\sim$38420 LUTs) and 97% of BRAMs ($\sim$2025 Kb) while reaching operating frequency of almost 53 MHz. For this configuration a complete SR (including ME) of a QCIF frame to CIF (2x SR) takes around 60 seconds.

2.4.2 Single-image SRIR implementations

In [Mal06] Mallat et al. propose a novel approach for carrying out SR in the frequency domain based on bandlets [MP07, PM08]. The algorithm does not rely on motion estimation but rather on a dictionary-based bandlet matching executing in a single-frame context. The meta data describing the patches of the LR image (frequency representation) is produced by means of geometric total variation estimation. Based on these metrics most suitable geometry reconstruction method, out of a provided *a priori* set, is found and applied. The processing differentiates the processing (and the reconstruction sets) between fine grain patches (areas rich in details) and coarse grain patches (plain/derived of details areas). The above-described super-resolution image reconstruction method (with noise removal) based on geometric spatio-temporal bandlets transformation is implemented by *Let it wave* LB-101. The LB-101 is a technology converter for Broadcast/ProAV and Home Cinema applications capable of SD (Standard-Definition) to HD (High-Definition) upconversion as well as a cross-conversion from 1080i to 720p and 1080i to 1080p format. The LB-101 is available as a standalone integrated circuit with reference design, a soft macro IP implementable on Altera FPGAs and/or HardCopy Structured ASICs, or as a complete, easy to integrate module mezzanine board. LB-101M FPGA implementation requires 70000 logic elements (when mapped onto Altera Cyclone-II 70 device) and can be found in high-end electronics devices, i.e. in Analog Way's HD Optimizer.

In [NEC09] NEC has introduced its approach to single-image SRIR, namely the NEC μPD9245GJEC SoC. The device implements NEC Electronics proprietary single-frame super-resolution algorithm with blur reduction. Details on the algorithm have not been disclosed, however, it is most likely to be a dictionary based hallucination in frequency domain similar to the one implemented by the LB-101. NEC claims that its design is capable of upscaling: (i) quarter VGA (QVGA) resolution (320x240 pixels) to wide VGA (WVGA) resolution (800x480 pixels), (ii) NTSC format resolution (720x480 pixels) to wide extended graphics array (WXGA) resolution (1366x768 pixels) or super XGA (SXGA) (1280x1024 pixels), at the rate of 60 frames per second (fps). The μPD9245GJEC is also available as a soft macro destined for NEC's cell-based ASIC (CB-90 and CB-12) and gate array (CMOS-12M) libraries.

Okuhata et al. in [OIOS13] present an image up-converter that claims to deploy 'super-resolution' to provide a more accurate depiction of edge and details than those of conventional interpolation algorithms. The algorithm checks each region in a given frame for presence of edges and then applies distinct interpolation functions to pixels which are part of an edge as well as pixels located in 'smooth' regions of the image. The fact that this algorithm

does not use a dictionary but a static set of coefficients suggests that the algorithm belongs to the reconstruction based single-image family or SR methods. Nevertheless, based on the given details this algorithm can also be classified as an adaptive form of linear interpolation which is capable of selecting a suitable set of convolution coefficients according to the edge orientation in the vicinity of the interpolated pel. When implemented using Verilog and mapped onto an Altera Arria II GX EP2 AGX125 EF35C4 device the implementation supports a maximum resolution of 1920x1080 pixels at a maximum frame rate of 60 fps at a 148.5 Mhz operating frequency.

In [Goh14] Goshi proposes a novel SR method based on non-linear signal processing. The algorithm detects edges in the input frame using a high pass filter and uses them as the input to a non-linear function in order to create harmonic waves that have higher frequency than the LR input. Once saturated by a limiter, the created high frequency details are added to the upscalled version of the original input enhancing its quality. The quality enhancement claim is justified by presentation of 2DFFT results that show additional high frequencies in the output image. No complementary objective quality assessment values are presented. The author claims successful implementations in FPGAs but does not provide any implementation data justifying the claim.

TABLE 2.3 *Summary of hardware implementations executing in multi-frame context presented in Section 2.4.*

Contributor	Year	Method	Platform	Input	SR	fps	LUTs	BRAM	Limitations
Callico [CLL+06]	2006	NISR ISR	Picasso COTS* (ARM+VLIW)	N/A	N/A	N/A	N/A	N/A	Memory occupancy and access latency. Implements ME.
Callico [CN07]	2007	XSR (NISR)	Excalibur XA10 (ARM+PLD)	QCIF	2x	15	N/A	N/A	Memory access latency and bus contention.
Angelopoulou [ABCC09]	2008	IBP	Celoxica AD-MXRC4SX (Virtex-4 FPGA)	VGA	2x	25	5000+	150+	Limited SR quality, memory access buffering scheme, multi-pass nature.
Bowen [BB08]	2008	IBP	Xilinx XC2V6000 (FPGA)	CIF	4x	61	35707	∼2400 Kb	Limited SR quality due to limited resoureces (memory).
Singla [STdA13]	2013	ESR (NISR)	Xilinx S6LX45T (FPGA)	CIF(/HD)	2x(/4x)	0.02	∼38420	∼2025 Kb	Software running on soft-core processors.

2.5 Conclusions

This chapter focused on the presentation of the basic concepts and the state-of-the-art of the super-resolution process. Over the last 40 years several approaches to the super-resolution problem have been developed. The most important approaches were described and organized in a complete taxonomy presented in this chapter. Super-resolution of video sequence introduces an additional level of complexity to the already complex super-resolution problem. The challenges and approaches to super-resolution of video sequences have been introduced. The non-uniform grid projection algorithm that is of relevance to this thesis has been contextualized as belonging to the interpolation-based family of the direct classical multi-frame super-resolution algorithms. The main advantage of this class of SR algorithms is their relatively low computational load, which is essential in making them suitable candidates for real-time implementations in hardware.

Finally, the chapter concluded with the presentation of the state-of-the-art of the super-resolution hardware implementations in FPGAs. The availability of such implementations is scarce. The main factors limiting the success of FPGA-targeted implementations have been identified as being resource-related. In particular: (i) Most implementations have been found to be limited by the available device memory. (ii) Iterative algorithms implementation have been found to be additionally limited by logical resources and tend to offer lower-then-the-reference output image quality due to the limited number of implemented iterations. In spite of the above drawbacks, FPGA devices are still the best platform for multi-frame SR systems prototyping in hardware.

Chapter 3

The non-uniform grid projection algorithm

3.1 Introduction

In this work we tackle the challenge of providing real-time SR of video sequences by using the non-uniform grid projection algorithm that has been proposed in [MC03]. NUGPA is a fusion algorithm that looks for and exploits the non-redundant data encountered in a set of images due to the warping associated with movement and aliasing caused by the band-limited registration sensors. Using the classification presented in Section 2.2.3, NUGPA falls into the interpolation-based fusion techniques of the direct multi-image family of spatial domain methods. The main advantage of this class of SR algorithms — the relatively low computational load, which is essential in making real-time applications possible — comes at the price of a limited degradation model. Moreover, the optimality of the reconstruction algorithm is not guaranteed, since the reconstruction step ignores the errors that occur in the interpolation stage. In this work we assume that the blur, noise characteristics, point spread function and decimation factor is common and space invariant in all LR images. Additionally, we will consider the relative motion to be purely translational.

Over the years two versions of NUGPA have been developed by IUMA in cooperation with Philips research, namely, the iterative and non-iterative (or single-pass) versions. Based on the comparison of both versions presented in [MC03] we have opted for using the non-iterative version due to its deterministic execution time and better quality of the super-resolved image. Additionally, implementations of iterative algorithms (e.g. iterated back projection) have been reported to not be able to implement sufficient number of iterations in order to provide satisfactory quality of the output image [ABCC08, BB08]. The single-pass

nature of the non-iterative NUGPA approach not only facilitates hardware implementations but also increases the chances of providing software-level quality of the output. The other main issue of the FPGA-targeted SR implementations, namely high memory requirements, will be tackled in the following chapter.

3.2 The non-uniform grid projection algorithm

NUGPA carries out super-resolution image reconstruction in accord with the three-stages classical flow of fusion-based algorithms presented in Fig. 1.1. The stages defined by the classical approach, namely (i) pre-processing, (ii) multi-image fusion, and (iii) post-processing, correspond to the motion estimation, fusion-based reconstruction and non-uniform interpolation stages of the NUGPA, respectively. Due to this mapping, NUGPA is an example of the interpolation-restoration direct methods. Detailed description of execution flow is the focus of this section.

3.2.1 Image registration

In the first stage the warping function and its metrics are estimated. The warping function usually is not known *a priori*, hence, regions which could contain supplementary information have to be found at run-time. What is known is that these regions are believed to differ only slightly from regions that they could enrich. In order to register images and find regions that are most probable of containing additional information the NUGPA super-resolution algorithms deploys a variant of block matching motion estimation. During image registration a set of neighboring pels (hereafter a *macro-block* (*MB*)) from the processed frame is matched against pels from other frames from the sliding frame window. For each MB only a limited set of pels (called *search area*, SA) confined within certain spatial vicinity (defined by the so called *search area radius*, SAR; expressed in number of pels) participate in the process of candidate set creation during motion estimation. The motion estimation process calculates the so called *hyperdata*, which in our case comprise the *motion vector* (MV) and the associated similarity criteria values. The motion vector is a vector that identifies the MB for which the similarity indices have the value closest to the optimum being sought for. An example of block-matching for a SFW comprising two reference frames, search area radius equal to MB width (MB_{width}), comprising $(2 \times MB_{width} + 1)^2$ candidates, is illustrated in Fig. 3.1 (for clarity only nine candidates are shown).

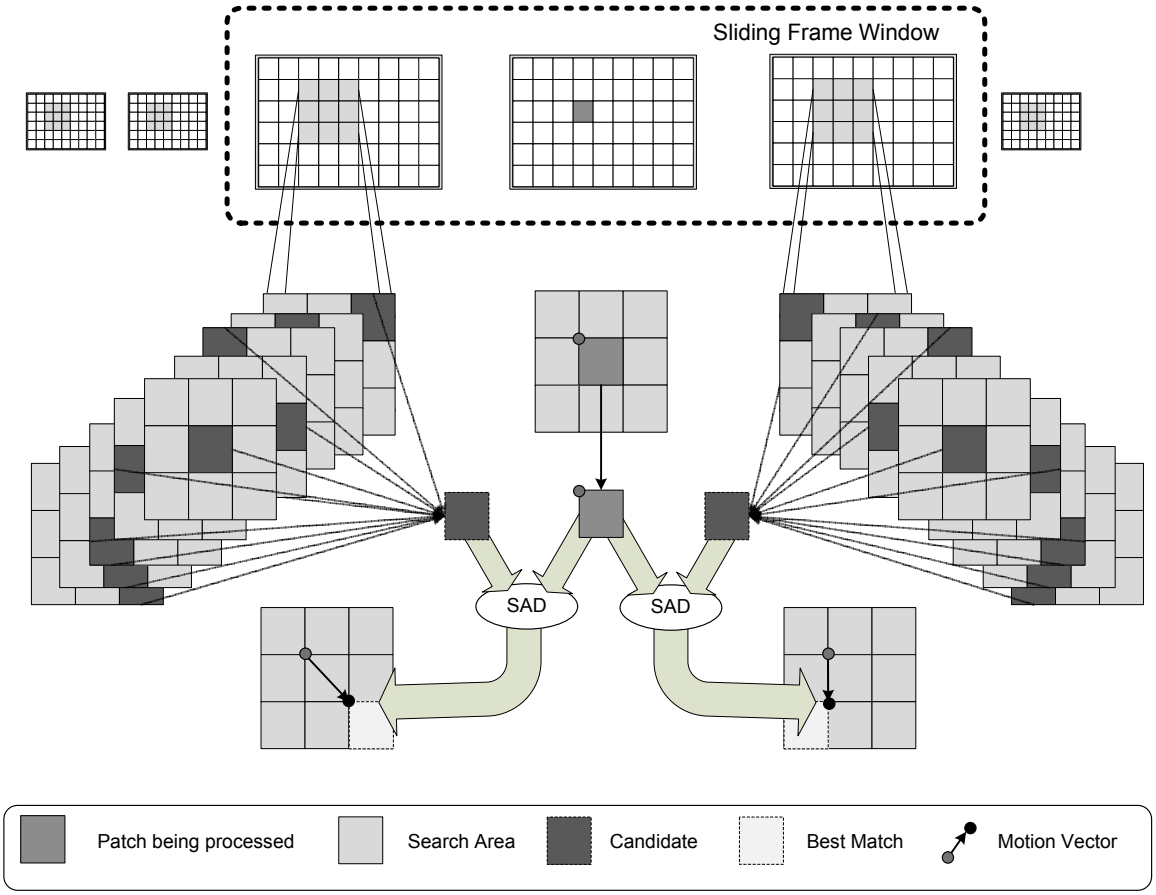


Fig. 3.1 Image registration using block matching motion estimation with sum of absolute differences (SAD) used as the similarity criterion.

3.2.2 Fusion-based reconstruction

The *hyperdata* produced during the motion estimation stage are passed on to the SR kernel. The restoration processes starts with HR grid creation. The dimensions of this grid are determined by the used image registration precision (hereafter $precision_{ir}$ or $precision_{me}$). First, the so called *up-holes* transformation is carried out. During this transformation a HR grid gets filled with LR pels of the frame being super-resolved. Having $g(x,y,t)$ representing pixels of the LR frame captured at time instance t, with spatial coordinates (x, y), the new HR spatial coordinates $(\overline{x},\overline{y})$ are computed by multiplying the LR coordinates by the motion estimation precision, in accord with (3.1).

$$(\overline{x},\overline{y}) = (x * precision_{ir}, y * precision_{ir}) \tag{3.1}$$

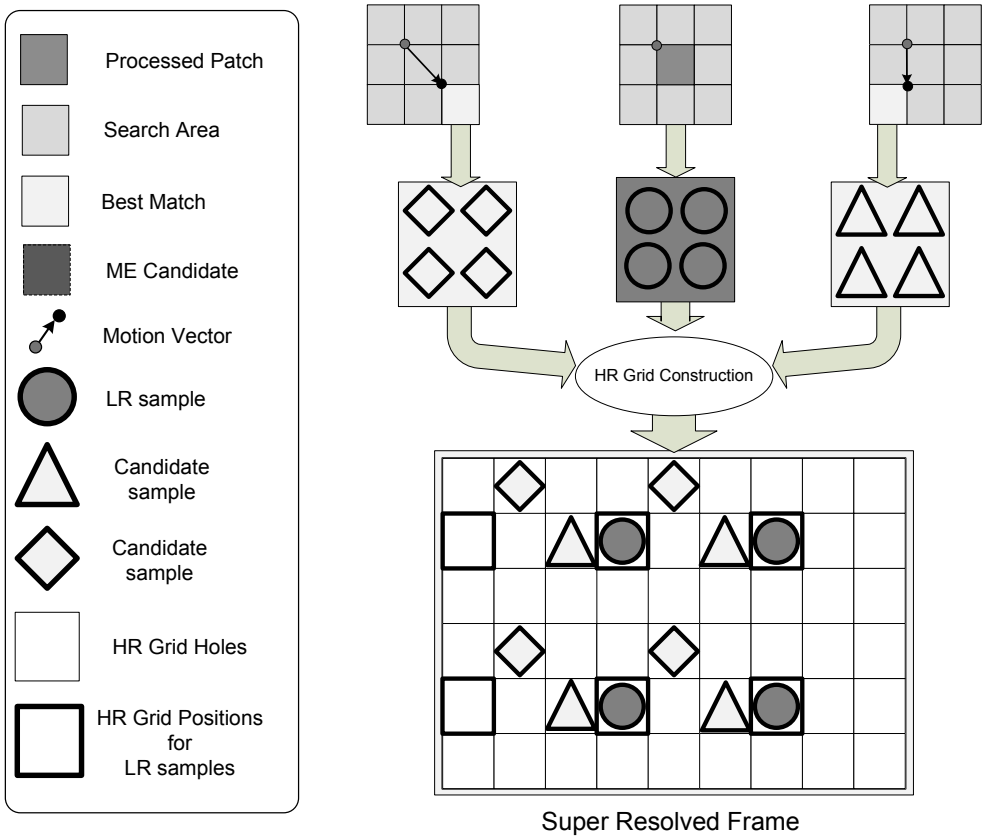


Fig. 3.2 The NUGPA super-resolution kernel execution flow.

Having placed all LR pixels on the HR grid, pixels from the set of preceding and succeeding LR frames contained in the SFW are considered. At this point, most of the HR grid coordinates do not contain valid data. Those coordinates are referred to as *holes*, and are to be fused with data extracted from the current frame window. During the fusion, the received hyperdata (i.e. MVs and SADs) determine which data will contribute to which hole value estimation, and with what weight. It is allowed for more than one value to take part in the process of forming the new super-resolved value. In the case of more than one pel value contributing to the process, each contributing pel value is multiplied by its weight and added together to form an intermediate super-resolved value. For each coordinate, a sum of weights of the contributions is maintained and updated. Having considered all frames from the frame window, the final holes values are computed by dividing the intermediate values by the sum of weights associated with them. The data extracted from reference frames are allowed to be used to modify only the holes values. This guarantees that the beforehand placed LR values remain unchanged during the restoration stage. Execution flow of data extraction, HR grid filling and fusion is illustrated in Fig. 3.2.

3.2.3 Non-uniform interpolation

A common scenario is that not all holes of the HR grid are filled during the reconstruction stage. Unfilled positions values are estimated by means of interpolation. Finally, the post-interpolation HR grid is adjusted to the expected outcome dimensions indicated by the scale factor (hereafter *scale* or $scale_{sr}$). The scale determines the relation between the expected super-resolved outcome and the LR input dimension. For scale values smaller than the $precision_{ir}$ value, HR grid downsampling is carried out. When the aforementioned parameters values are equal, no further processing is needed and the post-interpolation HR grid becomes the final super-resolved image produced by NUGPA.

3.3 Mathematical model of the non-iterative restoration-interpolation classical super-resolution process

We have created in chapter 2 a complete taxonomy of different approaches and algorithms developed in the literature for the SR process. In this section we introduce mathematical models for the image registration stage and the restoration-interpolation stage in an attempt of complementing existing algorithmic descriptions with a consistent closed form analytical model that captures the layered computation scheme of the SR process. The different transformations operating over different image spaces are clarified and underlined.

In this section a mathematical description of the interpolation-restoration non-iterative super-resolution is presented. Before we start exposing our analytical approach to the super-resolution process, let us define the following terminology:

- $precision_{ir}$: precision of the image registration (motion estimation) process that is a value from a set $P_{ir} = 2^i : i \in N$ (only powers of 2).

- $precision_i$: precision of the i^{th} intermediate stage of image registration where $i : i \in N \wedge 1 \leq i < \sqrt{precision_{ir}}$ and $precision_i = 2^i$.

- $scale_{sr}$: scale of the increase in number of pixels (per dimension) between the input and the output of the super-resolution system such that $scale_{sr} \leq precision_{ir}$.

- $card(Q)$: cardinality of a set Q defined as the number of elements it contains. The traditional notation $|Q|$ is not used in order to avoid ambiguity with absolute value determinant.

- X^2*:* coordinate space in two dimensions in which image registration is carried out at its lowest level of accuracy (full-pixel), such that $X^2 \subset N^2$.
- $\overline{X}^2$*:* coordinate space in two dimensions in which image registration (motion estimation) is carried out at its highest level of accuracy specified by the value of the $precision_{ir}$ parameter, such that $\overline{X}^2 \subset N^2 \wedge card(\overline{X}^2) = card(X^2) \cdot precision_{ir}^2$.
- $\tilde{X}_i^2$: coordinate space in two dimensions in which the i^{th} intermediate stage of image registration (motion estimation) is carried out at the level of accuracy specified by $precision_i$, such that $\tilde{X}^2 :\subset N^2 \wedge card(\overline{X}^2) = card(X^2) \cdot precision_i^2$.
- $\hat{X}^2$: coordinate space in two dimensions in which corresponds to the fidelity of the final representation of the super-resolved image such that $\hat{X}^2 :\subset N^2$ and $card(\hat{X}^2) = card(X^2) \cdot scale_{sr}^2 \wedge card(\hat{X}^2) = card(\overline{X}^2) \cdot \frac{scale_{sr}^2}{precision_{ir}^2}$.
- *x, y:* coordinates describing the location of a pixel in X^2 space such that $(x,y) \in X^2$.
- $\tilde{x}_i, \tilde{y}_i$*:* coordinates describing the location of a pixel in $\tilde{X}_i^2$ space such that $(\tilde{x}_i, \tilde{y}_i) \in \tilde{X}_i^2$.
- $\overline{x}, \overline{y}$*:* coordinates describing the location of a pixel in $\overline{X}^2$ space such that $(\overline{x}, \overline{y}) \in \overline{X}^2$.
- $\hat{x}, \hat{y}$*:* coordinates describing the location of a pixel in the space of the super-resolved image such that $(\hat{x}, \hat{y}) \in \hat{X}^2$.
- κ*:* maximal number of low resolution images comprised in the frame window.
- $p_t(x,y)$*:* picture element (pel or pixel) sampled at the moment *t* located at the coordinates (x,y).
- δ_x: value of displacement along the horizontal axis.
- δ_y: value of displacement along the vertical axis.
- $\overrightarrow{\Delta} = (\delta_x, \delta_y)$: displacement vector along the horizontal and vertical axis in a 2-D space.
- $\delta_x(x,y)_{(L \to P)}$: displacement along the horizontal axis of pixel located at (x, y) of frame 'P' with respect to the reference frame 'L'.
- $\delta_y(x,y)_{(L \to P)}$: displacement along the vertical axis of pixel located at (x, y) of frame 'P' with respect to the reference frame 'L'.

- $\overrightarrow{\Delta}(x,y)_{(L\rightarrow P)} = (\delta_x(x,y)_{(L\rightarrow P)}, \delta_y(x,y)_{(L\rightarrow P)})$: displacement vector in a 2D space of pixel p(x, y) of the frame 'P' with respect to the reference frame 'L'.

- Based on the above we find that the following relations are true:

$$card(X^2) < card(\tilde{X}^2) < card(\overline{X}^2) \tag{3.2}$$

$$card(X^2) \leq card(\hat{X}^2) \leq card(\overline{X}^2) \tag{3.3}$$

3.3.1 Overview of the non-iterative restoration-interpolation classical multi-image super-resolution

Calling f(x,y) the underlying continuous image, let us denote *f(x,y,t)* the low resolution input image captured at time instance t. Now, let us assume that all the input sub-system effects (lenses filtering, chromatic irregularities, sample distortions, information loss due to format conversions, system blur, etc.) to be time invariant and encapsulated in *h(x,y)*. So, assuming linear effects in lens, sensors, and color processing, the input to the algorithm will be the two dimensional convolution expressed as

$$g(x,y,t) = f(x,y,t) **h(x,y). \tag{3.4}$$

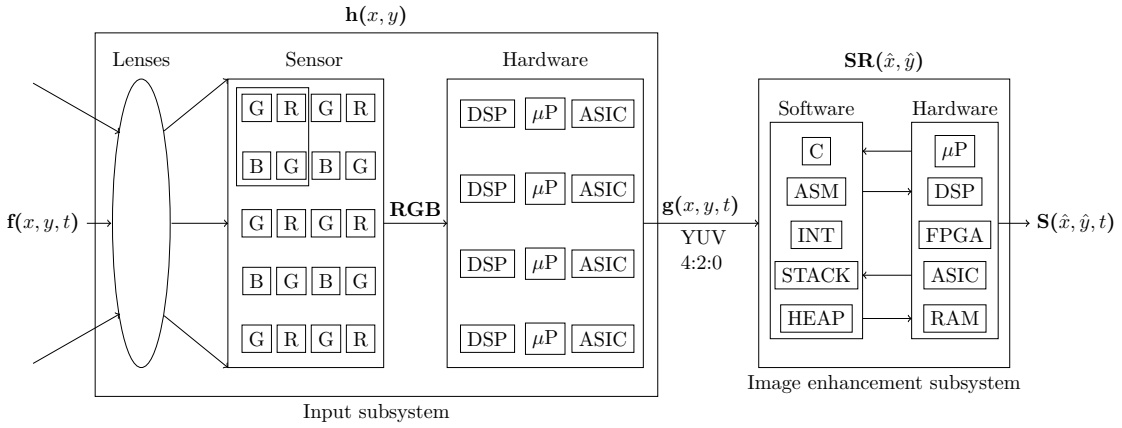
Calling $S(\hat{x},\hat{y},t)$ the image obtained after applying the SR algorithm, and $SR(\hat{x},\hat{y})$ to the SR algorithm itself, the input and the output of the SR processing are related as

$$S(\hat{x},\hat{y},t) = g(x,y,t) **SR(\hat{x},\hat{y}) \tag{3.5}$$

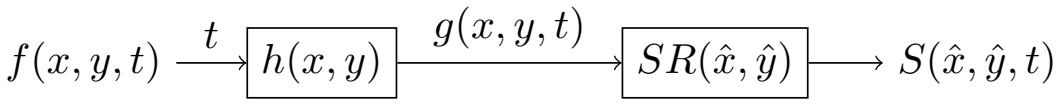
The typical SR algorithm itself can be seen as composed of two stages: (i) image registration and (ii) restoration-interpolation. This is modeled as

$$SR(\hat{x},\hat{y}) = SRP(r(x,y),k(x,y)) \tag{3.6}$$

where, *SRP* is an operator that represents the SR process that encapsulates the processing carried out by image registration kernel $r(x,y)$ and the *super-resolution kernel* (SRK) represented by $k(x,y)$. Both of theses operations are the focus of the following sections. All of the relationships between the defined operations for real and simplified systems are summarized in figures Fig. 3.3(a) and Fig. 3.3(b), respectively for the case of real-life and



(a) Real system.



(b) Simplified model.

Fig. 3.3 Organization of super-resolution image enhancement systems. Based on [MC03]

simplified systems.

Multi-image super-resolution image reconstruction (SRIR) is based on the fact that due to aliasing images of the same scene captured at different time instances can contain complementary information should sub-pixel warps be observed between theses representations, as explained in Section 2.2.2.2.1. As the algorithm executes in a multi-images context, the SRIR processing starts with the algorithm performing image registration followed by a search for the regions captured at different time instances that are most likely represent the same regions of the scene sampled with sub-pixel warps.

In order to be able to detect the sub-pixel movement and find additional information the cardinality of the space $(\overline{X}^2)$ of image registration has to be higher than the one of the input, that is, the registration process has to be done in a space oversampled with respect to the input. This leads to the aforementioned assumption that $card(\overline{X}^2) > card(X^2)$. Thus, the first aspect to notice is that there are several coordinate spaces with different cardinality. In our case the algorithm works with four resolutions: the low resolution space $(X)^2$, the intermediate-accuracy image registration space $\tilde{X}^2$, the finest-accuracy image registration space $\overline{X}^2$ and the super-resolution space $\hat{X}^2$. Thus, we have to model the transformations between the defined coordinate spaces. To this end we use two operators: the *downsampling*,

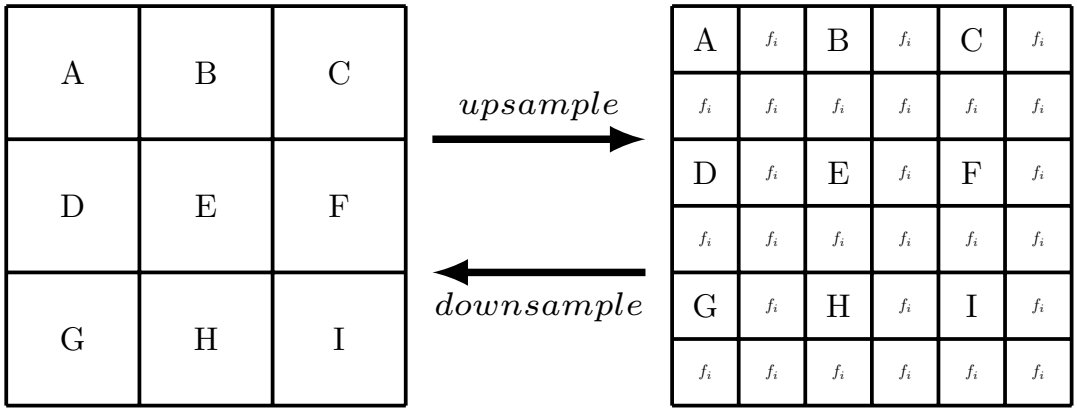


Fig. 3.4 Effect of using the upsampling operator on a 3×3 image and downsampling a corresponding 6×6 image (scale change factor of two; f_i represents values estimated using interpolation).

and the *upsampling* operators. The former carries out the transition from higher to lower cardinality and in the case of transformation from $(\overline{x},\overline{y})$ to $(\hat{x},\hat{y})$ is modeled as

$$downsample(f(\overline{x},\overline{y}))^d = \{f(\overline{x},\overline{y}) : \overline{x} = \hat{x} \cdot d \wedge \overline{y} = \hat{y} \cdot d \wedge \hat{x},\hat{y} \in \hat{X}^2 \wedge \overline{x},\overline{y} \in \overline{X}^2)\}, \quad (3.7)$$

where d is the scaling factor in each dimension defined in this particular case as

$$d = \sqrt{\frac{card(\overline{X}^2)}{card(\hat{X}^2)}}. \quad (3.8)$$

The latter is used for transformations from lower to higher cardinality spaces (in our case from (x,y) to $(\overline{x},\overline{y})$) and is modeled as

$$upsample(f(x,y))^s = \begin{cases} f(x,y) : x = \overline{x}/s \wedge y = \overline{y}/s & \text{if } x,y \in N \\ f_i(\overline{x},\overline{y}) & \text{otherwise} \end{cases}, \quad (3.9)$$

where f_i is an arbitrary function and s is the scaling factor in each dimension defined in this particular case as

$$s = \sqrt{\frac{card(\overline{X}^2)}{card(X^2)}} \quad (3.10)$$

The relation between the upsampling and downsampling operators are illustrated in Fig. 3.4.

3.3.2 Image registration for the non-uniform grid projection algorithm

The choice of the appropriate image registration method is made per application. The selected image registration has to meet the registration requirements in terms of accuracy and execution time, especially if real-time execution is being targeted. Our implementation of the non-uniform grid projection algorithm carries out image registration by deploying *block matching*—a direct area-based image registration in spatial domain with piece-wise mapping. The used block-matching implementation limits the transformations model to purely translational ones. This is a common design choice for systems aimed at real-time processing. More complex transformations models (e.g. affine) prove to be simply beyond capabilities of today's customer grade hardware. In fact, even real-time registration of transformations with 2 DoF can be a very demanding task, especially in cases of large search areas traversed using not efficient search strategies.

3.3.2.1 Provision of sub-pixel accuracy

The non-uniform grid projection algorithm falls into the category of classical multi-frame super-resolution. As aforementioned, these algorithms are based on exploiting aliasing and/or variations created due to sub-pixel variations between the captures. Thus, in order to be able to provide any enhancement these algorithms require that the registration be carried out with higher-than-full-pixel accuracy. The techniques described up till now have been designed to carry out the registration without considering the sub-pixel domain. Even though, a sub-pixel estimate could be created based on the results of the full-pixel ones (e.g. weighted average of a set of 'best' results), the accuracy offered by this approach is in most cases insufficient. There are several techniques that allow to obtain a better accuracy [TH86, GSP86]. One common approach, and the one used by the authors of this work, is to modify a well known full-pixel technique and perform (some of) its steps at a finer, sub-pixel level. This approach operates on a pyramid of images where the search is cascaded through different accuracy levels. The representations of each level of the pyramid are created by re-sampling the previous coarser-level representation. Alternatively, interpolation can be avoided by using re-sampling based on Taylor series approximation. However, this approach is too complex to allow real-time execution.

In our implementation, sub-pixel accuracy has been provided by preforming the search in three steps cascaded over three levels of accuracy. The estimations are cascaded from coarser-to-finer. Before being used, the results from the coarser-level are projected onto the finer-grain coordinates space. These coordinates after the projection are used as the refer-

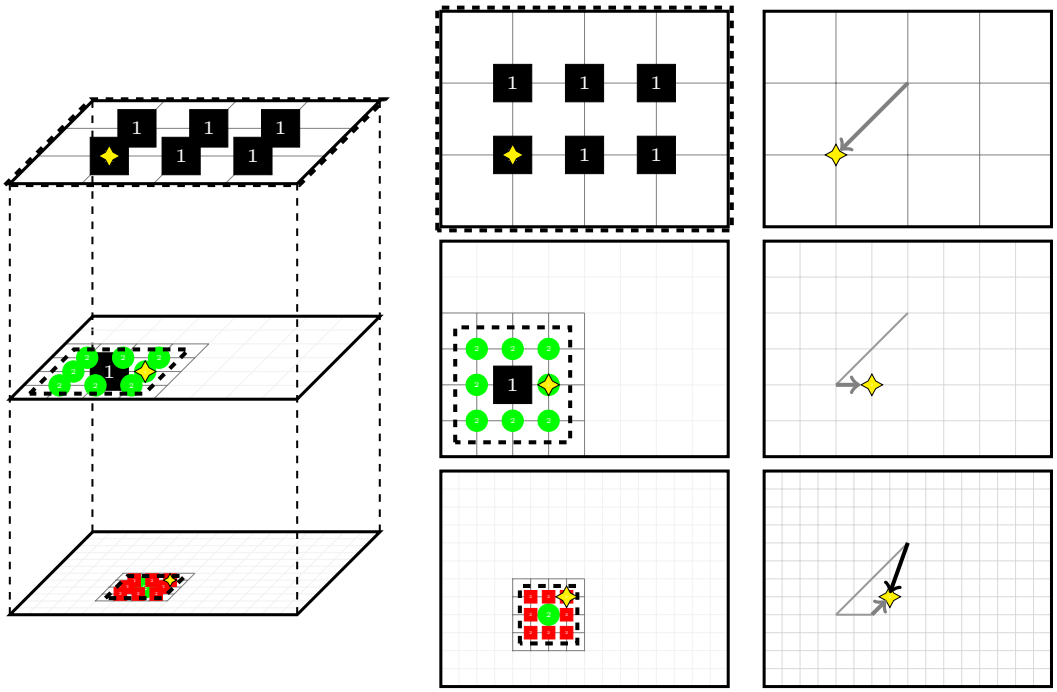


Fig. 3.5 Image registration using iterative refinement across three scales.

ence point of finer-level search area. The search area encompasses coordinates that correspond to fractional coordinates at the coarser-level. The missing values of these coordinates, the so-called *holes*, are estimated using bilinear interpolation. Once the re-sampling is done the search continues. The coarsest-level corresponds to the full-pixel granularity search algorithm as the first stage of motion estimation, followed by a sub-pixel motion refinement. An example of a particular search carried out using the implemented block-matching algorithm is illustrated in Fig. 3.5. In order to estimate the hyperdata (results of the registration process) with quarter-pixel accuracy the searches are cascaded over three levels. Each finer-granularity level corresponds to re-sampling with a magnification factor of 2.

3.3.2.2 Similarity criterion

The last component of the image registration puzzle is the selection of the similarity criterion. In this work, we carry out the similarity criterion selection under the assumption that the images being registered have been captured with insignificant variances in intensity. This is the typical case. This assumption allows to avoid normalization which is required if the above is assumption is not made. The most widely used similarity criterion for correlation-based image registration in spatial domain, among others, are the: *Cross-*

Correlation Function (CCF), *Mean Square Error* (MSE), *Mean Absolute Error* (MAE), *Sum of Absolute Differences* (SAD) and the *Pixel Difference Classification* (PDC).

Cross-correlation is the basic criterion. It has proved itself to be useful for images which are misaligned by small rigid or even affine transformations. The problem with CCF and MSE is that, both of these criteria require square root computations, which implementation in hardware is costly. The criteria based on l_2 norm penalize heavily large projection errors. These errors are usually caused by variations that should be eliminated (e.g. outliers), but are also related to aliasing in higher frequencies. By suppressing the use of these data we may effectively limit the availability of the complementary information on which SRIR operates. The MAE, SAD and the PDC criteria are based on computing the absolute differences, making them computationally much simpler and easier to implement in hardware. Moreover, these methods deploy l_1 norm which can handle outliers much better than l_2 and is more robust. The possible disadvantage of these functions is the fact that they are not differentiable at the origin. This makes them not well suited for gradient-descent approaches. This is not the case of the modified three-step search. A crucial disadvantage of the PDC is that its use results in many 'ties' that require additional computations to be broken. Also, when compared with SAD, it requires additional comparisons even in the tie-less scenario. The fact that SAD neither requires the final division nor has to deal with fractional values (and their representation) makes it the preferred similarity criteria for hardware implementations.

3.3.2.3 Compilation determinable parameters

Current version of the implemented block-matching defines a set of customizable parameters. These are referenced as the (template) Macro-Block width (*MB width* or MB_{width}), *Search Area Radius* (*SAR*), the *registration precision* ($precision_{ir}$) and the number of sensed images used in registration (simply, *reference frames* (*RF*)).

Registration precision defines the finest-level accuracy, and indirectly the number of levels in the search hierarchy/pyramid. The reference software operated at quarter-pixel accuracy, corresponding to registration precision of 4 ($precision_{ir} = 4$).

Setting the value of the macro-block width allows to choose the size used to tessellate the image (and matching). The size of MB significantly influences registration accuracy, as well as the computational and memory requirements. Smaller MB size relaxes memory requirements, at the cost of higher computational complexity and higher probability of 'false' motion vectors [BAA05]. This macro-block width is defined at compilation time and can be set to one of the supported values (4, 8 or 16). The chosen value is interpreted as the

one dimensional span of the template expressed in (full) pels, that is, as the number of pels in horizontal/vertical direction that make up the template. Current version supports only square-shaped templates (block). Run-time variable template size is not yet supported.

NUGPA specifies two dimensions of search areas: one with fixed and with one parametrizable number of candidates. Both of the templates include all of the pels of the candidate template, but (can) differ in the number of included pels from its immediate vicinity. The fixed template includes only the pels which are the immediate neighbours of the candidate template. This effectively fixes the maximal number of candidates to $4 \times MB_{width} + 4$. The customizable template allows inclusion of a larger range of the candidate template neighbors. The maximal distance at which a pel will be included in the search area is defined by the value of the search area radius parameter. The customizable template is used to define the search area used for matching at the full-pixel (coarsest) level. The candidates set is built by picking up every pel of a spiral-shaped path around the initial guess pixel (upper left corner of the block), turning SAR times, and starting with the top-left pixel in every turn. The cardinality of the set of candidates is modeled as

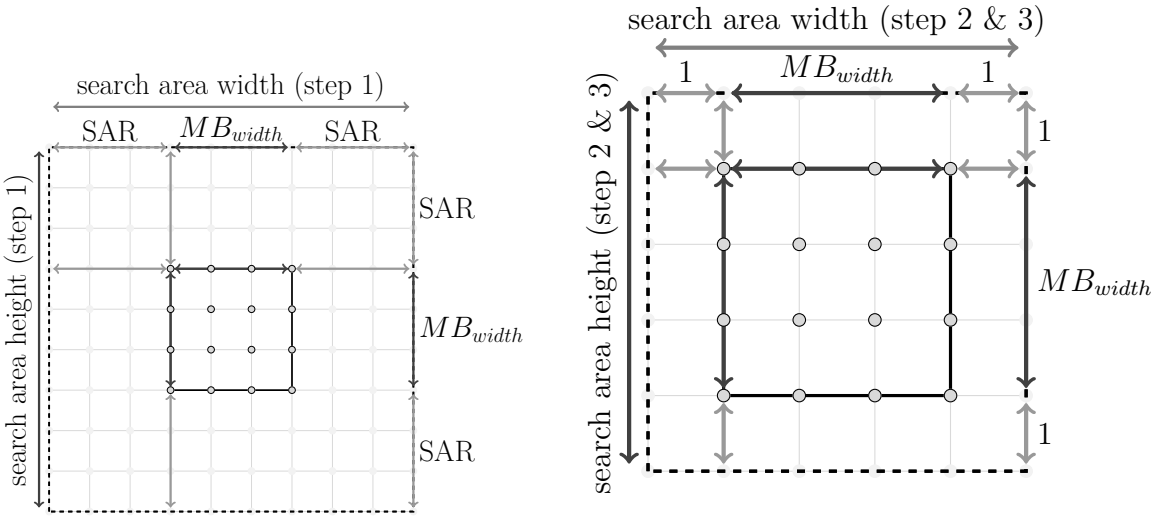
$$card(C^p) = 1 + 2 \times precision_{ir} \times (1 + 2 + \ldots + SAR) = 1 + 2 \times precision_{ir} \times SAR \times (SAR + 1), \tag{3.11}$$

where C^p represents the set of candidates of the parametrizable search area. The fixed template is used for performing the second and thirds step of search. Both of the defined templates are illustrated in Fig. 3.6 for the case of using a square-shaped template of MB_{width} width and a search area radius of SAR pels.

As aforementioned, the number of sensed frames is specified by setting the value of the RF parameter. Our algorithm uses the sliding-window-frame approach in which the registration process is carried out for each possible combination of the source and sensed images from the frame window. Thus, the time required for image registration grows proportionally with the number of used sensed images. The same frame window is used by the super-resolution process that follows image registration. Thus, this value has a significant impact not only on the execution time of image registration but also on the quality of the super-resolution process as a whole.

3.3.3 Mathematical model of the image registration stage

As aforementioned, area-based image registration in spatial domain is carried out by means of template matching. In this work only the case of purely translational model in 2D space with piece-wise (block) mapping is considered. In this case, image registration is modeled



(a) Step 1 (customizable; full-pixel). *(b) Step 2 & 3 (fixed; sub-pixel).*

Fig. 3.6 Search area templates used by different steps of the implemented matching.

as a function of a set T of N blocks $\tau_i, i \in [1,N] \wedge \in N$, a search space described as candidate set C with M elements, and a similarity function ε. The outcome of image registration, called the *hyperdata*, contains (i) a set of the minimal value of the dissimilarity criterion $e_{(\tau \to C)}$, (ii) a set of displacement vectors $\overrightarrow{\Delta}_{(\tau \to C)}$ and (iii) auxiliary data. The hyperdata uniquely identify, for each of the defined template blocks, the candidate for which the minimal dissimilarity score has been observed. In classical SR the sets T (templates) and C (candidates) are exclusive and refer to regions of images of the same natural scene captured observed with different viewing conditions (temporal or spatial).

Before the matching takes place, image registration process starts off by creating higher resolution representations of the input images. To this end, all the images undergo the upsampling transformation with the f_i encapsulating an interpolation algorithm and scale $s = \sqrt{\frac{card(\overline{X}^2)}{card(X^2)}}$:

$$g(\overline{x},\overline{y},t) = upsample^s(g(x,y,t)). \tag{3.12}$$

Regions of these representations obtained through tessellation will be used as the templates τ being registered.

3.3.3.1 Modeling exhaustive optical flow matching

The modeling will start off with presentation of the model of exhaustive template matching of a single pel. This model will be then gradually extended in order to model the version

of block-matching used by the non-uniform grid projection algorithm implementation. The exhaustive template matching of a single pel describes the case in which the template is a singular pixel and the candidate set is the whole sensed frame (effectively $N = M$). Let us define ε as a function that given intensities of a pair of pels returns a value that measures their similarity, being the pair a template-pel $p_\tau : \tau \in [1,N]$ located at $(\overline{x},\overline{y})$, and a candidate-pel $p_c : c \in [1,M]$ from the candidate set C located at coordinates $(\overline{x}+\delta_{x_c},\overline{y}+\delta_{y_c})$. Then, this value, denoted as $e_{(\tau\to c)}$, for the assumed purely translational model, is modeled as

$$e_{(\tau\to c)} = \varepsilon_{(\tau\to c)}(p_\tau(\overline{x},\overline{y}), p_c(\overline{x}+\delta_{x_c},\overline{y}+\delta_{y_c})). \tag{3.13}$$

The dissimilarity criterion is constructed in such a way that it assumes lower values for 'better' matches. Thus, the 'best' match for the template from the candidate set of pels is found by minimizing the dissimilarity function computed over all the pels of the candidate set (exhaustive search). Considering (3.13) in the context of all $c \in [1,M]$, it can been seen that in the investigated case the candidate set corresponds to the whole search space and is exhaustively traversed by using all possible displacements between the template and the candidate pels. Let us define a set of all possible displacements Δ_C^τ that uniquely identifies each and every candidate from the candidate set C for a template τ. Then, the search for the best match of a single pel as the search for a displacement vector $\overrightarrow{\Delta}_{(\tau\to C)}$ for which the dissimilarity function reaches its minimum can be modeled as:

$$\overrightarrow{\Delta}_{(\tau\to C)} \triangleq arg \min_{\Delta=(\delta_{x_c},\delta_{y_c})\in\Delta_C^\tau} \varepsilon_{\tau\to c}(p_\tau(\overline{x},\overline{y}), p_c(\overline{x}+\delta_{x_c},\overline{y}+\delta_{y_c})). \tag{3.14}$$

3.3.3.2 Modeling exhaustive region-based matching

Now, let us relax the assumption that the template contains only one pixel. This requires changes in the similarity criterion, so that it would allow all the pels contained within the limits of the pattern to contribute to the overall similarity score (sub-sampling is not considered here). Let us define a region $W(\overline{x},\overline{y})$ as a local neighborhood centered at coordinates $(\overline{x},\overline{y})$ and comprising pels whose coordinates are within the distance or radius of m pels $[\pm m,\pm m]$ from the center of the set. Coordinates that point outside of the image boundaries form a corner case of the process. In this description it is assumed that these pels are detected handled by the function that computes the dissimilarity score. Now let us define $\varepsilon_{(W_\tau\to W_c)}$ as a function that computes the dissimilarity score given two regions $W_\tau(\overline{x},\overline{y})$ and $W_c(\overline{x}_c,\overline{y}_c)$ centered, respectively, at coordinates $(\overline{x},\overline{y})$ and $(\overline{x}_c,\overline{y}_c) : \overline{x}_c = \overline{x}+\delta_{x_c}; \overline{y}_c = \overline{y}+\delta_{y_c}$.

Then the $e_{(W_\tau \to W_c)}$ is computed as

$$e_{(W_\tau \to W_c)} = \varepsilon_{(W_\tau \to W_c)}(W_\tau(\overline{x},\overline{y}), W_c(\overline{x}_c,\overline{y}_c)). \tag{3.15}$$

Assuming that $\varepsilon_{(W_\tau \to W_c)}$ can be modeled as a combination, denoted as Υ, of similarity measures computed element-wise over the pels of the two regions leads to

$$e_{(W_\tau \to W_c)} = \sum_{m_x=-m}^{m} \sum_{m_y=-m}^{m} \Upsilon(\varepsilon_{(\tau \to c)}(p_\tau(\overline{x}+m_x,\overline{y}+m_y), p_c(\overline{x}+m_x+\delta_{x_c},\overline{y}+m_y+\delta_{y_c}))). \tag{3.16}$$

Considering the template matching for sets of pels and assuming that all candidate regions W_c from the candidate set C to be matched with the template region are uniquely identified by a set of possible displacements $\Delta_C^{W_\tau}$, the minimization problem (3.14) becomes

$$\overrightarrow{\Delta}_{(W_\tau \to C)} \triangleq arg \min_{\Delta=(\delta_{x_c},\delta_{y_c}) \in \Delta_C^{W_\tau}} \varepsilon_{(W_\tau \to W_c)}(W_\tau(\overline{x},\overline{y}), W_c(\overline{x}+\delta_{x_c},\overline{y}+\delta_{y_c})). \tag{3.17}$$

3.3.3.3 Sum of absolute errors as a robust matching criterion

The template matching implemented by NUGPA deploys the sum of absolute differences as the matching criterion. The value of SAD is computed between the template and the candidate patch located at coordinates identified by a vector representing relative displacement between the regions. Let us define a region as a series of $n = m_x \times m_y$ pels luminance values represented as a lexicographically ordered vector $p_i = [p_{i_1}, p_{i_2}, \ldots, p_{i_{n-1}}, p_{i_n}]^T$. Then, the dissimilarity $e_{(W_\tau \to W_c)}$ is determined by computing the sum of differences $SAD_{\tau \to c}$ between the representations of the template p_{τ_i} and candidate p_{c_i}. That is,

$$e_{(W_\tau \to W_c)} = SAD_{\tau \to c} = \sum_{i=1}^{n} |p_{c_i} - p_{\tau_i}|. \tag{3.18}$$

Block matching is considered an effective scheme for textured regions, but exhibits performance degradation when carried out for regions with high homogeneity [BAA05]. In order to prevent quality degradation, homogeneous regions are identified and handled differently. For this purpose, additional SAD between pels belonging to a candidate MB c_i and average pel value for this MB ($\overline{p}_{c_i}$ computed as in (3.19)) is estimated, in accord with (3.20). This value, represented as $SAD_{c_i}^{intra}$, forms a part of the hyper data and is passed on to the SR kernel.

$$\overline{p}_{c_i} = \frac{\sum_{j=1}^{n} p_{c_j}}{n} \tag{3.19}$$

$$SAD_{c_i}^{intra} = \sum_{j=1}^{n} |\overline{p}_{c_i} - p_{c_j}| \tag{3.20}$$

3.3.3.4 Modeling of heuristic region-based matching

As aforementioned, due to its computational cost the exhaustive (or full) search is not used in practice. Practical implementations of block-matching rely heavily on heuristic search strategies, which operate only on a subset of the possible candidate set C defined for a particular search space. In order to model heuristic search (or search strategies) let us define a function Φ that given the set of possible candidates C, the search strategy α and the reference coordinates x_r, y_r produces a set of allowable candidates C^{α} on which the heuristic search operates.

$$C^{\alpha} = \Phi(C, \alpha, (x_r, y_r)) \tag{3.21}$$

3.3.3.5 Modeling hierarchical sub-pixel matching

In order to be able to achieve sub-pixel accuracy NUGPA carries out image registration in three steps, each executing at different accuracy-level. The results of registration are cascaded between the steps. The cascade is formed in the direction of the finer steps. The results of template matching at a coarser-level are used as the initial point to guide the matching in the subsequent, finer-accuracy space. Before the matching takes place the results have to be projected onto the finer-step's grid and the vicinity of the initial guess has to be re-sampled in order to create values included in the subsequent-step's search area. Re-sampling is carried out using the already defined upsampling operator with f_i being an bilinear interpolation kernel and the projection scale $s_{K\to F}$ from coarser K to finer F step

$$s_{K\to F} = \sqrt{\frac{card(F)}{card(K)}}. \tag{3.22}$$

The projection operation is carried out by multiplying the values of the coarser-accuracy results by the projection scale. Denoting the displacement returned from template matching carried out at the coarser as $\overrightarrow{\Delta}^{K}_{(W_\tau \to C)} = (\delta^{K}_{x_{(W_\tau\to C)}}, \delta^{K}_{y_{(W_\tau\to C)}})$, the projection scale $s_{K\to F}$, then corresponding displacement at a finer-level is modeled as

$$\overrightarrow{\Delta}^{F}_{(W_\tau \to C)} = (\delta^{F}_{x_{(W_\tau\to C)}}, \delta^{F}_{y_{(W_\tau\to C)}}) = (s_{K\to F}\cdot\delta^{K}_{x_{(W_\tau\to C)}}, s_{K\to F}\cdot\delta^{K}_{y_{(W_\tau\to C)}}) = s_{K\to F}\cdot\overrightarrow{\Delta}^{K}_{(W_\tau \to C)} \tag{3.23}$$

Let's label the accuracy levels from the coarsest (step 1) to the finest (step 3), respectively as full-pixel (FP), half-pixel (HP) and quarter-pixel (QP). Then, using the new terms and the defined operations the hierarchical matching of a single region W_τ carried out by the NUGPA is modeled as a series of three subsequent minimizations, that is

$$\overrightarrow{\Delta}^{FP}_{(W_\tau \to C^{\alpha_{FP}})} \triangleq arg \min_{\Delta=(\delta_{x_c},\delta_{y_c})\in\Delta^{W_\tau}_{C^{\alpha_{FP}}}} SAD_{(W_\tau \to W_c)}(W_\tau(x,y), W_c(x_c,y_c)), \tag{3.24}$$

$$\overrightarrow{\Delta}^{HP}_{(W_\tau \to C^{\alpha_{HP}})} \triangleq arg \min_{\Delta=(\delta_{x_c},\delta_{y_c})\in\Delta^{W_\tau}_{C^{\alpha_{HP}}}} SAD_{(W_\tau \to W_c)}(W_\tau(x,y), W_c(\tilde{x}_c,\tilde{y}_c)), \tag{3.25}$$

$$\overrightarrow{\Delta}^{QP}_{(W_\tau \to C^{\alpha_{QP}})} \triangleq arg \min_{\Delta=(\delta_{x_c},\delta_{y_c})\in\Delta^{W_\tau}_{C^{\alpha_{QP}}}} SAD_{(W_\tau \to W_c)}(W_\tau(x,y), W_c(\overline{x}_c,\overline{y}_c)), \tag{3.26}$$

where, the candidate sets used in the sub-pixel matching are created as

$$C^{\alpha_{HP}} = \Phi(C, \alpha_{HP}, (s_{FP\to HP}\cdot x + \delta^{HP}_{x_{(W_\tau \to C^{\alpha_{FP}})}}, s_{FP\to HP}\cdot y + \delta^{HP}_{y_{(W_\tau \to C^{\alpha_{FP}})}})), \tag{3.27}$$

$$C^{\alpha_{QP}} = \Phi(C, \alpha_{QP}, (s_{HP\to QP}\cdot x + \delta^{QP}_{x_{(W_\tau \to C^{\alpha_{HP}})}}, s_{HP\to QP}\cdot y + \delta^{QP}_{y_{(W_\tau \to C^{\alpha_{HP}})}})). \tag{3.28}$$

Registration of the whole image is carried out by repeating the above process for all regions of the source image. In case of performing the registration for multiple sensed images, the whole registration process is repeated for each combination of the source-sensed images. In the case of a set κ containing k images of which each is divided into w template regions containing up to $m \times m$ pels, then, using the defined terms, the image registration process for image $t : t \in \kappa$ is executed $M \times (k-1)$ times for each template region of each frame $i \in \kappa \wedge i \neq t$. The result of the image registration process (the hyperdata) of the image t over a set of $k-1$ sensed images (all images except the image t) are $w \times (k-1)$ displacements in $\overline{X}^2$ space and (corresponding) computed sums of absolute differences, forming a set $\{\overrightarrow{\Delta}^{QP}_{(W_\tau \to C_i)}, SAD_{(W_\tau \to C_i)}, SAD^{intra}_{c_i}\}$ for $\tau : 1 \leq \tau \leq w \wedge \tau \in N$; $i : i \in \kappa \wedge i \neq t$ and quarter-pixel accuracy of the $\overline{X}^2$ space.

3.3.4 Mathematical model of the restoration-interpolation stage

In case of the non-uniform grid projection algorithm the second stage of the SR process is the restoration-interpolation. As the name suggests, at the highest level of abstraction, the restoration-interpolation stage can be seen as composed of two stages: (i) the reconstruction

or fusion of the information extracted from the input images, and (ii) interpolation used to compute the data that are still missing. The reconstruction process carries out the fusion of the image to be super-resolved with the non-redundant data found in the HR projections of the remaining κ images. The non-redundant data are extracted from the locations identified by the hyperdata. In that sense, this part of the reconstruction process can be seen as an inverse of the image registration step. As it was the case with image restoration, the first step of the reconstruction process is the up-sampling of the input low resolution frames $g(x,y,i) : i \in \kappa$ to the resolution of image registration. The resulting HR representations $g(\overline{x},\overline{y},i) : i \in \kappa$ are created using the up-sampling operator (3.9) with the interpolation kernel f_i defined to return a constant value of $\emptyset$. This is modeled as

$$g(\overline{x},\overline{y},i) = upsample^{\sqrt{\frac{card(\overline{X}^2)}{card(X^2)}}}(g(x,y,i)). \tag{3.29}$$

The super-resolution kernel $k(x,y)$ processing is carried out by applying the fusion kernel FK on the frames from the set κ, carrying out interpolation process INT on the output of the fusion and downsampling the obtained representation. Let us defined the hyperdata obtained from the image registration of an image t from a set κ containing k images, over all of the remaining images of this set as Ω_t^{κ}. Then,

$$k(x,y) = downsample^{\sqrt{\frac{precision_{ir}}{scale_{sr}}}}(INT(FK(g(\overline{x},\overline{y},\kappa),\Omega_t^{\kappa}),int(\overline{x},\overline{y})), \tag{3.30}$$

where $int(\overline{x},\overline{y})$ represents the interpolation kernel used in the interpolation process.

Let us define the fusion operator $\oplus$ that carries out the operation of projecting pels of one region onto another. Using this operator the fusion of an image over a set of images is modeled as (3.31) where $\hat{s}(\overline{x},\overline{y},t)$ represents the outcome of the fusion process for frame t.

$$\hat{s}(\overline{x},\overline{y},t) = \sum_{i=1,i\neq t}^{k} g(\overline{x},\overline{y},t) \oplus g(\overline{x},\overline{y},i) \tag{3.31}$$

Then the fusion process of the image captured at the time t with another image from κ can be modeled as an act of performing a series of fusions of regions of these images. Let $\oplus$ operate in a way that allows it to carry out the projection of the region that is its right-hand argument onto a grid defined for region which is its left hand argument. Assuming, cardinality of τ to be w and that the fusions of each region result in an immediate update of

the reference region, this processing carried out by the FK can be modeled as

$$\hat{s}(\overline{x},\overline{y},t) = FK(g(\overline{x},\overline{y},\kappa),\Omega_t^{\kappa}) = \sum_{i=1,i\neq t}^{k}\sum_{r=1}^{w} W_r(\overline{x}_r,\overline{y}_r) \oplus W_{e_{r\to i}^{max}}(\overline{x}_r+\delta_{x_{r\to i}},\overline{y}_r+\delta_{y_{r\to i}}). \quad (3.32)$$

It is a common case, that after all of the frames have been fused, the projection grid contains many coordinates that have not been assigned a value and maintain the initial value $\emptyset$. The creation of values for these coordinates is carried out using the $int(\overline{x},\overline{y})$ function defined as in (3.33), where $f_i(\overline{x},\overline{y})$ encapsulates the non-uniform interpolation kernel. The non-uniform interpolation computes the missing values using mean computed over its neighbor. Only the non-missing values contribute in the process. The current version of the algorithm uses location-independent weights which are equal for all participating pels.

$$int(\overline{x},\overline{y}) = \begin{cases} f_i(\overline{x},\overline{y}) & \text{if } \hat{s}(\overline{x},\overline{y},t) = \emptyset \\ \hat{s}(\overline{x},\overline{y},t) & \text{otherwise} \end{cases} \quad (3.33)$$

The restoration and the interpolation transformations are carried out in the $(\overline{x},\overline{y})$ coordinates space. Should $precision_{ir} \neq scale_{sr}$ then the fused image after interpolation has to undergo a transition from $\overline{X}^2$ to $\hat{X}^2$ space. This step is carried out using the down-sampling operation (3.7) with scale $d = \frac{precision_{ir}}{scale_{sr}}$ and is modeled as

$$S(\hat{x},\hat{y},t) = downsample^d(INT(\hat{s}(\overline{x},\overline{y},t), int(\overline{x},\overline{y}),)), \quad (3.34)$$

where $S(\hat{x},\hat{y},t)$ represents the super-resolved frame.

3.4 Quantitative evaluation of the non-uniform grid projection algorithm

This section will focus on evaluating the performance, in terms of the observed super-resolved image quality, of applying NUGPA on luminance component of an image. First, the topic of image quality assessment will be introduced and details on the proximity measures used in this study will be provided. Then, the test set-up and the results of the study on the quality of the super-resolved image as a function of algorithm parameters as well as a brief comparison with most popular solutions of the problem of image resolution enhancement. The subject of the study will be the reference software that implements the flow presented in Section 3.2. This software has formed the starting point in our struggle for

provision of real-time SRIR in hardware.

3.4.1 Image quality assessment

The ultimate goal of super-resolution is to increase image quality. Quality assessment for a digital video/image, especially compressed ones, is not as straightforward as in case of analog video signals and forms a dynamic research domain on its own. Analog video signal could be evaluated with some measurements like (non-)linear distortions, noise, etc., whose physical meaning is clear and easily understandable. Assessment of digital signal, due to quantization and compression, requires a profound understanding of the human psycho-visual system if the provided mathematical models are to offer meaningful correlation with the results of subjective quality assessment of human observers. Hence, over the last 30 years several evaluation criteria have been proposed in order to evaluate the quality of digital images and compare the improvements offered by image enhancement algorithms [SSBC10]. These methods —the *image quality assessment* (IQA) methods— can be classified into two categories: the *objective* and *subjective* evaluation methods. The former methods are aimed at quantifying the perceived quality by designing (and using) mathematical models that are able to predict the quality of an image accurately and automatically (without surveillance of a human observer). An ideal objective method should be able to mimic the quality predictions of an average human observer and be accurately reproducible [MEMS14]. The main drawback of the real-life objective evaluation is that it still bears limited relationship to the amount of visible distortion perceived by humans. The subjective evaluation methods are carried out by humans who quantitatively evaluate the image. These methods are seen as the ultimate test of the image quality as they take into account intangible attributes (that are most likely) not included in the mathematical models used by the objective methods. In practice, however, subjective perceptual evaluation is usually too inconvenient, difficult to set-up, time-consuming and expensive to be used. Even though a number of standard test procedures have been developed [ITU02, ITU98a, ITU94a, ITU08, ITU98b, ITU98c, ITU94b] subjective perceptual evaluation is still not as reliable as one might hope. Reliable subjective perceptual measurements are still the focus of intensive on-going research. Thus, in the case when reference image is available, objective evaluation is used to benchmark image processing algorithms and choose the algorithm that provides the higher quality images.

The objective IQA methods can be further classified based on the availability of a reference image, which is considered to be distortion-free and have perfect quality, into three

groups: (i) the *full-reference* image quality assessment where the undistorted, perfect quality reference image is fully available, (ii) the *reduced-reference* image quality assessment where only some features of the reference image are employed in evaluation, and (iii) the *no-reference* image quality assessment in which there is no access to reference image [MEMS14]. When the reference image is available, as is the case considered in this study, quantification the degree of similarity between the produced outcome and the desirable outcome is possible. In other words, it is desired to measure how closely does the super-resolved image approximate the reference if the latter is available. This measurement is done using the so called proximity measures that quantify the *'distance'* between the two images in a perceptually meaningful way [TK08]. In order to assure ease of comparison with most of the publications on image enhancement algorithms, this study will focus on presenting proximity measures obtained using the most widely used objective full reference IQA methods, namely the PSNR and *Multi-Scale Structural SIMilarity* (MSSIM) [WBSS04, WBS05, WSB03a].

3.4.1.1 Peak signal-to-noise ratio

PSNR is by far the most widely used proximity measure. This is mostly due to its simplicity, clear physical meaning, ease of efficient implementation, sensitivity to small changes in picture degradation and ability to be accurately reproducible. For measuring PSNR, first the mean square error of the obtained image/video frame vs the reference (both composed of N pels) is computed by averaging the squared intensity differences between the distorted ($\hat{pel}_{SR}(i)$) and the reference ($pel_{ref}(i)$) image pixels:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{pel}_{SR}(i) - pel_{ref}(i))^2. \tag{3.35}$$

The MSE is the norm of the arithmetic difference between the reference and the test signals. It is an attractive measure for the (loss of) image quality due to its simplicity and mathematical convenience. PSNR is then calculated as the logarithmic ratio between the peak signal (maximal value of the signal; determined by number of bits used to represent video signal) and the square root of the MSE [Boc09, Appendix C]:

$$PSNR = 10 * \log_{10}\frac{(max_{value})^2}{MSE}. \tag{3.36}$$

PSNR is usually expressed in decibel.

The simplicity and ease of implementation of PSNR are significant advantages making

it the most wide-spread objective quality measure. While it is true that PSNR, under certain conditions [Boc09], can provide relevant information about the image and its similarity with the reference, PSNR usefulness is limited as long as the perceptual video quality is considered as it has been proven that it does not correlate well with perceptual video quality [EF95, PS00, WB02, WSB03b, WBSS04, OLL+04]. Also, by using MSE/PSNR it is implicitly assumed that errors at different locations are statistically independent which is not guaranteed without extra processing. A last thing to keep in mind is that PSNR values depend on the input image size, being higher for bigger images.

3.4.1.2 Structural similarity index

The MSE/PSNR measures are based on the assumption that an image signal whose quality is being evaluated can be considered as a sum of an undistorted reference signal and an error signal. Even though this assumption is true, the derived concept that objectively quantified strength of the error signal corresponds to the visibility of the errors and the perceived quality is not always sound. Two distorted images with the same PSNR/MSE may have very different types of errors, some of which are much more visible than others [WBSS04].

In the last three decades, a great deal of effort that has been dedicated to the development of IQAs methods that would take advantage of known characteristics of the human visual system and provide a higher correlation with quality perceived by human observers have resulted in introduction of several *new-generation* quality assessment methods [WBSS04, HMM99, LWBK02, WLB02]. The time when quality assessment models were based only on a digital model of eye have ended and nowadays this model is being extended to include a digital model of primary visual cortex. These methods, referenced usually as perceptual image quality assessment methods, attempt to weight different aspects of the error signal in reference to their visibility determined by taking into account the psychophysical characteristics of the *human visual system* (HVS). The error sensitivity approach estimates perceived errors to quantify image degradations, while the new philosophy considers image degradations as perceived changes in structural information variation and tries to capture the loss in parts that the HVS hypothetically extracts for cognitive understanding.

One should bear in mind that most studies still provide PSNR as the baseline that is accompanied by a more accurate measure. As for the time of writing, there has been no agreement on the optimal measurement that should be used to complement the PSNR results, but *Structural SIMilarity* (SSIM) [WBS05] seems to be the algorithm that has gained the most popularity and acceptance and is the IQA used in most of the recent academic

publications. Natural image signals are highly structured: their pixels exhibit strong dependencies, especially when they are spatially proximate, and these dependencies carry important information about the structure of the objects in the visual scene. Structural similarity considers the HVS as highly adapted for extracting structural information from the scene. Thus, rather than looking at the individual differences at pixel levels it assumes the structural similarity as a better measure of perceived quality — instead of measuring the error (like in case of PSNR) SSIM focuses on measuring the structural changes. The structural similarity index was first proposed in [WBSS04]. Over the years many variations that based on the structural similarity paradigm have been proposed. The most used ones are the single-scale SSIM [WBS05], multi-scale SSIM [WSB03a], SSIM for ranged pictures (R-SIM) [MB08] and *Speed* SSIM [WL07]. Results of various surveys [MEMS14, SSBC10] recommend the use of MSSIM in full-reference IQA.

3.4.1.2.1 Single-scale structural similarity. Structural similarity is based on the hypothesis: (i) that distortions in an image that come from variations in lighting, such as contrast or brightness changes, are non-structural distortions, and that these should be treated differently from structural ones, (ii) and that one could capture image quality with three aspects of information loss that are complementary to each other: (a) correlation distortion, (b) contrast distortion, and (c) luminance distortion. In other words, SSIM compares local patterns of pel intensities that have been normalized for luminance and contrast. This is done in order to capture the loss of structure in the signal, structure that the HVS hypothetically extracts for cognitive understanding. In particular, both the SSIM Index and the HVS are highly sensitive to degradation in the spatial structure of image luminances.

The basic (single-scale) SSIM algorithm requires that the two images being compared be properly aligned and scaled so they can be compared pel by pel. The computations are performed in a sliding $P \times Q$ (typically 11×11) Gaussian weighted window. Three similarity functions are computed on the windowed image data: luminance similarity $l(x,y)$, contrast similarity $c(x,y)$, and structural similarity $s(x,y)$, which for two images X and Y are calculated as follows.

$$l(x,y) = \frac{2\mu_X(x,y)\mu_Y(x,y) + C_1}{\mu_X^2(x,y) + \mu_Y^2(x,y) + C_1}, \tag{3.37}$$

$$c(x,y) = \frac{2\sigma_X(x,y)\sigma_Y(x,y) + C_2}{\sigma_X^2(x,y) + \sigma_Y^2(x,y) + C_2}, \tag{3.38}$$

$$s(x,y) = \frac{\sigma_{XY}(x,y) + C_3}{\sigma_X(x,y) + \sigma_Y(x,y) + C_3}, \tag{3.39}$$

where

$$\mu_x(x,y) = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) X(x+p, y+q), \tag{3.40}$$

$$\mu_y(x,y) = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) Y(x+p, y+q), \tag{3.41}$$

$$\sigma^2(x,y) = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) [X(x+p, y+q) - \mu_X(x,y)]^2, \tag{3.42}$$

and

$$\sigma_{XY}(x,y) = \sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) \cdot [X(x+p, y+q) - \mu_X(x,y)] \cdot [Y(x+p, y+q) - \mu_Y(x,y)], \tag{3.43}$$

where $w(p,q)$ is a Gaussian weighing function such that $\sum_{p=-P}^{P} \sum_{q=-Q}^{Q} w(p,q) = 1$, and C_1, C_2 and C_3 are small constants that provide stability when the denominator approaches zero. Typically:

$$C_1 = (K_1 L)^2, C_2 = (K_2 L)^2, C_3 = C_2/2, \tag{3.44}$$

where L is the dynamic range of the image (255 for 8-bit grayscale images) and $K_1 \ll 1$ and $K_2 \ll 1$ are small scalar constants. The three similarity functions are then combined into the general form of the SSIM index:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma}, \tag{3.45}$$

where α, β and γ are parameters used to adjust the relative importance of the three components. Assuming the $\alpha = \beta = \gamma = 1$ and $C_3 = C_2 \div 2$ (typical case) and using (3.37), (3.38) and (3.39) in (3.45) the final form of the index becomes:

$$SSIM(x,y) = \frac{(2\mu_X \mu_Y + C_1)(2\sigma_{XY} + C_2)}{(\mu_X^2 + \mu_Y^2 + C_1)(\sigma_X^2 + \sigma_Y^2 + C_2)}. \tag{3.46}$$

3.4.1.2.2 Multi-scale structural similarity. Single-scale version of the SSIM does not take into account the fact that the capability to perceive details by human observers differs with the viewing conditions like sampling density of the image, viewing distance, and the observer's HVS perceptual capabilities [SBPL11]. This phenomena is well known and has

been exploited intensively in *Computer-Generated Imagery* (CGI) applications where the level of details of a texture or a model varies with the viewing conditions — depending on the distance from the observer and display density different sets of textures/models are being displayed [Dur96, DH97]. Multi-scale SSIM tackles the issues of incorporating the influence of these conditions on image quality by creating multiple scaled-down versions of the compared images and carrying the assessment not only for the original images but for all created versions. The scaled-down versions used in the assessment are created by iteratively applying a low-pass filter and downsampling the filtered image by a factor of 2 as illustrated in Fig. 3.7. This effectively removes the limitation of the evaluation being appropriate for only one specific viewing condition.

Let us label the original image as *scale 1*, and the highest *scale M* as corresponding to the most downsampled image (obtained after $M-1$ iterations). At the i-th scale, the contrast comparison and the structure comparison are calculated and denoted as $c_i(x,y)$ and $s_i(x,y)$, respectively. The luminance comparison is computed only for the highest scale (scale M) and is denoted as $l_M(x,y)$. The overall evaluation is obtained by combining the measurements at different scales using

$$MSSIM(x,y) = [l_M(x,y)]^{\alpha_M} \cdot \prod_{i=1}^{M} [c_i(x,y)]^{\beta_i} \cdot [s_i(x,y)]^{\gamma_i}, \tag{3.47}$$

where, as in (3.45), α_M, β_i, γ_i are used to adjust the relative importance of different components. The values of these cross-scale parameters need to be evaluated empirically during the calibration process. The number of iterations required for the MSSIM to give relevant values is reasonably small. The authors of this method use five levels in their work [WSB03a]. For bigger images (in terms of number of pels) it may be justified to use more levels.

3.4.2 Output quality as a function of SR parameters

The goal of the study presented in this section has been to provide a quantitative evaluation of the reference software implementation of NUGPA (hereafter *SRiuma* or SR_{iuma}) and to determine the impact that the SR algorithm parameters, tabulated and briefly described in Table 3.1, have on the quality of the obtained super-resolved images. For the purpose of algorithm performance quantification a set of 120 combinations of the algorithm parameters values—MB_{width} (4, 8 and 16), SAR (2, 4, 8 and 16), number of RFs (2, 4, 8, 12 and 16), and the scale (2 or 4)—has been defined to be used in simulations.

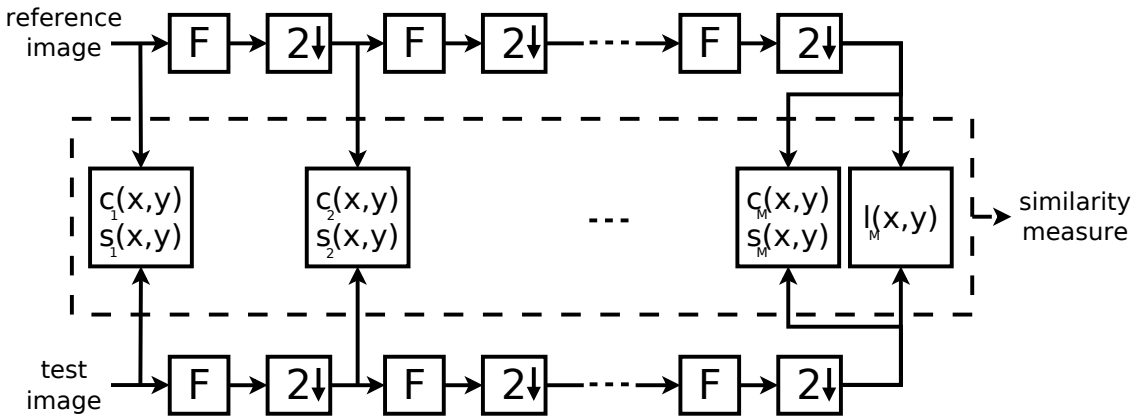


Fig. 3.7 Multi-scale structural similarity measurement system. F: low-pass filter; 2↓ downsampling by 2. Modified version of [WSB03a].

3.4.2.1 Experimental set-up

The reference image (and thus the output) format has been chosen to be the YUV 4:2:0p 8bit CIF (288x352 pels) sequence. This resulted in the LR input being a QCIF (144x176 pels) or a 72x88 pels frame format for the tested values of the super-resolution zoom (*scale*) of 2 and 4, respectively. Even though, due to high computational complexity and the fact of proven better correlation between the objective proximity measures (MSE, PSNR) with the results of subjective evaluation, it is common to use images of lower resolutions (32x32, 64x64, 128x128) [ESHEK12, chapter 2], in this study higher resolution inputs have been used in order to facilitate better quality of details inspection and more accurate MSSIM index assessment.

Results of the study presented in [WBS05] suggest that the use of the color components does not significantly change the performance of the used quality measurement model. This was expected as it is a well-known fact that most of the energy is express by the luminance component [IP91]. These carried out experiments considered only the quality of luma component, which is the only one for which super-resolution is carried out. The simulations were carried out in accord with the flow presented in Fig. 3.8:

1. First the YUV 4:2:0p 8bit low resolution (QCIF or 72×88 pels) sequences were obtained from the CIF reference sequence. In order to do so, the CIF sequence was loaded, and decimated, forming a sequence that was then stored.

2. The LR sequences were used as input for tested algorithms (the choice of the benchmark is the focus of the next subsection), resulting in CIF YUV 4:2:0p 8bit output.

TABLE 3.1 *Algorithm parameters and their values used in the study on image quality.*

Parameter	Value	Description
MB_{width}	4, 8, 16	Number of pels that a MB contains in each dimension. Determines the memory used for storage of a MB and the number of MBs in a frame.
FR_{cols}	176	Number of pel columns in a LR frame.
FR_{rows}	144	Number of pel rows in a LR frame.
$precision_{ir}$	4	The ratio between the 1D dimensions of the finest accuracy grid used in image registration and the full-pixel grid used for input representation.
$scale$	2, 4	The ratio between the 1D dimensions of the outcome accuracy grid and the full pixel grid used for input representation;
SAR	2, 4, 8, 16	The absolute distance (in full-pixel coordinates) used to determine if a pel is contained within the limits of the SA; With MB_{width} determines the SA_{size}.
int_{window}	2	The absolute distance (in finest IR accuracy coordinates) used for the determination of the neighboring MB's pels that can contribute to the interpolation process.

3. The output luma pels ($\hat{pel}_{SR}$) and the original CIF sequence pels pel_{ref} were post-processed by the IQA that quantified the grade of similarity between these two produced representations by calculating, among others, the PSNR and SSIM indices. In calculations both images are consider a lexicographically ordered sets containing N pels (total number of pels in the image).

   (a) MSE has been computed using (3.35).

   (b) PSNR was then computed based on the MSE value using (3.36) which for an 8 bit pel representation becomes

$$PSNR = 10 * \log_{10} \left(\frac{255^2}{\frac{1}{N} \sum_{i=1}^{N} (\hat{pel}_{SR}(i) - pel_{ref}(i))^2} \right). \tag{3.48}$$

   (c) Single-scale structural similarity index has been calculated as follows:

      i. First, the C constants were computed using (3.44) with $K_1 = 0.01$, $K_1 =$

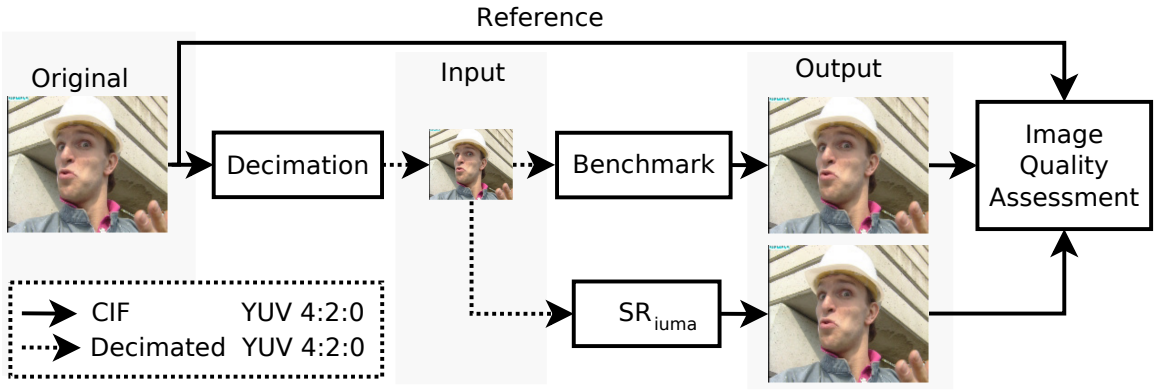


Fig. 3.8 Data flow of the experiments used to assess the output quality of SRIR.

0.03, and dynamic range (L) of 255 (8-bit grayscale).

ii. Then, the mean intensity μ_X were computed. The luminance comparison was then a function of μ_X and μ_Y.

$$\mu_X = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.49}$$

iii. Following, the mean intensity was removed from the signal. In discrete form, the resulting signal was the $x - \mu_X$. Standard deviation (the square root of variance) was used as an estimate of the signal contrast. An unbiased estimate in discrete form was given by

$$\sigma_X = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_X)^2\right)^{\frac{1}{2}}. \tag{3.50}$$

The contrast comparison was then a function of σ_X and σ_Y.

iv. Third, the signal was normalized (divided) by its own standard deviation, so that the two signals being compared have unit standard deviation. The structure comparison $s(x,y)$ was conducted on these normalized signals $(x-\mu_X)\sigma_X$ and $(x-\mu_Y)\sigma_Y$.

v. Finally, the three components were combined as in (3.46).

(d) Multi-scale similarity index has been computed by using

i. First, the C constants were computed using (3.44) with $K_1 = 0.01$, $K_1 = 0.03$, and dynamic range (L) of 255 (8-bit grayscale).

ii. Low pass filter, implemented as convolution (Matlab `filter2` function) of

a 11-by-11 window with Gaussian distribution (standard deviation = 1.5) with the image being filtered, was applied and its results were downsampled by 2.

iii. Contrast and structure similarities were computed as in the aforementioned single-scale SSIM flow.

iv. The iteration counter was incremented. If the final iteration has been reached the luminance index $l_M(x,y)$ was computed. Otherwise, steps 3(d)ii–3(d)iii were repeated.

v. The computed SSIM indices were combined using parameters values from [WSB03a] ($M = 5$, $\beta_1 = \gamma_1 = 0.0448$, $\beta_2 = \gamma_2 = 0.2856$, $\beta_3 = \gamma_3 = 0.3001$, $\beta_4 = \gamma_4 = 0.2363$ and , $\alpha_5 = \beta_5 = \gamma_5 = 0.1333$) and (3.47) which became

$$MSSIM(x,y) = [l_5(x,y)]^{0.1333} \cdot \prod_{i=1}^{5} [c_i(x,y)]^{\beta_i} \cdot [s_i(x,y)]^{\gamma_i}. \tag{3.51}$$

3.4.2.2 Determination of the image quality benchmark

This initial aim of this study was to compare the results of our algorithm with two competitive solutions: one interpolation and one SRIR implementation. Before the assessment of the proposed SRIR implementation was to be carried out the image quality benchmark needed to be chosen. In order to do so the available competitive solutions presented in the state of the art Section 2.4 have been looked at. A natural competitor and the one that had been successfully implemented in hardware [ABCC08, BB08] would have been the IBP algorithms. Unfortunately, to the best knowledge of the authors, software implementations of these implementations were either not available [ABCC08, BB08] or the available software [LCA07] was found to be unable to process real-life video sequences without severely corrupting the output.

In order to determine the interpolation algorithm to be used as the quality benchmark simulations for three test sequences using six interpolation algorithms have been carried out as presented in Fig. 3.9. The sequence used in the experiments are briefly described in Table 3.2. These sequence form part of the *Xiph.org Video Test Media [derf's collection]* available online [Xip15].

The produced images allowed computing the PSNR and MSSIM indices. The computed PSNR and MSSIM indices are presented in Fig. 3.10 and Fig. 3.11, respectively. The results for the nearest neighbour, bilinear, bicubic, lanczos2 and lanczos3 interpolation [BB09, Key81, Get11, MN13] have been obtained by means of invocation of the

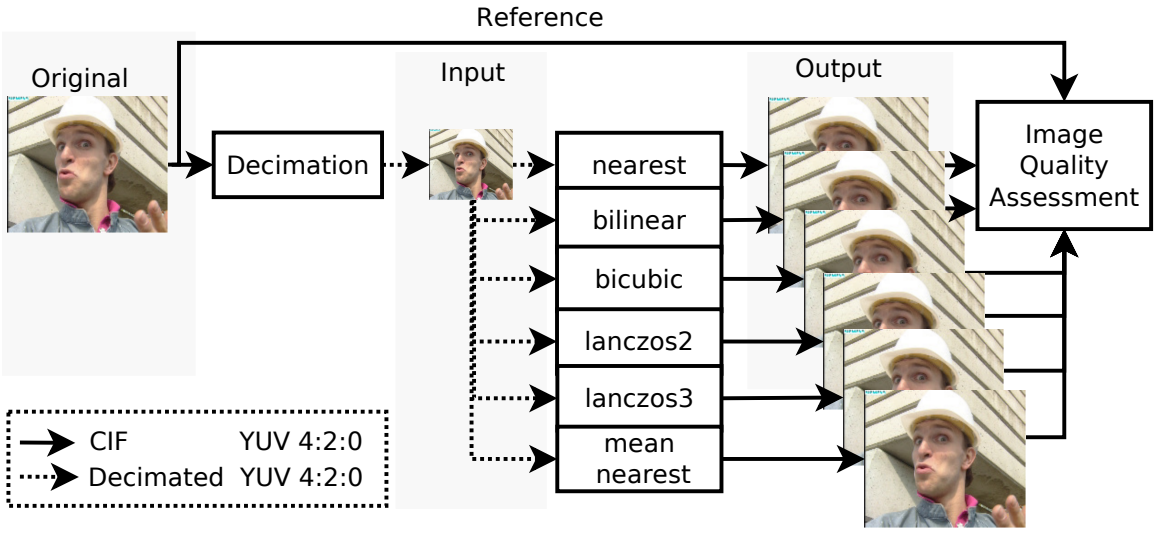


Fig. 3.9 Data flow of the experiments used to choose the benchmark algorithm.

`imresize()` function from the Image Processing Toolbox of the MatLab programming environment [Mat96]. The non-uniform mean nearest neighbours (neighbourhood of 5x5 pels) algorithm have been implemented in software using ANSI C. From the resulting MSSIM values it can be seen that the mean nearest neighbors implementation has consistently outperformed the other five implementations reporting the highest index value in all test cases. In terms of the reported PSNR, the use of the mean nearest neighbors interpolation leads to superior results in 5 out of 6 performed tests, being outperformed only in one case by the bilinear interpolation. Based on these results the decision to use the non-uniform mean nearest neighbors as the benchmark for image quality assessment of our algorithm has been made.

3.4.2.3 PSNR and MSSIM as a function of algorithm parameters

Once the benchmark algorithm has been chosen, the flow presented in the previous section has been used to carry out simulations for all of the aforementioned 120 combinations. Test sequences used in experiments are the same that have been used in the benchmark determination tests. The experiments were conducted on a Sun Ultra 24 Workstation, hosting one Intel Core 2 Quad cpu Q9300@2.5 GHz, 3.25GB of RAM memory, and using Windows XP Professional operating system with Service Pack 3. The produced output has been post-processed using functions from the Matlab *MeTriX MuX Visual Quality Assessment Package* library [Gau07]. Let us remind that the version of the SR software used in this study is not robust, thus it is not capable of eliminating outliers and it does not include mechanisms to

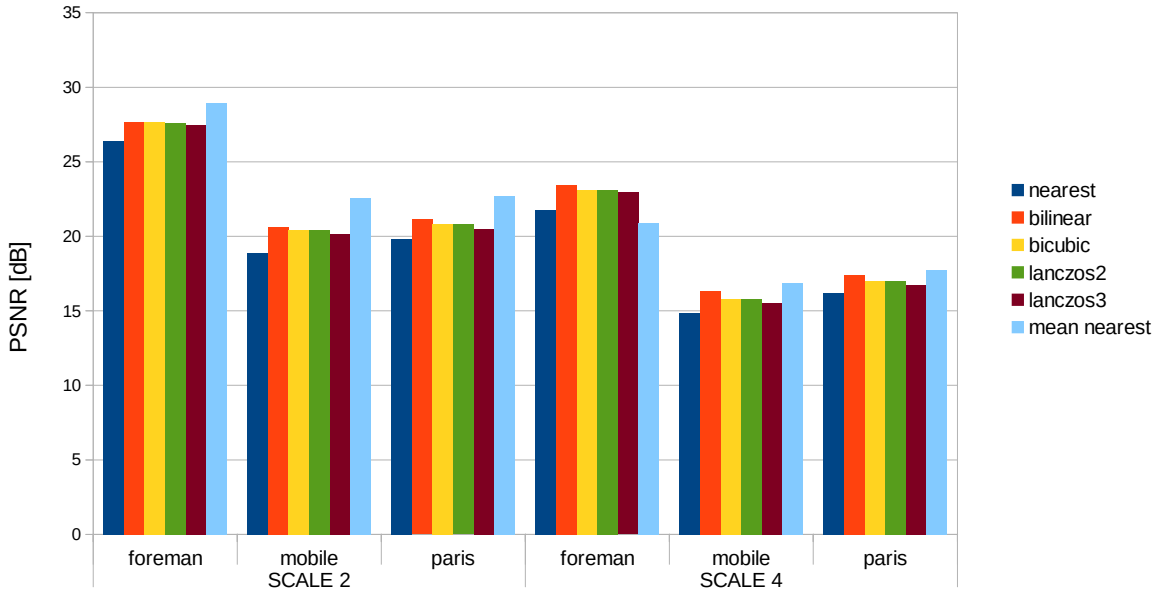


Fig. 3.10 Average PSNR (300 initial frames) observed for the interpolated foreman, mobile and paris sequences.

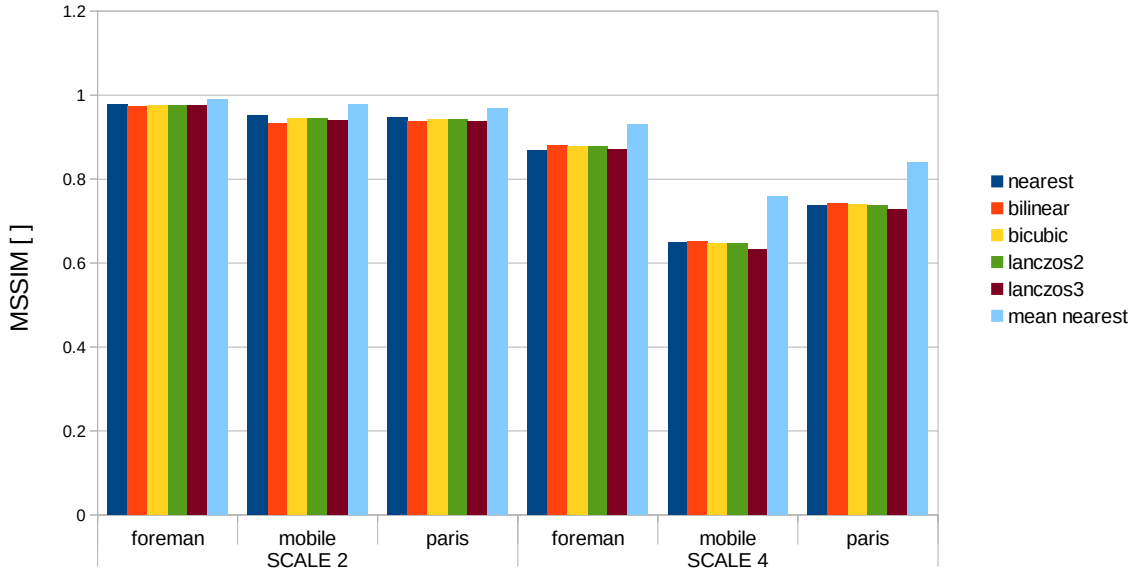
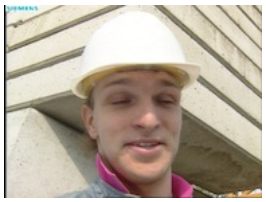


Fig. 3.11 Average output MSSIM (300 initial frames) observed the interpolated foreman, mobile and paris sequence.

TABLE 3.2 *Test sequences used in the experiment.*

Sequence	Visualization	Description
Foreman		Presents a worker performing rapid random head movements, while talking to the camera. The sequence then changes to present the working site. This sequence presents global and rapid local movement, along with context change.
Mobile		An electrical train moving on the table, a spinning planetary system and a calendar hanging on the wall. The train is moving horizontally, whereas the calendar is pulled to present vertical movement. The ball rotates representing a mixture of both types of movement. Global movement is present as the camera is moving to the left. This sequence contains rich and fine grain textures.
Paris		Presents a couple sitting by a table and talking. While talking, the woman is juggling a ball, and the men is playing with a pen. The sequence presents rapid local movements. Global movement is absent.

correctly handle context changes, nor capable of correctly handling non-translational movement (panning, rotations, etc.). Also, noise attenuation by averaging is significantly limited as no proximity-based fusion weights computation is carried out (weights are set equal to 1).

The focus of this section is the presentation of the PSNR and MSSIM indices obtained for the foreman, mobile and paris sequences. These three sequences have been found to be representative for all of the carried out test. The PSNR values computed for the foreman, mobile and paris sequences for SR scales of 2 and 4 are presented in Fig. 3.12 and Fig. 3.14, respectively. The corresponding MSSIM values are shown in Fig. 3.13 and Fig. 3.15, respectively. From these figures one can see that there is a strong correlation between the trends observed for both scales. In other words, a change of the SRIR parameters has a similar effect on the way the observed outcome quality is changed for a given sequence for both scale values. Even though, there is a strong correlation between the observed outcome quality for results obtained for different scale values, that can be clearly seen when comparing both of the aforementioned figures side by side, the way the algorithm parameters impact the outcome quality is not as straightforward. The observed correlations have been

found to be heavily dependent on the used test sequence. In order to analyze the way that each of the parameters impact the output quality the change in the outcome quality indices observed for the combinations for which other-than-the-analyzed parameters are fixed has been investigated.

In the case of the foreman sequence, the highest PSNR and MSSIM values have been observed for the macro-block comprising 8x8 pels, SAR of 16 pels, and 4 reference frames (2 in each temporal direction; (SFW comprising 5 frames)) for both tested scales values. For this sequence the change of the MB size parameter manifests itself as a ($< 0.5dB$) translation towards lower PSNR values. The *magnitude* of the observed translation is about two times bigger in the case of the switch to MB size of 4 than for the case of switch to the value of 16. In the case of the foreman sequence, increase in search radius has always led to a higher PSNR, with the noticed gain being higher for combinations with higher number of available reference frames. The observed impact of the change in the number of used reference frames on the output quality has been the greatest from all of the tested parameters (for a fixed scale). For foreman, better results have been reported for smaller number of available reference frames (2 and 4) which outperform interpolation in most cases. For the case with 8 reference frames SRIR, managed to surpass interpolation in some cases (3) when provided with large search areas. From the carried out tests none of the combinations with 16 reference frames managed to match the quality of the benchmark interpolation. The gap between the computed values for different RFs numbers is reduced with the increase of the search area radius. The noticed values converged the fastest for the medium MB size (8x8) and the slowest for the largest tested MBs (16x16). The described type of correlation between the output's quality observed for scale value of 2 holds true also for scale value of 4. The difference between the two diagrams are the overall lower PSNR and MSSIM values and the fact that, in case of PSNR, the use of SRIR resulted in significantly better (> 2.0 dB) quality than the benchmark interpolation for all 60 tested parameters combinations. As far as MSSIM is considered, SRIR managed to outperform the interpolation only in some cases ($\approx 20\%$) with medium/large MBs and relatively large search areas.

In the case of the mobile sequence, the highest PSNR and MSSIM values have been observed for the macro-block comprising 16x16 pels, SAR of 2 pels, and 8 (scale = 2) or 16 reference frames (scale = 4). The outcome quality has been observed to be better for larger macroblocks with the exception of PSNR observed for the cases with 2 reference frames and scale of 4. In contrast with the trends observed for foreman, for the mobile test sequence use of more RFs resulted, in most cases, in higher quality index—showing that the algorithm was able of taking advantage of additional information contained in these frames.

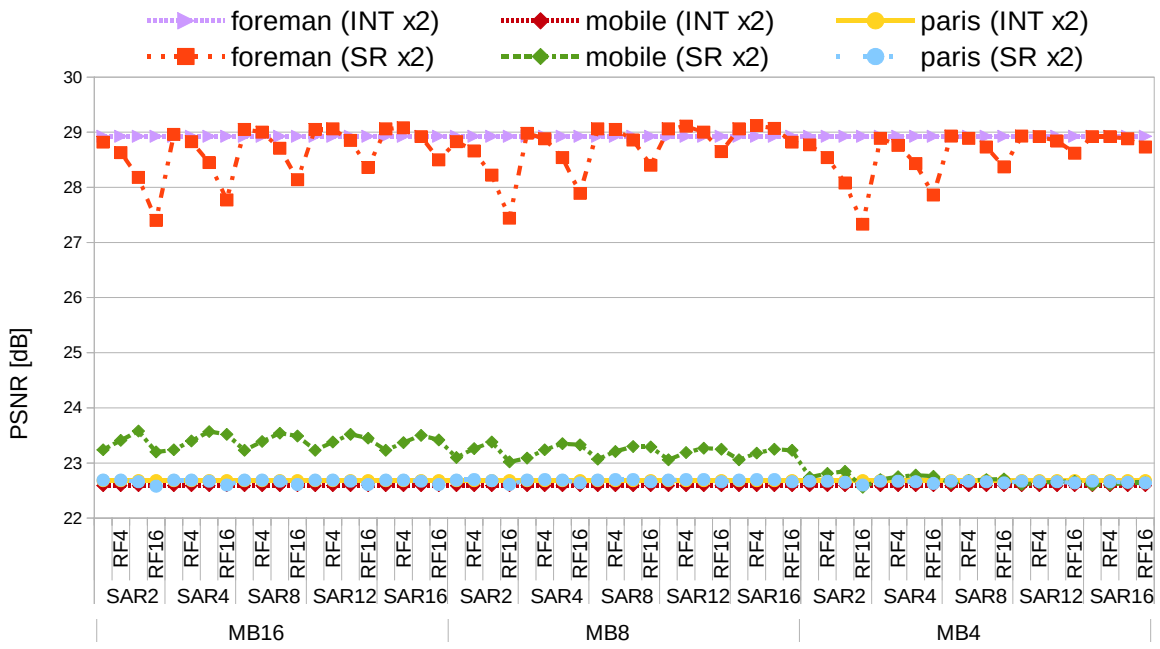


Fig. 3.12 Average PSNR observed for the super-resolved and interpolated foreman, mobile and paris sequences (initial 300 frames; scale=2).

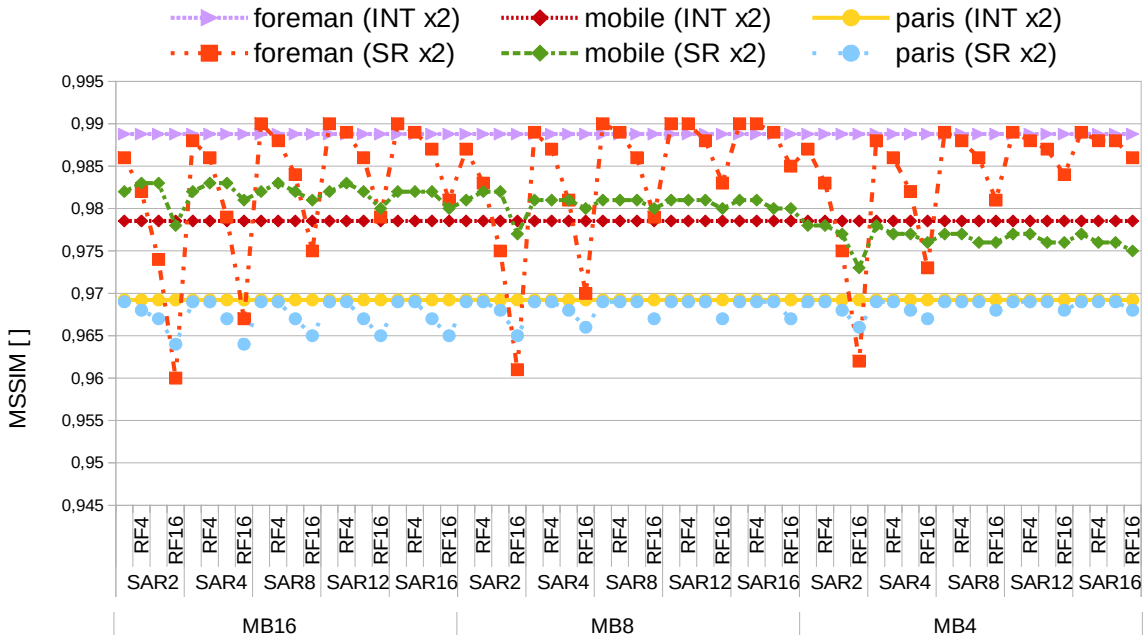


Fig. 3.13 Average MSSIM observed for the super-resolved and interpolated foreman, mobile and paris sequences (initial 300 frames; scale=2).

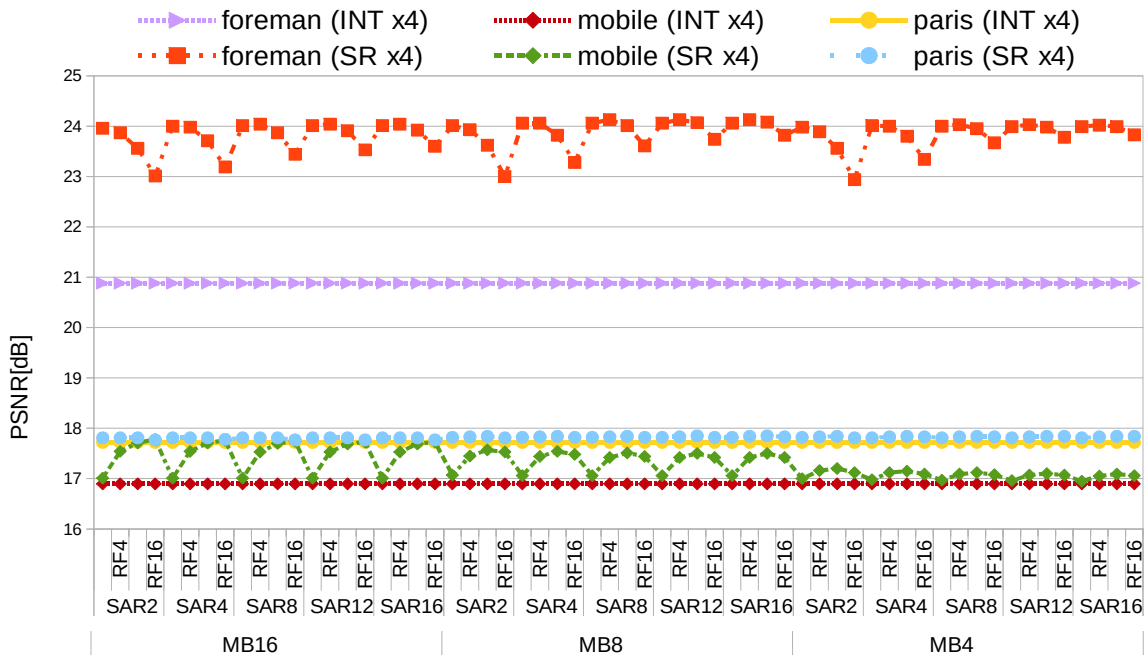


Fig. 3.14 Average PSNR observed for the super-resolved and interpolated foreman, mobile and paris sequences (initial 300 frames; scale=4).

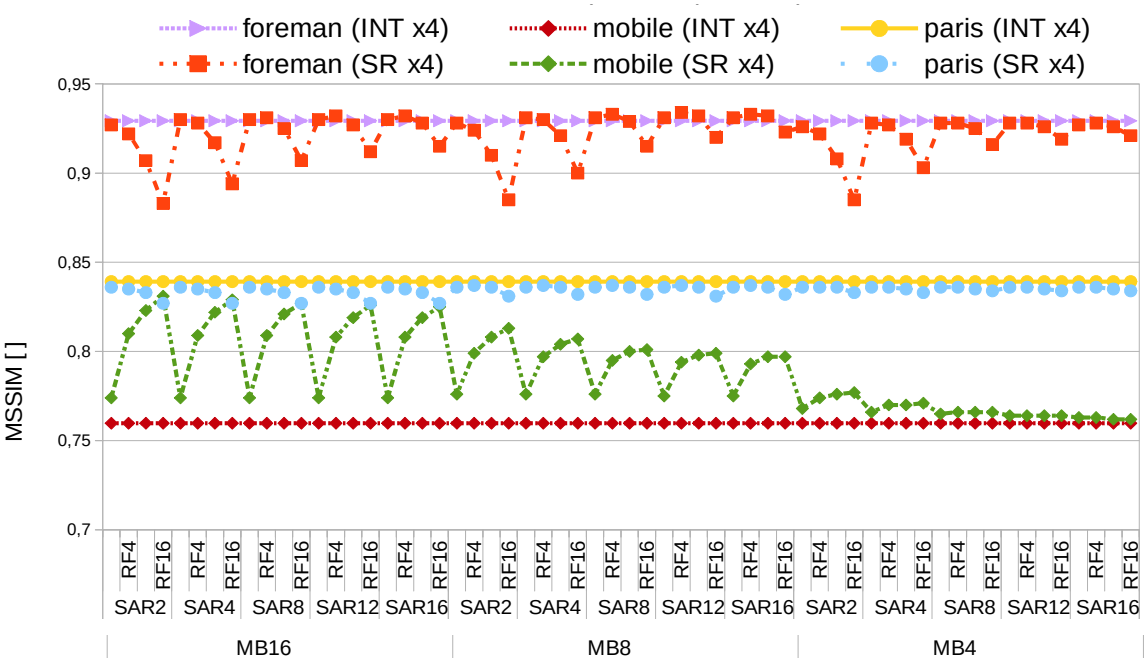


Fig. 3.15 Average MSSIM observed for the super-resolved and interpolated foreman, mobile and paris sequences (initial 300 frames; scale=4).

The only exception were the results for combinations with scale of 2 and 16 reference frames for which the worst metrics values has been observed for the smallest search area (2 pels). Also, increasing search radius resulted, in most cases, in decreasing the outcome quality as opposed to the cases observed for the foreman sequence. Once again, the difference between the diagrams obtained for different values of the scale parameter are the overall lower quality indices and the behavior of the aforementioned exceptions. For the mobile sequence the use of SRIR resulted in higher observed PSNR values than interpolation for 119 out of 120 tested parameters combinations. As far as MSSIM is considered, SRIR outperformed interpolation for most cases ($\approx 84\%$) with the exception of cases when the smallest tested MB size is used in conjunction with scale $= 2$.

In the case of the paris sequence, the highest PSNR values have been observed for the macro-block comprising 8x8 pels, maximal search area (SR of 16 pels), and 4 (scale = 2) or 8 reference frames (scale = 4). The highest MSSIM values have been observed also for the cases with medium MB size, but this time using the minimal search area (SR of 2 pels) and 2 (scale = 2) or 4 reference frames (scale = 4). The differences in the computed PSNR and MSSIM values, with the exception being the combinations with 16 reference frames, are relatively small. In case of PSNR, most of the combinations with medium MB size, again with the exception being the combinations with 16 reference frames, managed to provide better-than-interpolation quality of results. Depending on the value of the scale parameter, the worst PSNR has been noticed for either the smallest (scale = 2) or the largest (scale = 4) tested MB sizes. The increase of search area has had an insignificant influence on the observed quality for this test sequence. The increase in value of this parameter results in an initial increase of the observed PSNR and MSSIM that quickly reaches its peak and enters a state of saturation that, in case of PSNR, finally leads to a decrease in quality. This can be clearly observed for the combinations with the smallest MB size. The duration of the increase/saturation periods is correlated with the number of RFs used, with longer periods being observed for higher number of RFs used. These relations manifest themselves clearly for the case of scale = 4, where, in the case of medium and small MBs and sufficiently large search areas, some of the combinations with higher number of RFs do not reach the saturation state allowing them to outperform some of the ones with lower RF number. In terms of observed MSSIM values, interpolation have managed to outperforms SRIR for all test combinations with scale = 4 and all but two cases with scale = 2.

The PSNR and MSSIM indices computed for all the test combinations have been compared with the ones computed for the benchmark. The results—average gain/loss of PSNR and MSSIM vs the baseline interpolation quality—for scales 2 and 4 are presented in figures

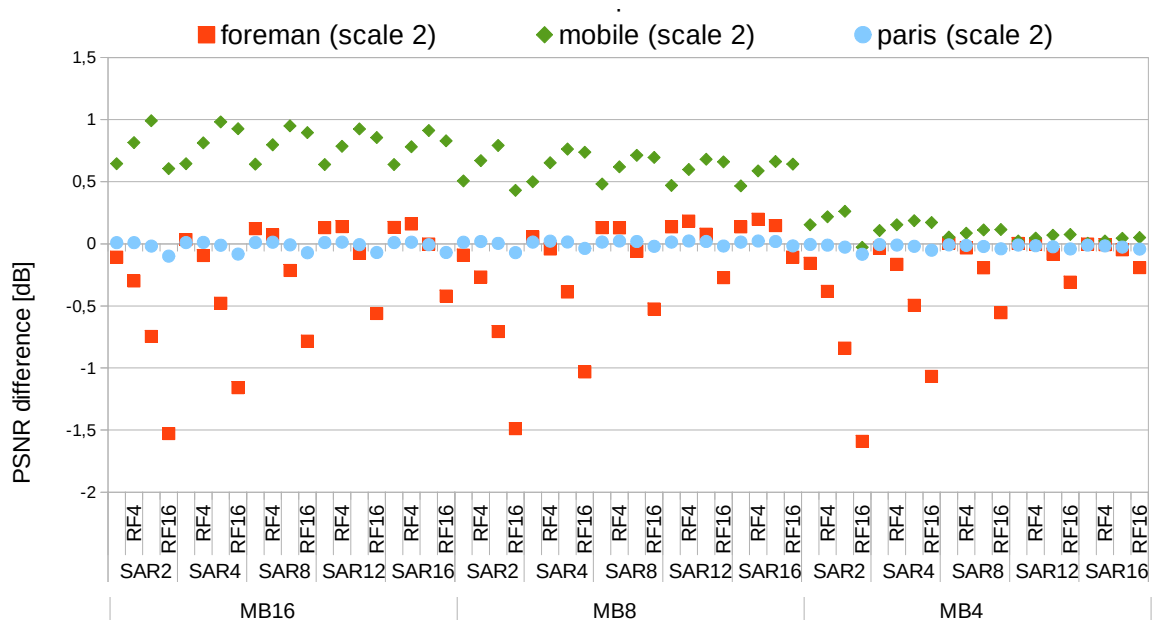


Fig. 3.16 Gain/loss in average PSNR vs interpolation for the foreman, mobile and paris sequences (initial 300 frames; scale=2).

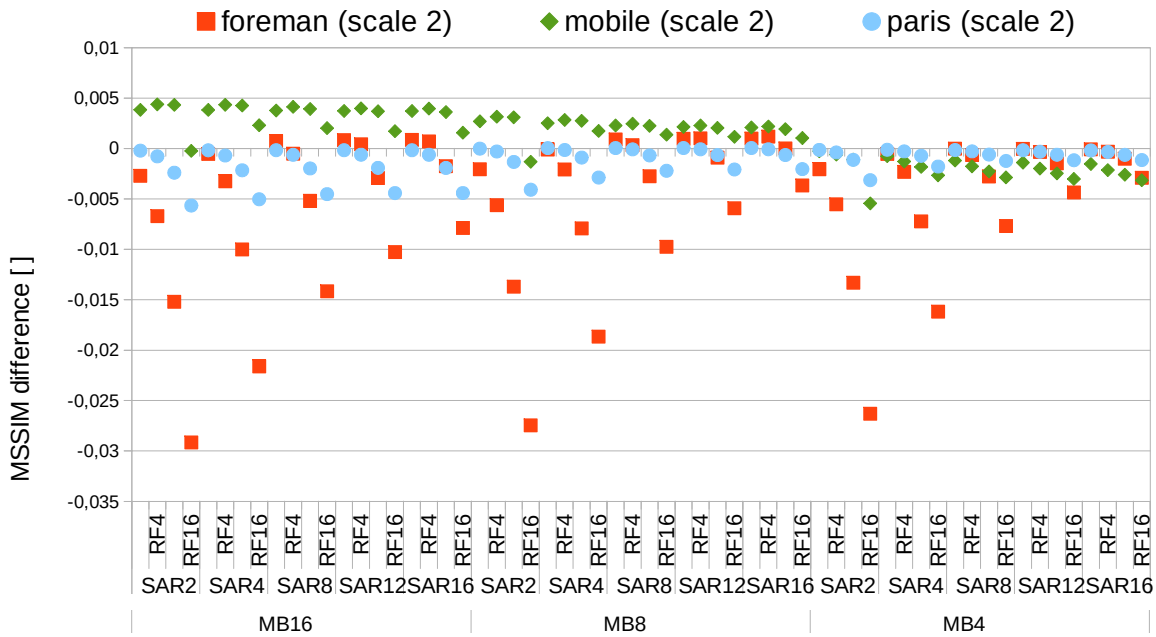


Fig. 3.17 Gain/loss in average observed MSSIM vs interpolation for the foreman, mobile and paris sequences (initial 300 frames; scale=2).

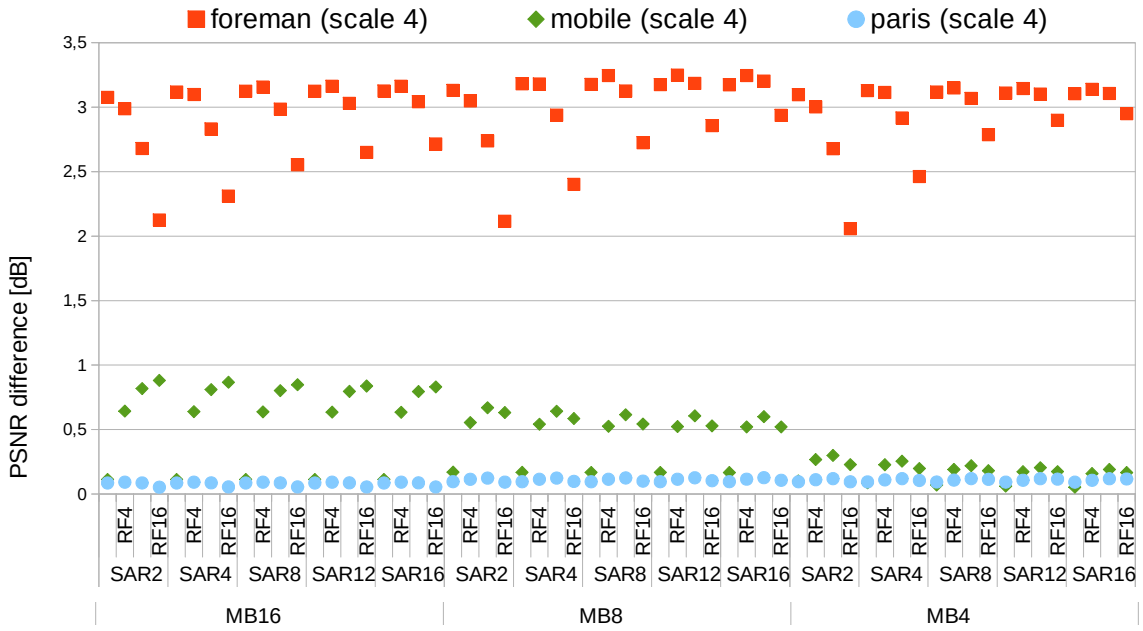


Fig. 3.18 Gain/loss in average observed PSNR vs interpolation for the foreman, mobile and paris sequences (initial 300 frames; scale=4).

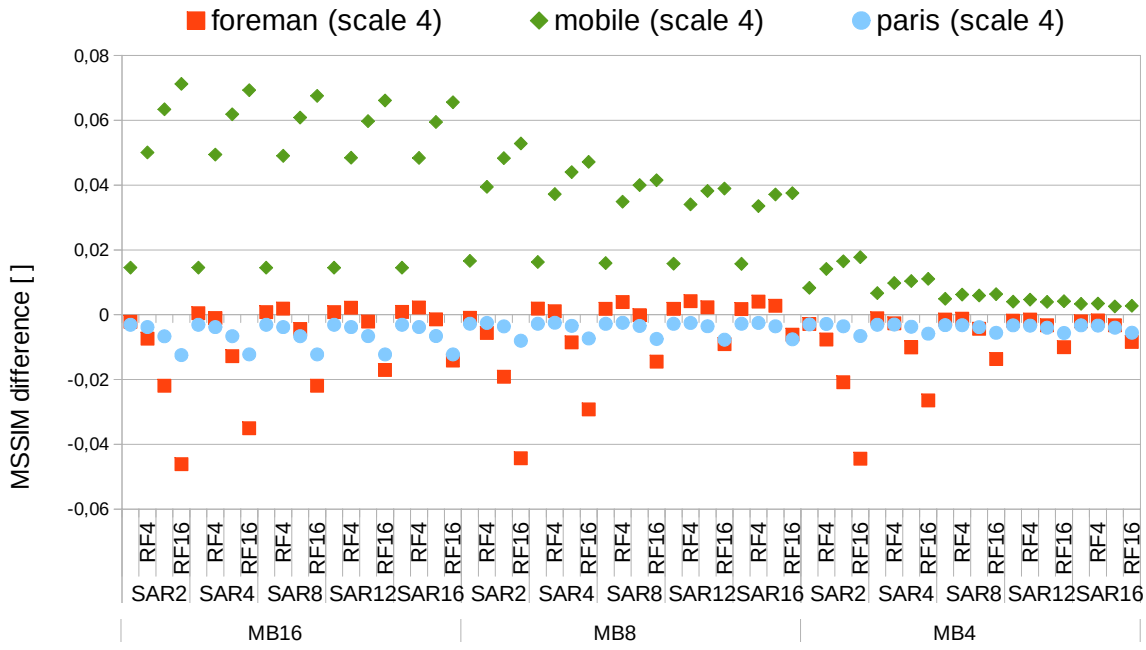


Fig. 3.19 Gain/loss in average observed MSSIM vs interpolation for the foreman, mobile and paris sequences (initial 300 frames; scale=4).

Fig. 3.16 and Fig. 3.18 (PSNR), and Fig. 3.17 and Fig. 3.19 (MSSIM), respectively. From figures presenting the PSNR gain it can be seen that for the case of the higher scale SRIR have managed to outperform interpolation for all test sequences and for 100% of test combinations. In the case of scale value of 2, *SRiuma* software has outperformed the interpolation in terms of provided PSNR in about $\approx 30\%$, $\approx 95\%$ and $\approx 60\%$ of the tested combinations, respectively for the foreman, mobile and paris sequence. The combinations that fail to provide a gain are not shared between the sequences, which complicates the process of determination of the optimal configuration and leads to the requirement of calibration. Determination of the optimal configuration is out of the scope of this thesis. Nevertheless, it is important to be aware of the existing correlations between the performance and algorithm parameters at the time of designing a hardware implementation. Finally, the gain noticed for combinations with scale = 2 is significantly smaller than the one observed for the higher scale value. For the performance measured using the MSSIM index, the results do not look as good and the percentage for which the SRIR processes outperforms interpolation has been observed to drop significantly, especially when paris sequence is considered. For scale of 4 approximately $\approx 28\%$, $\approx 100\%$ and $\approx 6\%$ of the tested combinations, respectively for the foreman, mobile and paris sequence, outperform interpolation in terms of provided MSSIM. For scale of 2 these numbers fall to approximately $\approx 20\%$, $\approx 60\%$ and $\approx 0\%$ of the tested combinations, meaning that the SRIR process has not been successfully carried out for the paris sequence.

A side-by-side comparison of computed PSNR and MSSIM values for the used test sequences has led to conclusion that, even though, SRIR is capable of outperforming the baseline interpolation in terms of PSNR (especially for higher scale values) in many cases this gain does not result in better structural similarity. Results of a study on correlation between the gradients of PSNR and MSSIM (measured as the difference vs the value of indices computed for interpolation), illustrated in figures Fig. 3.20 and Fig. 3.21, respectively for scale = 2 and scale = 4, have shown many differences in predictions of the output quality (denoted in figures as value $= 0$). From these figures it can be seen that most of the gradients are in accord (denoted as value > 0) for the combinations for which scale = 2. On the other hand, in the case of scale = 4 only the gradients for the mobile and for the foreman sequences ($\approx 20\%$) correlate. In case of the paris sequence the two measures did not agree even for one configuration for which scale was set to 4.

Some of the trends that manifest themselves in the presented results for all of the sequences need to be further explained:

- *Combinations with lower number of RFs outperforming the ones with higher number*

of RFs. For most of the carried out tests the combinations with lower number of RFs consistently outperformed the ones with more RFs. The main cause of this behavior is the lack of robustness of the implementation of the algorithm that has been used in tests. Lack or robustness means that in case of sequences with context changes and/or non-translational movements many outliers are likely to be used in the fusion step of the algorithm. The probability of outliers participation in the process increases (significantly) with the number or RFs, thus combinations with higher RF number have a higher risk of being affected by the lack of outliers elimination. This trends manifests itself clearly for the foreman and paris sequences.

- *Increase of SAR resulting in decreased quality.* It is a common case that the image registration process in not an ideal one. Moreover, the measures used in the process of ME are not guaranteed to be always accurate and usually represent a trade-off between accuracy and computational complexity. Thus, it is possible that the motion vectors determined during ME do not represent the true motion vectors. Use of non-true motion vectors during the fusion is most likely to corrupt the outcome quality. Increasing the search area beyond the bounds where the true motion takes place increases the probability of false-motion vectors being chosen over the true ones. The probability of non-true motion vectors selection is further increased for smaller MBs especially for the ones located in plain/non-textured areas [CLS+08]. Both of this trends manifest themselves clearly for the mobile sequence, where most of the motions are slow, gradual and linear and thus in nearest spatial vicinity of the a MB being registered. When rapid-movements are present increasing the radius of search area is most likely to result in gain of quality as it has been observed for the foreman sequence.

- *Greater gain observed for scale = 4 than for scale = 2.* The difference in processing carried out for different scale values is limited to the (down)scaling step of the algorithm in which values for some (high resolution) pels are being discarded. For higher scales, lower resolution input is used meaning that more pels are being discarded and are not present in the input image. Discarding many pels leads to significant loss of details. This significantly hinders the performance of the interpolation process leading to higher gain noticed for SR vs the benchmark. Additionally, for higher values of the SR scale the intermediate HR grids contain lower percentage of the pels directly copied for the LR input whose values are not allowed to be modified during the processing. This leaves more room for improvement using SRIR, given a sufficiently

large set of reference images.

- *Loss in MSSIM for combinations with gain in PSNR.* This was expected, as the core version of the SR software does not have the property of being robust, thus many outliers take part in the fusion process, with the same impact as the non-outliers. This is almost guaranteed to lead to compromised image structure (and decreased MSSIM), but can result (on average) in only small MSE changes in cases where the average difference in luma values of outliers and the reference is low enough.

Presented figures prove that, even though, for some test (algorithm values) combinations SR_{iuma} has not been capable of outperforming the baseline interpolation on a regular basis, there is at least one combination that provides a better-than-interpolation quality of results for each of the tested sequences (with the exception of MSSIM for paris). Experiments have shown that it is quite difficult to indicate one configuration that would guarantee the best performance over a set of sequences. Depending on the test sequence optimal value of macro-block size, search area, and reference frames varies. Thereby, when designing a hardware implementation a large set of super resolution parameters values should be taken into account, and the resulting organization/architecture should be made as customizable as possible.

3.5 Conclusions

The focus of this chapter was the non-uniform grid projection algorithm. The chapter started with a brief description of the algorithm flow, the used motion estimation and the fusion-based SR kernel. Next, the mathematical models of the algorithm and the aforementioned stages were developed. Following the theoretical introduction, the objective quality of the super-resolved image produced by the reference software implementation was quantitatively and qualitatively evaluated. For the needs of this evaluation, average PSNR and MSSIM values have been computed for three representative test sequences and 120 configurations of the software implementations. The obtained results have shown that the analyzed NUGPA software implementation:

(i) can provide better than interpolation quality of results expressed as PSNR or MSSIM,

(ii) the quality of the output super-resolved image is very sensitive to the changes in values of algorithm parameters,

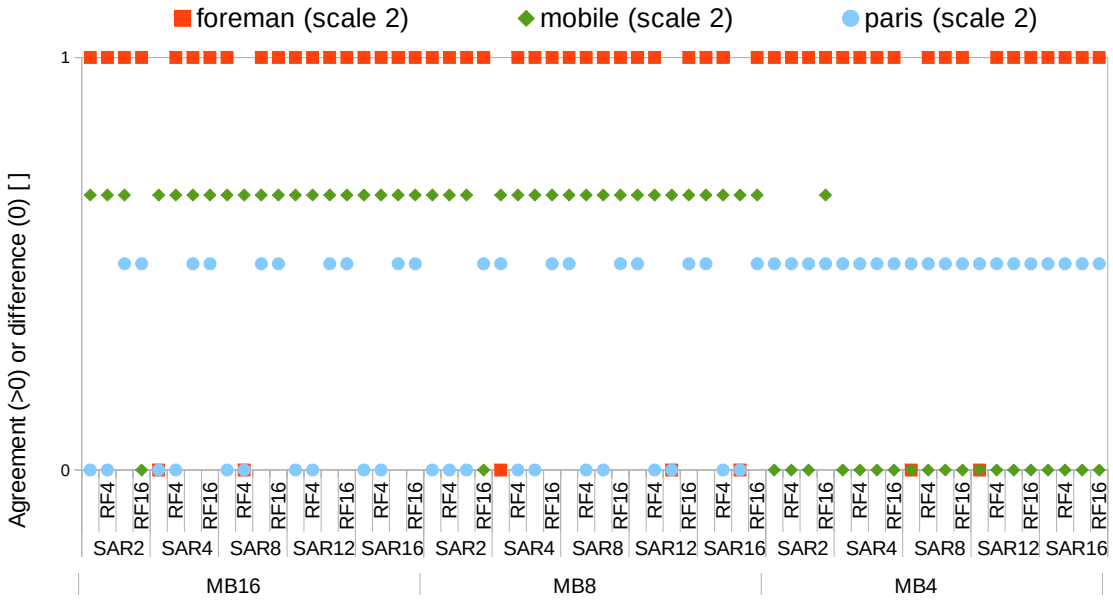


Fig. 3.20 Correlation between PSNR and MSSIM indices (scale=2).

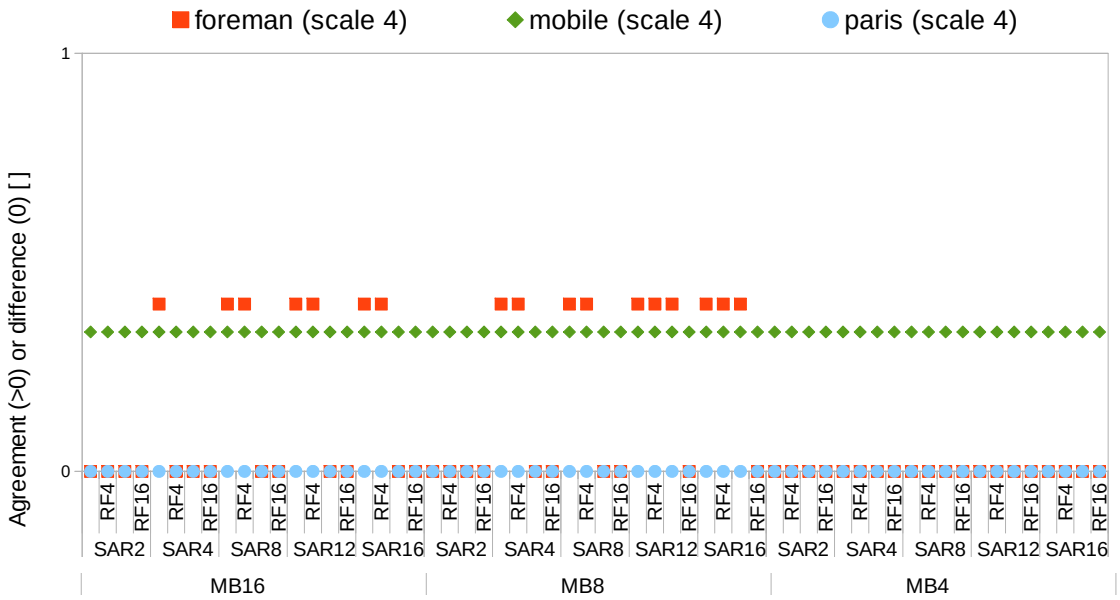


Fig. 3.21 Correlation between PSNR and MSSIM indices (scale=4).

(iii) obtaining satisfactory results requires the configuration to be fine-tuned for the particular input sequence,

(iv) does not implement sufficient mechanisms to limit the impact of outliers on the observable output quality, and

(v) presents less variance in observed quality of results for smaller macroblock sizes.

Chapter 4

Proposed super-resolution algorithm

4.1 Introduction

As seen in chapter 2, the main two factors limiting the success of hardware implementations are the iterative nature of processing and high resource occupancy. The choice of the non-iterative non-uniform projection algorithm described in chapter 3 solves the former issue, promising efficient hardware implementation. Unfortunately, the memory occupancy of the non-iterative algorithm presented in the previous chapter remains prohibitively high for an FPGA implementation that intends to avoid using any off-device memory. A quick evaluation of the reference software implementation and the algorithm data flow identified the frame buffers used by the fusion kernel as the main factor contributing to the total memory occupancy. The implemented execution flow required at least two of the frames being processed to be readily available from memory throughout the fusion process. Thus, in order to eliminate the frame-level buffers, the algorithm's execution flow needs to be modified. In particular, the data path's flow granularity has to be refined in order to reduce the algorithms memory requirements.

Execution flow granularity is one of the main characteristics of any algorithm. The spectrum of possible granularity choices spans from one pel (finest granularity) up to a (set of) frame(s) (coarsest granularity). The choice of the execution flow has a significant impact on the implementation's characteristics, among others, memory/logic occupancy, system's throughput and scalability. The deciding factor that determines the choice of the granularity of the execution flow is the type of targeted implementation and its bottlenecks. Software (super-resolution) implementations are (usually) not limited by the available memory but rather by the memory hierarchy and access latency. Thus, these implementations tend to apply processing on the coarsest type of elements possible, usually a (multiple of a) whole

frame. By doing so, a higher amount of data can be loaded at once using coalesced accesses, leading to the minimization of the number of memory accesses, the delay caused by of memory access latency, and optimization of the overall observed implementation's performance.

On the other hand, processing of coarse grain structures like a whole frame requires significant amount of memory. In most cases, the required amount of memory is too big to fit into the internal device memory forcing the use of external memory. Moreover, when targeting a PLD/FPGA device the use of a coarser granularity leads to higher entropy of the system that increase the complexity of logic required to model the system at the hardware level. That may result in higher fan-out, increase the occupancy of resources and extend the critical route latency [Koc13], meaning that significant trade-offs may have to be made in order to achieve performance that allows the SR processing to be applied in real-time. Thus, hardware implementations do not process coarse grain structures, but rather carry out the SR transformation for finer granularity elements. The most successful implementations available up-to-the-date apply SR on the finest grain possible — a single pel. This approach, known as the *optical flow*, allows to tailor the processing by breaking it into stages (iteratively executed for subsets of the data) that fit within a defined memory budget, and then use streaming to meet the target performance (throughput) [BB08, ABCC09].

Nevertheless, hardware implementation of optical flow faces problems of its own. Due to the extremely fine granularity of the flow, its implementation requires a very deep pipeline. Implementation of deep pipelines has to deal with the overhead introduced by inter-stages glue logic and/or latches, high logic occupancy (less room for optimization — low reuse of resources), high power dissipation due to repeated transfers of pels belonging to the *reference structures* (RS; corresponds to RF or SA). All of the above have a significant impact on the overall system efficiency in terms of resource occupancy. In many cases, these requirements are so high that only a sub-optimal version of the algorithm can fit in the device, leading to a sub-optimal quality of the super-resolved image.

In this work we propose a novel flow for the non-iterative NUGPA. The proposed flow internally applies full SR processing on a small set of pels (called *macro-block*). By effectively forming a cross-over point between the flows that use coarse or very fine granularity elements, the proposed flow is expected to allow to combine their strengths while alleviating their weaknesses.

4.2 Execution flows of the NUGPA

As aforementioned, SRIR is typically carried out in two steps: extraction of data that characterizes the input and reconstruction (where possible) of missing data. The former task encapsulates a series of computations needed to estimate values of metrics characterizing the input data. These metrics are used in the latter step (called the *super-resolution kernel*, SRK) to identify pels from *reference structures* that when merged with the data being upscaled can lead to additional details reconstruction. The SRK processing ends producing a higher resolution image (called the super-resolved image).

In order register the images and extract the hyperdata the NUGPA uses *block matching* (BM) motion estimation. Detailed description and evaluation of motion estimation is presented in [BAA05] and [CLT+08]. This section focuses solely on the SRK, presenting and comparing in detail the reference and the proposed execution flows.

4.2.1 Reference coarse grain execution flow

The base for the study has been a software implementation of a complete SR system, comprising ME and NUGPA SR kernel coded using ANSI C. The SRK execution flow implemented by this software is presented in Fig. 4.1. The SRK flow comprises two internal subflows: one for the luma component and another for the chroma components processing. Chroma picture elements carry a significantly lower energy concentration than luma pels, and thus have a significantly lower impact on the subjective quality of the super-resolved image [IP91]. Therefore, super-resolution is applied only on the luma pels. Spatial resolution of chroma components is augmented using bilinear interpolation [GW08], meaning that no additional details are being reconstructed for these components. The luma pels undergo full NUGPA processing presented in Fig. 4.1:

1. The frame to be processed is loaded from memory. As the luma and chroma components are to undergo different processing, thus, when presented with a YUV formatted LR input, the software starts the processing by separating one type of the picture elements from the other.

2. Value ‘0’ is used in step 7 to identify the pels whose value has to be created using interpolation. This allows to eliminate the need of an extra memory to store the map of pels to be interpolated. In order to distinguish the input LR pels with value equal to ‘0’ (called *zeroes*) from the pels marked for interpolation (called *holes*) the original

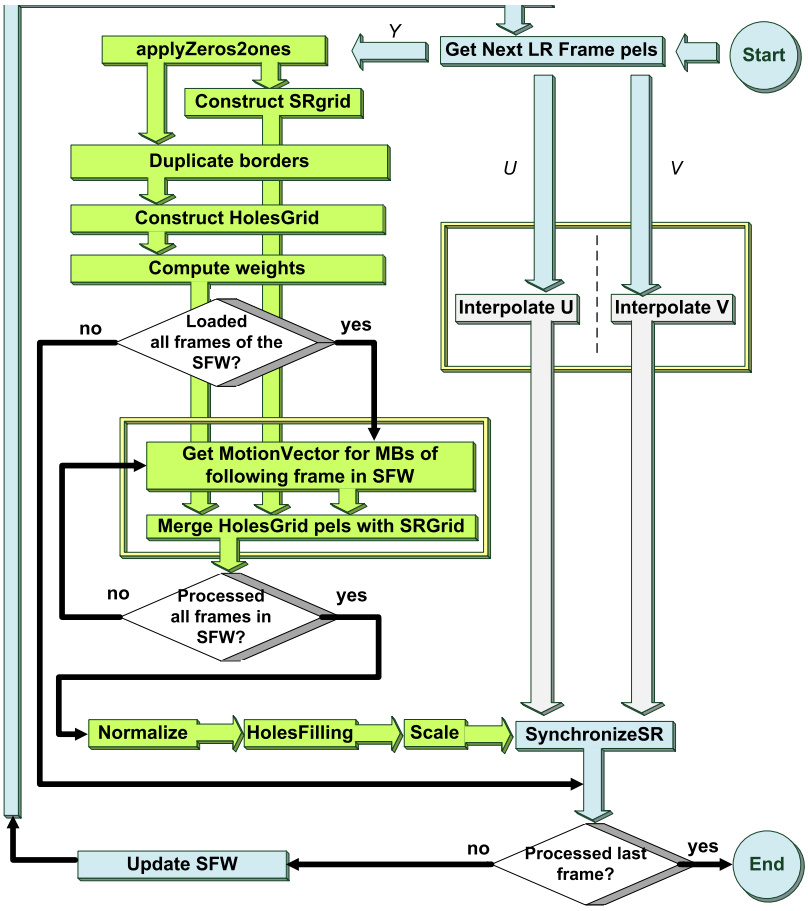


Fig. 4.1 The frame-level execution flow of the NUGPA super-resolution kernel.

value of the zero is changed to be equal to '1'. This process will be referenced to as the *zeroes2ones* (or *zeros2ones*) transformation.

3. Two HR representations of the transformed LR frame, the *holesGrid* and the *srGrid*, are created. The HR representations dimensions are several times bigger than the LR ones. For quarter pixel ME the growth is of a factor of 4 for each dimension (resulting in total size growth of 16 times). The *srGrid* is constructed directly from the output of the *zeroes2ones* transformation. For *holesGrid*, pels belonging to the borders (outer rows and columns) of the *zeroes2ones* outcome are duplicated before the spatial augmentation takes place. Border duplication is carried out in order to (somewhat) solve the aperture problem that could arise in ME.

4. Steps 1–3 are repeated until all frames from the sliding frame window have their HR representations.

5. Candidate pels specified by the ME metrics are extracted from all *holesGrid* of frames belonging to the SFW. Extracted data are fused with the *srGrid*.

6. Each element of the *srGrid* is divided by the sum of weights associated with it. Steps 5–6 make up the *shiftAdd* transformation.

7. The outcome of the *shiftAdd* transformation is usually composed of many pels with value equal to zero. These pels values get estimated by computing the mean value of pels belonging to a square-shaped neighborhood centered at the hole's position. The process in which a hole is assigned a value is called *holesFilling*.

8. The *srGrid* after *holesFilling* is the super-resolved frame, with the scale factor equal to precision of used in image registration (equal to 4 for quarter pixel precision). When a different scale factor is specified, *srGrid* has to be adjusted to the expected dimensions. This task is carried out by the *scale* function. This function discards certain pels, producing the final super-resolved representation of the luma component.

9. Finally, the synchronization of the luma and chroma flows takes place. This step is needed in order to provide the super-resolved frame in YUV format.

10. The SFW update policy is applied. Typically, one of the reference frames is discarded (in the FIFO order) and a new frame is loaded in its place.

11. Steps 1–10 are repeated for each processed frame.

The presented processing is carried out at frame-level. Transitions from one step to another take place only after step's transformation has been applied on all pels fo the frame being super-resolved. The above-presented processing requires that HR representations of all frames from SFW be available from memory throughout the complete duration of the super-resolution process.

4.2.2 Proposed finer grain execution flow

The used *SRiuma* reference software originally operated at frame-level, posing memory storage requirements precluding the planned FPGA implementation using only the on-device available *Block RAM* (hereafter BRAM) memory. In this work we propose a novel flow for a non-iterative fusion algorithms that mitigates the problem of high memory occupancy associated with the presented coarse-granularity flow. In contrast with the frame-level processing, which requires that HR representations of all frames from SFW be available from

memory throughout the complete duration of the super-resolution process, the proposed approach requires that only search areas, instead of whole frames, be available from memory. As a result, the requirements for memory storage are reduced, at the expense of increasing the number of memory accesses. Processing of a patch instead of only a singular pel allows more optimized logic synthesis which is expected to result in moderate (lower than for the optical flow) logic occupancy allowing full algorithm implementation on the targeted FPGA device.

The proposed execution flow of the NUGPA is presented in Fig. 4.2. This flow is similar to the FL flow (compare with Fig. 4.1). The main differences between the flows are: 1. the size of the basic element being stored in local memories, 2. substitution of reference frames by search areas, 3. the fact that transition to the subsequent step of the algorithm is carried out after one macro-block (not one frame) has been processed, and 4. the necessity of managing inter-macroblock dependencies, not present in frame-level processing, that significantly complicate the *holesFilling* process. These changes result increase the complexity of memory management and call for different buffering schemes in order to mitigate the expected increase in memory traffic.

4.3 Evaluation of memory occupancy of the reference and the proposed NUGPA execution flows

In order to quantify the memory storage requirements and their reduction associated with the change to the proposed *macroblock-level* (MBL) flow a theoretical study based on the available software implementation has been carried out.

First, the memories used by the implementations had been identified. Then, their size and its dependency on the SR parameters have been evaluated. These steps have led to the creation of equations determining memory size for each type of identified memories. Based on these equations the memory storage requirements of each step of the investigated execution flows have been quantified and compared.

4.3.1 Identification of memory storage requirements

In order to provide a quantitative evaluation of the memory storage requirements of the reference and the proposed NUGPA execution flows: (i) the memory storage requirements of the flows' steps had to be identified and modeled as a function of algorithm's parameters,

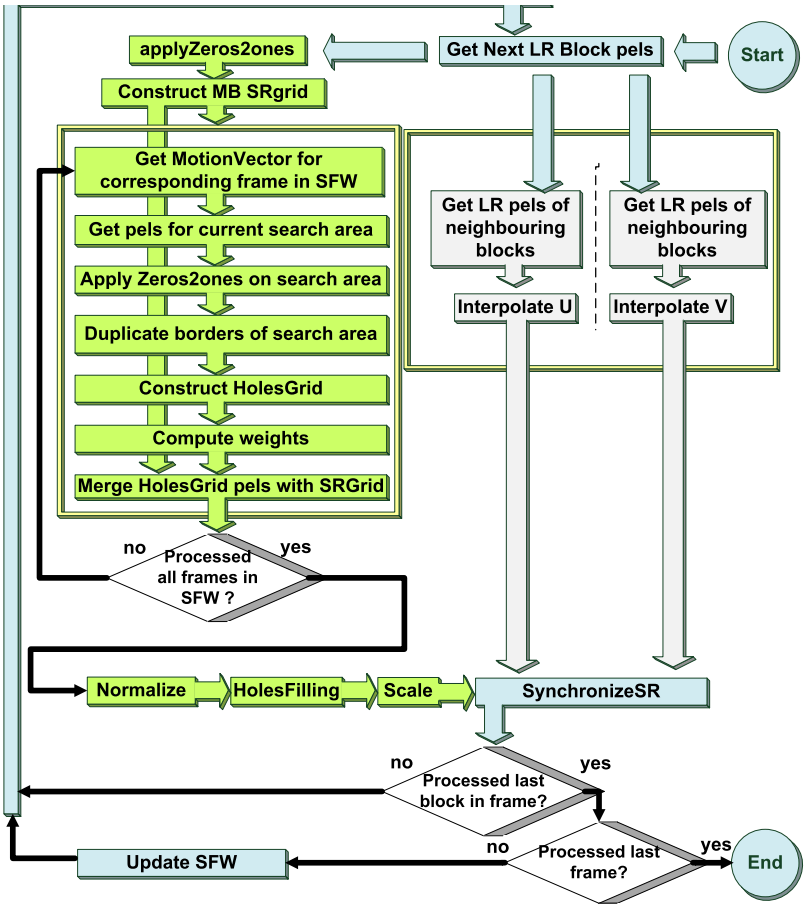


Fig. 4.2 The macroblock-level execution flow of the NUGPA super-resolution kernel.

(ii) the memories instantiated by each step had to be identified and their size modeled using the equations established in step (i).

4.3.1.1 Memory types and their dependency on algorithm parameters

From an analysis of the NUGPA flow and the structures defined in the code of the software implementation we have inferred that the parameters that determine the size of memories are: (i) the number of frame rows FR_{rows} and columns FR_{cols} from which the size of the LR representations is derived, (ii) the macroblock width MB_{width} which determines the size of the macroblock (we assume MBs are square shaped), (iii) the ME precision $precision_{me}$ which determines the size of the HR representations, (iv) the SR scale *scale* which determines the size of the super resolved representations, (v) search area radius (SAR) which determines the size of the search area, and (vi) the interpolation window int_{window} which determines the size of the memories used to solve data dependencies in *holesFilling*. Short

TABLE 4.1 *Equations for memory occupancy estimation of the identified memory types used by the investigated software implementations.*

Memory type	Equation for size in elements	Group
MB_{lr}	$= MB^2_{width}$	
MB_{hr}	$= MB_{lr} * precision^2_{me}$	MB
MB_{sr}	$= MB_{hr}/scale^2$	
FR_{lr}	$= FR_{rows} * FR_{cols}$	
FR_{hr}	$= FR_{lr} * precision^2_{me}$	/frame
FR_{sr}	$= FR_{hr}/scale^2$	
SA_{lr}	$= (MB_{size} + 2 * SAR)^2$	/SA
SA_{lr}	$= precision^2_{me} * SA_{lr}$	
int_{buffer}	$= (FR_{rows} + 2 * MB_{size})$ $*int_{window} * precision_{me}$	Auxiliary
$reorder_{buffer}$	$= (SFW - 1) * FR_{lr}/MB_{lr}$	

descriptions of these parameters, along with the range of values tested, has already been presented in Table 3.1 on page 90.

Based on the SR parameters dependency ten memory types, presented in Table 4.1, are defined, systematized and divided into two groups: (i) MB/frame/SA representations and (ii) auxiliary memories. The MB/frame/SA representations comprise three subtypes of representations: the low resolution (LR), high resolution (HR), and super-resolved (SR) representations. The LR representation precision corresponds to the full-pixel accuracy of the ME process. The HR representation is the LR input upscaled to the maximal precision used in the ME process ($precision_{ME}$). The SR representation is the output format obtained from the HR representations by means of decimation.

The frame-level implementation uses only the MB/frame types of memories. The auxiliary memories encapsulate memories required for operation at MBL and are not used by the FL implementation (SA memories are also MBL specific). The auxiliary memories are used for hyperdata (ME metrics) reordering and storing (in the $reorder_{buffer}$, (RB)) and for pels buffering (in the int_{buffer}) in order to handle data dependencies arising in the interpolation process. The LR and HR representations of frames are absent from the MBL implementation: instead the search area (SA) and macroblock (MB) representations are used. The memories used in order to store the sums of weights maintained for each pel of the HR representation during the fusion process, required to compute the final high resolution pel value, for weights sum smaller than 256 (case considered in this work) are of size of the MB/FR_{hr} representation.

The goal of memory identification is to form equations that model the size of all defined

memory types as a function of the SR parameters presented in Table 3.1. All memory sizes depend on frame (FR) and/or MB size. The HR and SR representations are additionally influenced by $precision_{me}$ value which specifies grids resolution (size), and, in the case of the latter representations, also by the SR scale value. The search area radius (SAR) is used for SA size determination. The most upper left pel of the currently processed MB is considered the reference point of the search area. The SA spans from the reference point in all direction over the number of pels specified by the search area radius. The search area additionally includes a MB_{size} pels in the right and bottom direction in order to be capable of loading all pels of MB whose most upper left pel is within the SAR distance from the search area reference point. A large SA usually leads to a more accurate determination of the motion, at the cost of: an increase of the number of computations required to carry out motion estimation, and additional memory storage requirement in SRK. The int_{window} determines the size of buffer for data dependency resolution in the *holesFilling* step.

The resulting equations that model the size of identified memory types as function of the SR parameters are presented in Table 4.1, where sub-indexes LR, HR, SR represent, respectively, the low resolution, high resolution, and super-resolved representations of a frame (FR), macroblock (MB) and search area (SA).

4.3.1.2 Memory occupancy of execution flows' steps

The analyzed SR software forms the base for a hardware implementation, and as such, it should be viewed as a representation of a system comprising prototypes of not only software but also hardware entities. Therefore, in order to present a more comprehensive memory requirements evaluation, each of the steps of the algorithm execution flows is considered a representation of a hardware entity and is to be analyzed independently. The goal of the analysis is to identify the type and size of memories that these entities use.

The *zeroes2ones* entity receives the input data LR pels, applies the *zeroes2ones* transformation and stores the LR data in output memories. Thus, the memory requirements of the entity are two LR FRs (at FL) or one LR MBs plus one LR SA (at MBL).

The *duplicateBorders* entity which carried out the duplicate borders transformation has been eliminated from the software implementation and the transformation is now emulated by a modified addressing scheme used by the *shiftAdd* entity. This optimization is similar to the one described in Section 4.6.1.

After being processed by the *zeroes2ones* entity the LR data are passed to the entities that perform the *up-holes* transformation. The outcome of this transformation is a HR representation of the LR input. For the FL flow the input of the *upHoles* and *upGrid* entities,

which carry out the *up-holes* transformation, is a LR frame, thus, each of these entities requires one instance of memory of HR frame type. For MBL the inputs are a LR MB and a LR SA, respectively.

The *shiftAdd* entity receives the HR representations produced by the *upHoles* and *upGrid* entities and stores them in local memories. When all frames/SAs of the SFW are available from local memories, the *shiftAdd* transformation is carried out. The relevant data are extracted from the RFs/SAs and added to the HR representation of the FR/MB being processed (received from *upGrid*, stored locally). For each pel of the HR representation a sum of weights is maintained and used to compute the final HR pel value. For weights sum smaller than 256 this memory size is equal to the size of HR FR/MB representation. For frame-level implementation this translates to requirements of SFW+2 memories of size (in elements) of a HR FR type. In the case of MB-level operation, SFW-1 memories of size HR SA and three of size HR MB are required.

The *holesFilling* (interpolation) is performed in a pel-by-pel manner. For frame-level operation the interpolation input and output is a HR frame. Thus, two memories of HR frame size are required. Nevertheless, the input can be shared with the *shiftAdd* lowering the memory requirements to one HR frame. For MB-level operation the input data are the pels of HR MB being processed. Thus, not all data necessary for the interpolation process are readily available from the input memory and the interpolation cannot be carried out for pixels belonging to borders of the MB. The pels that cannot be processed due to unfulfilled data dependencies have to be stored until the data dependencies are resolved. The number of pels that have to be stored is determined based on the width of the frame and the int_{window} parameter that specifies the number of neighbouring pels that contribute to the interpolation process.

The outcome of the *holesFilling* is received by the *scale* entity. This entity, given a HR input, produces its SR version that has the size determined by the scale parameter value. Thus, when operating at FL this entity requires one FR of HR and one of SR size. When operating at MBL the memory requirements are of one MB of HR and one MB of SR size.

The above description of the entities concludes the description of the entities shared by both implementations. The system operating at MB-level produces for each given LR MB its SR representation. Nevertheless, the outcome of the SR process, and the input of the display routine, is supposed to be a super-resolved frame. Also the input from ME is not streamed in the order required by MBL flow.

In order to handle the aforementioned MB-level specific issues two entities that are not present in the frame-level system have been defined, namely: the *reorderBuffer* and the

TABLE 4.2 *Equations for memory requirements of the algorithm steps defined for the analyzed software implementations.*

Entity/Step	Frame-level	MB-level
zeroes2ones	$2 * FR_{lr}$	$MB_{lr} + SA_{lr}$
upHoles	FR_{hr}	SA_{hr}
upGrid	FR_{hr}	MB_{hr}
shiftAdd	$FR_{hr} * SFW$	$SA_{hr} * (SFW - 1)$
	$+FR_{hr}$	$+2 * MB_{hr}$
	$+FR_{hr_{weights}}$	$+MB_{hr_{weights}}$
holesFilling	FR_{hr}	$MB_{hr} + int_{buffer}$
scale	FR_{sr}	MB_{sr}
reorderBuffer	—	$reorder_{buffer}$
frameReconstruction	—	FR_{sr}

frameReconstruction entities. The former entity handles the reordering of the ME metrics produced by the block matching entities to an order suitable for the MB-by-MB processing. The buffer used for reordering (of $reorder_{buffer}$ type), called the reorder buffer, needs to store (SFW-1) ME data words for each MB of a frame being super-resolved, requiring an entry for each RF of the SFW. The amount of memory required for this buffer implementation is significant. The task of managing the buffer is carried out by the *reorderBuffer* entity.

The *frameReconstruction* is introduced in order to allow seamless interoperability with the display controller, which expects at its input a super-resolved frame (and not a super-resolved MB). This entity reconstructs the super-resolved frame from super-resolved MBs produced by the MB-level system.

In this work we assume that the task of the synchronization of luma and chroma components does not require additional memory resources as the luma pels are stored in the output memories of the *frameReconstruction* entity, and the much less computationally demanding process of interpolation of chroma pels is left to be implemented in software and will not use internal FPGA resources.

Based on the carried out analysis equations modeling each entity memory requirements have been created. These equations are presented in Table 4.2. This analysis assumes sequential execution of code encapsulated by defined entities. This assumption allows sharing of input/output memories between the communicating entities (i.e. the output memory of entity *upGrid* is utilized as the input memory of *shiftAdd*, and the output of entity *shiftAdd* is used as the input of *holesFilling*).

TABLE 4.3 *Computed minimal and maximal memory occupancy of the super-resolution kernel for frame- and MB-level flows and QCIF input.*

Execution flow	Frame-level		MB-level	
Computed value of	Min	Max	Min	Max
Memory occupancy [KB]	3317	9158	122	1051
Vs. frame-level equivalent[a][%]	100	100	3.67	11.47
Vs. MB-level equivalent[a][%]	1416	871	100	100

[a] with the same values of parameters.

4.3.2 Quantitative analysis of memory occupancy of the frame- and MB-level execution flows

Having determined the memory structures used by each entity, the amount of memory (in bytes) required by each entity has been evaluated. The memory storage requirements of the frame- and the MB-level implementations have been computed based on the equations from Table 4.1 and Table 4.2. The computations have been carried out for a set of 96 combinations of SR parameters values. The tested combinations set has been created by sweeping through the MB_{size} (4, 8 and 16), SAR (2, 4, 8 and 16), number of RFs (2, 4, 8 and 16), and the scale (2 and 4) parameters values.

The minimal and maximal values of memory storage requirements obtained for the frame- and MB-level flows are presented in Table 4.3. (The results are obtained for memories with each element requiring 1 byte. The only exception is the reorder buffer, whose entries occupy 12 bytes.) The comparison shows that the change to the MB-level flow is expected to result in a significant reduction of *Total Memory storage Requirements* (TMR). Minimal memory required by the FL and MBL implementations is 3316 KB and 122 KB, respectively. Maximal memory use is 9158 KB for FL and 1051 KB for MBL execution flows. The carried out study shows that for the considered range of values of the chosen parameters the macroblock-level implementation uses no more than 14.7% of the equivalent FL system (for a QCIF input).

4.3.2.1 Frame-level flow memory occupancy as a function of super-resolution parameters

The expected minimal (3317 KB) and maximal (9158 KB) memory storage requirements for the FL version have been computed, respectively, for minimal and maximal value of the RF number and scale parameters. The SAR parameter does not influence the FL memory

TABLE 4.4 *Memory occupancy of the frame-level super-resolution kernel for investigated algorithm parameters values.*

		Number of reference frames				
		2	4	8	16	
Scale	2	3317	4109	5693	8861	Memory
	4	3614	4406	5990	9158	size [KB]

requirements. The total occupancy for different MB sizes have been investigated. The MB size does not have a significant impact on the TMR of the frame-level implementation. Thus, FL implementation total memory storage requirements for different MB_{width} values are approximated as being constant. The estimated total memory storage requirements for FL implementation are presented in Table 4.4.

From all of the considered parameters the frame-level super-resolution kernel memory storage requirements are mostly determined by the number of used RFs. Each additional reference frame support results in an increase of TMR approximated at around 4% of the maximal TMR noticed for the FL implementation. Both implementations were tested with two values of the scale parameter (2 and 4). The switch to higher value adds a constant value of 3/4 the size of a HR frame to the TMR. This makes up for less than 3% of the maximal total memory requirements noticed for the frame-level SRK.

4.3.2.2 MB-level flow memory occupancy as a function of super-resolution parameters

The MBL implementation TMR for tested combinations of parameters values are presented in Fig. 4.3(a). The minimal TMR are noticed for the MB_{width} of 8 (comprising 8x8 pels) with other parameters set to minimal values. Maximal TMR were noticed for the MB_{width} of 4 (4x4 MBs) with other parameters set to maximal values. The MB_{width} of 4 results in significantly higher number of entries in the RB leading the TMR being determined by the RB size. For other MB sizes the RB impact is of much lesser importance leading to lower TMR. High memory occupancy noticed for the MB_{width} of 16 (16x16 MBs) is caused by the increased size of buffers that depend on the MB_{size} (representations of MB, grids and interpolation buffer). The change of the scale value influences both of the investigated flows in the same manner — adding a constant value of 3/4 the size of a HR FR. For MBL flow the impact of scale change is more significant due to smaller TMR. For MBL the TMR increase linearly with the increase of the SAR. Like for to the FL version, the TMR of the MBL version scale linearly with the number of RFs.

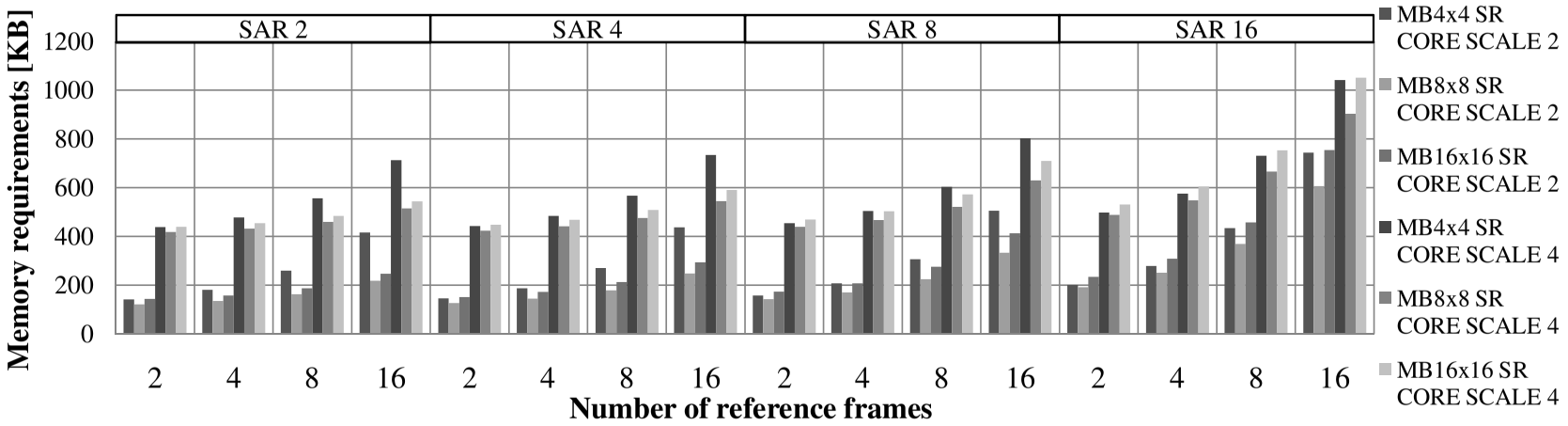
In order to measure the effect of the SAR and RF values on the TMR we introduce the $mem_{size_{inc}}$ and $mem_{size_{inc/RF}}$ figures. The $mem_{size_{inc}}$ figure represents the percentage of additional memory requirements (mem_{size}) noticed for a given SAR and RF combination (mem_{size}(SAR, MB_{size},scale,RF)) against the memory requirements noticed for SAR implementation with only two RFs (mem_{size}(SAR, MB_{size},scale,2)). The $mem_{size_{inc/RF}}$ represents the percentage of increase in TMR associated with addition of one RF. The $mem_{size_{inc}}$ and $mem_{size_{inc/RF}}$ have been computed using (4.1) and (4.2), respectively.

$$mem_{size_{inc}} = 100 \times \left(\frac{mem_{size}(SAR, MB_{size}, scale, RF)}{mem_{size}(SAR, MB_{size}, scale, 2)} - 1 \right) \tag{4.1}$$

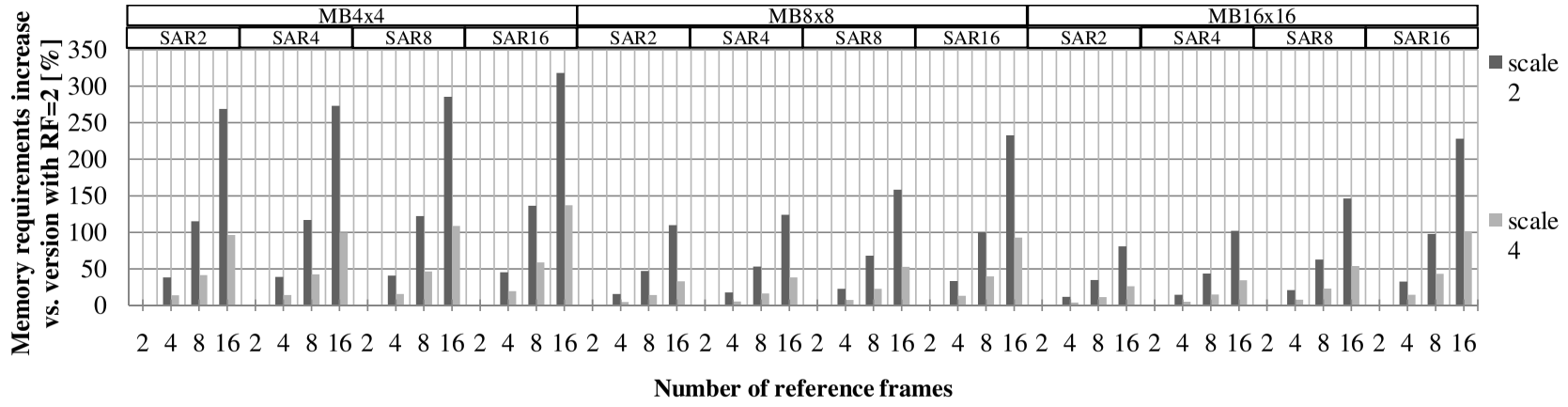
$$mem_{size_{inc/RF}} = \frac{mem_{size_{inc}}}{RF} \tag{4.2}$$

Computed $mem_{size_{inc}}$ and $mem_{size_{inc/RF}}$ values, presented in Fig. 4.3(b), show that the relative increase in memory occupancy introduced by inclusion of additional RFs is higher for smaller MBs. For 4x4 MBs the cost of switching from 2 to 16 RFs can require up to 318% of additional storage (about 22.8% per RF). For 8x8 and 16x16 MB the cost is 232% (16.6%) and 226% (16.1%), respectively. The increase in TMR introduced by additional RFs lowers for higher scale values due to lower share of the *holesGrids* in the TMR and increases with increasing the SAR (larger SAs). The $mem_{size_{inc/RF}}$ for FL, based on values from Table 4.4, is computed at 11.9% and 11% for scale value of 2 and 4, respectively.

The Fig. 4.3(b) shows that the impact of the SAR is stronger for bigger MBs — resulting in higher peak-to-peak difference between $mem_{size_{inc}}$ values estimated for subsequent RF values. This is caused by the fact that *holesGrids*/SA size is proportional to the squared sum of SAR and MB_{width}. This sum value is more sensitive to changes in the SAR for bigger MBs.



(a) Total memory storage requirements for different parameters values.



(b) Additional memory storage requirements associated with introduction of additional reference frames support as function of search area radius.

Fig. 4.3 Memory requirements of the macroblock-level super resolution kernel.

TABLE 4.5 *Computed memory storage requirements of defined entities of the analyzed software implementations*

Flow	Frame-level			MB-level		
Share in TMR [%] / Entity	Min	Max	Avg	Min	Max	Avg
zeroes2ones	0.55	1.5	1	0.01	1.07	0.22
upHoles	4.33	11.95	7.99	0.11	15.27	2.97
upGrid	4.33	11.95	7.99	0.03	2.77	0.55
shiftAdd	54.8	84.92	70.16	0.6	76.68	19.15
holesFilling	4.33	11.95	7.99	0.12	3.73	0.93
scale	1.12	10.96	4.9	0.01	0.7	0.11
reorderBuffer	—	—	—	0.73	80.68	20.44
frameReconstruction	—	—	—	10.51	93.24	55.66
Vs. frame-level equivalent[a]	100	100	100	2.93	14.73	8.2

[a] with the same values of parameters.

4.3.2.3 Share of steps in total memory storage requirements

The frame-level memory occupancy is dominated by the memories instantiated by the *shiftAdd* entity. The carried out study has identified this entity as the one with the highest minimal (54.8%), maximal (84.92%) and average (70.16%) share in total memory storage requirements. The rest of the entities average share is lower than 8%.

The change from frame-level to macro-block-level flow significantly changes the distribution of entities share in total memory storage requirements. The impact of the *shiftAdd* entity is significantly lowered. This entity, with the minimal, maximal and average share of 0.63%, 77.94% and 20.46%, respectively, remains (by far) the one with the most share in the TMR of all intrinsic (non-*adapter*) entities of the SRK. However, the total memory occupancy is mainly determined by the *frameReconstruction* entity, with the minimal, maximal and average share of 13.13%, 94.61% and 59.73%. The entity with the third most important impact on the TMR, with average share of 14.8%, is the *reorderBuffer* (adapter) entity.

High *frameReconstruction* share shown in Table 4.5 is caused by the fact that the output memory of this entity is of frame-level type. This entity share rises slightly with the increase in MB size and lowers significantly for higher SAR and RF number values, for which the share of entities comprising memories for search area processing (*zeroes2ones*, *upHoles*, and *shiftAdd*) increases. The *reorderBuffer* entity dominates the TMR for the smallest MB size (4x4) and scenarios with high number of reference frames with small SAR.

The *frameReconstruction* entity is the one not only with the highest maximal, but also

with the highest minimal share ($> 13\%$) in TMR. The latter is mainly caused by the fact that this entity output memory is of frame-level type and only depends on the frame size (constant), ME precision (constant) and the scale (variable) parameters. In fact, for scale factor value of 4, low number of RFs (lowering the impact of the *shiftAdd*), medium (when SAR is small) and large MB sizes (that lower the impact of RB), this entity becomes dominant with up to more than 94.5% of share in the TMR.

For small MB size the number of MBs in a frame is high. The number of entries (per RF) in the RB, and thus its size, is directly related to the number of MBs in a frame. Thereby, for small MB size (4x4) and high number of RFs the RB dominates the TMR with a share as high as 71%. For maximal MBs size (16) and minimal number of RFs the RB size becomes insignificant with share of less than 0.5%.

The *shiftAdd* entity memory occupancy depends on the number of RFs, MB size and SAR. Increase in any of these parameters results in *shiftAdd* entity memory storage requirements growth. The increase in RFs results in a significant increase not only in *shiftAdd*, but also in RB requirements. Nevertheless, changes in MB size effect in opposite changes in these two entities requirements. Decrease of the MB size causes a significant increase in the number of necessary entries in RB and a decrease in the number of pels that make up a SA. The latter leads to a reduction of the size of the HR SA representations. For big MB sizes, the size of the RB becomes insignificant, while the memory requirements of the *shiftAdd* increase. As the *frameReconstruction* memory storage requirements do not depend on the SAR and RFs parameters, for large MB sizes, high number of RFs and large SARs the *shiftAdd* entity dominates the memory requirements, reaching up to almost 78% of TMR. The RFs number and SAR impacts significantly also the *upHoles* entity, but this entity memory occupancy is never higher than the one of *shiftAdd*.

As shown, the MB-level implementation TMR are mostly dominated by the storage requirements of the frame-to-MB adapters. Nevertheless, these entities do not form an intrinsic part of the SRK and are only used for seamless frame-to-MB adjustments. Thus, their inclusion in the comparison may lead to a false interpretation of the changes in the memory requirements of the intrinsic parts of the SRK associated with the switch to MB-level processing flow.

In order to investigate the changes in memory storage requirements of the intrinsic SRK entities the adapter entities have to be excluded from the TMR evaluation. In Fig. 4.4 a graphical presentation of the changes in memory requirements of the intrinsic SRK entities is shown. The requirements of the *scale* entity are the ones that suffer the most significant change. This entity requirements are significantly lower for MBL due to the fact that the

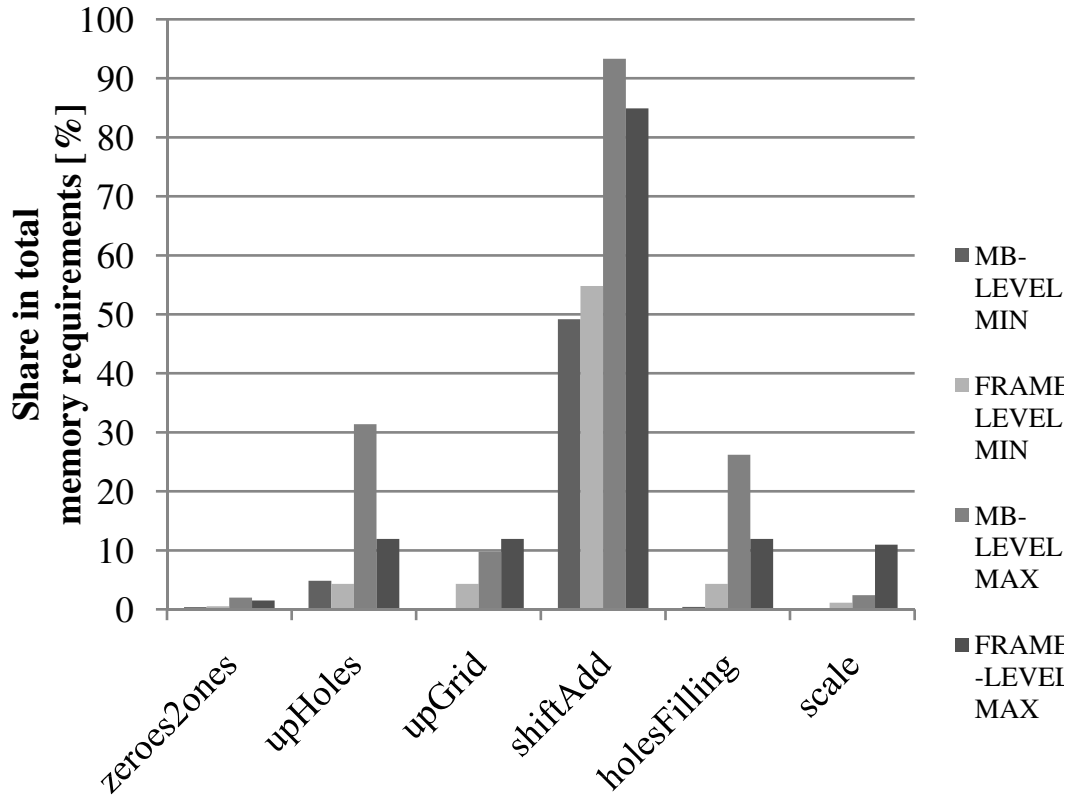


Fig. 4.4 Maximal and minimal percentage share of entities (excluding the adapters) in total memory requirements of the super-resolution kernels.

entity input is a HR MB instead of a HR frame. The increase in memory storage requirements noticed for the *upHoles* and *shitfAdd* is caused by the fact that the difference in size of *holesGrids* and *srGrid* is greater for the MB-level implementation (SA size/MB size instead of frame size/frame size). Apart from increasing the requirements of the aforementioned entities, the aforementioned change lowers the minimal share for the rest of the entities, and the maximal requirements of the *upGrid* entity. Increased share of the *holesFilling* entity is related to additional memory used for data dependency management which is not required in the FL system. This memory size does not vary with SAR, number of RFs or scale, leading to increased share in TMR for configurations with low values of these parameters. The introduction of the interpolation buffer memory (of int_{buffer} type) lowered the maximal and minimal share of the *upGrid* entity, as both the introduced memory and the *upGrid* entity share similar dependency on the SR parameters.

To recap, for tested combinations of parameters values the *frameReconstruction* and *reorderBuffer* dominate the memory requirements for 4x4 MBs. The *shiftAdd* has never been the entity with maximal share for this MB size. For large MBs (16x16) the *reorderBuffer*

never poses maximal requirements, which are noticed for *frameReconstruction* or *shiftAdd*. For medium MB size (8x8) it is possible to find a combination of SR parameters values for which each of the three blocks will have the maximal share in the TMR.

4.4 Evaluation of memory accesses count carried out by the macroblock- and frame-level NUGPA execution flows

In order to compute the count of memory accesses carried out by the MBL and FL SRKs, equations modeling the access counts as a function of the algorithm parameters have been created for each NUGPA step. This task has been carried out in three steps:

(i) determination of static and dynamic parameters that influence the count number,

(ii) identification of steps' memory access patterns, and

(iii) formation of equations modeling these patterns.

4.4.1 Modeling memory accesses

The carried out analysis of the software implementations has shown that for some parts of code the number of accesses does not depend only on the algorithm parameters (presented in Table 3.1) but also on run-time determinable values derived from received ME metrics. These values are used as arguments in the evaluation of conditions that control the implementations' execution flow and the number of carried out memory accesses.

4.4.1.1 Probability model of conditional memory accesses

In order to model the run-time determinable behavior of some of the memory accesses we have used a set of figure representing the probability of the state that directly conditions the execution of these accesses. For the purpose of this study three such figures have been defined, namely, the $SoWgt1_p$, the $subpixMv_p$ and the $holes_p$.

The $SoWgt1_p$ figure represents the probability of the sum of weights of pels computed by the *shiftAdd* step to be greater than '1'. Thus, this figure value defines how many pels undergo the normalization step and thus require one additional read access.

The $holes_p$ figure represents the probability of the value of a pel to be read from the input by the *holesFilling* step to be equal to '0'. This probability determines the percentage

of pels for which the interpolation process is to be carried out resulting in up to 24 additional read accesses.

The $subpixMv_p$ figure models the probability of the motion vector to represent subpixel movement. The extraction of pels from reference structures (this term encapsulates the *holesGrids* and the SAs) during the *shiftAdd* transformation is only carried out for RSs for which the associated mv is of subpixel type. Thus, this probability value determines the fraction of total number of pels (up to RF_{nr}) for which additional accesses are required. Indirectly this figure value impacts the other two figures values. Increase in $subpixMv_p$ probability results in more pels being read and fused. The additional pels either share the projection coordinates with other merged pels (increasing the $SoWgt1_p$), or do not — filling a hole — lowering the $holes_p$ value.

The values of these probability figures can be evaluated directly once the values of the received ME metrics are known. The $subpixMv_p$ is computed by dividing the count of mvs for which at least one scalar component value, after being divided by the value of $precision_{me}$, has a non-zero residual value (represented by $mvWithSubpixMv_{count}$), by the number of received MVs ($mvReceived_{nr}$). The $holes_p$ value is computed by subtracting the count of unique coordinates that have been assigned a new (non-zero) value during the fusion ($pelsWithValgt1_{count}$) from the number of addresses in the address space ($pelsInHrFrame_{nr}$; equal to size of a HR frame) and dividing the outcome by the number of addresses in the address space ($pelsInHrFrame_{nr}$). Similarly the $SoWgt1_p$ is computed by dividing the count of unique coordinates assigned new value formed by fusing two or more pel values ($valOf2orMoreMergedPels_{count}$) by the number of addresses in the address space ($pelsInHrFrame_{nr}$).

In this work, the $subpixMv_p$, $holes_p$, and the $SoWgt1_p$ values for quantification of memory access counts have been evaluated empirically by executing the software for multiple combinations of algorithm parameter values and computing the figures values as described above using (4.3), (4.4), and (4.5), respectively.

$$subpixMv_p = \frac{mvWithSubpixMv_{count}}{mvReceived_{nr}} \tag{4.3}$$

$$holes_p = \frac{pelsInHrFr_{nr} - pelsWithValgt1_{count}}{pelsInHrFr_{nr}} \tag{4.4}$$

$$SoWgt1_p = \frac{valOf2orMoreMergedPels_{count}}{pelsInHrFr_{nr}} \tag{4.5}$$

4.4.1.2 Modeling memory traffic of the frame- and MB-level execution flows

Once the figures modeling conditional accesses have been defined, an analysis of memory access patterns of each of the steps of the generalized NUGPA has been carried out. The presented study models the behavior of the steps for the system with fully loaded SFW (system with number of reference frames contained in the SFW = RF_{nr}; this state — by far — predominates throughout the execution time for sequences composed of more than RF_{nr} frames) and does not take into account buffering and data reuse schemes deployed by the FL and MBL implementations. In this work we consider that the steps carrying out the synchronization and data sending as not intrinsic parts of the SRK. Thus, these steps memory access patterns are not analyzed.

The *zeroes2ones* step reads the received pels from memories, applies the *zeroes2ones* transformation and writes the data in corresponding LR memories. Thus, this steps carries out one read and one write access per each received pel.

The outcome of the *zeroes2ones* step is accessed by the grids creation steps. Both of these steps carry out the *up-holes* transformation. The difference is that the *upGrid* step (construction of a *srGrid*) accesses memory representing the LR FR/MB and the *upHoles* step (construction of *holesGrids*) accesses memories to write the LR RS. The outcome of *up-holes* transformation is a HR representation of the LR input. Thus, the step requires one read and $precision_{me}^{2}$ write accesses per every input pel.

Nevertheless, if the block matching ME is carried out in accord with the order appropriate for the FL flow, then the data received by the SRK from motion estimation have to be reordered to allow correct processing by the MBL implementation. The tasks of reordering the hyperdata is carried out by the *reoderBuffer* step, which receives, reorders and writes the data in memory (of $reorder_{buffer}$ type). Then, for each processed MB, the required hyperdata are read from the memory and passed onto the *shiftAdd* step where they are stored local structures. For investigated implementations one meta data element in RB encapsulates four values, meaning that the number of accesses for transferring one RB entry is 4. Thus, for each MB being processed, the requirements for the MBL *reoderBuffer* step are of four read and four write accesses per each RF contained in the SFW.

Once the meta data are available the weights are computed and stored in registers for immediate use, thus this step does not contribute to the *Total Memory Access Count* (TMAC).

The *shiftAdd*, in order to extract the required pels, accesses the HR representations of the RS (*holesGrids*), reads their content and writes it in local memory. This action is repeated up to RF_{nr} number of times per each processed MB/FR. Then, the *srGrid* is being accessed for reading. For each pel read from the *srGrid* up to RF_{nr} local copies of *holesGrids* are

accessed for reading of the reference pel. The *holesGrids* read accesses performed by the *shiftAdd* step are conditional, that is, the pels are read only if the hyperdata associated with the MB being processed meet some conditions. In the case of our implementation, the accesses are carried out only if the mv value corresponds to a sub-pixel motion —this access probability is modeled by the $subpixMv_p$. Thus, the exact count of memory accesses depends on the probability of the motion vector being a subpixel one.

All pels read in the extraction process are projected onto the *srGrid* and fused —their values are multiplied by their weights and summed. Sums of weights are also computed. The resulting sums of values of pels and weights are written to local memories. The weights sums are read coordinate-by-coordinate in the normalization step that takes place after the *shiftAdd* transformation is completed. If the weight value is greater than '1' (probability modeled by $SoWgt1_p$) the corresponding sum of pels is read and the new value for that coordinate is computed by dividing the value of the sum of pels by its corresponding sum of weights. After computation the new value is stored in the output memory of the *shiftAdd* step.

To recap, per each processed pel of the *srGrid* the *shiftAdd* step requires: (i) $RF_{nr}+1$ of non-conditional reads and writes and up to RF_{nr} conditional reads to extract the pels, (ii) two write accesses following the fusion, (iii) one non-conditional read and one non-conditional write and one conditional read and one conditional write to normalize computed pel value.

The *holesFilling* step is performed in a pel-by-pel manner. This step starts off by accessing the input memory and reading a pel value. If the read value is greater than zero it is written into the output memory and processing of the subsequent pel is started. Otherwise, this step accesses the input memories and reads up to 24 values required for computation of the new value at the coordinate. The computed new value is written to the output memory. Thus, this step requires one non-conditional read and write and up to 24 conditional reads per each pel read from the input. The number of conditional accesses depends on the number of coordinates with value equal to zero. The probability of these conditional accesses is modeled by the $holes_p$ figure.

The outcome of the *holesFilling* step is read by the *scale* step. This step, given a HR input, produces its SR representation that has the size that meets the expected output size specified by the scale parameter value. Thus, statistically the step performs one read and one write per $precision_{me}^2/scale^2$ pels contained in the input memory.

Once the study on memory access patterns and their dependency on the conditional flow have been carried out, the equations describing the memory access counts of each step —as a function of the number of input pels (in_{pels}) the step has to process in one

TABLE 4.6 *Equations modeling the count of accesses to memories carried out by the algorithm steps defined for the analyzed software implementations (in_{nr} represents the number of pels passed as the input of every step).*

Entity	Read access count	Write access count
zeroes2ones	in_{nr}	in_{nr}
upHoles	in_{nr}	$in_{nr} * precision_{me}^2$
upGrid	in_{nr}	$in_{nr} * precision_{me}^2$
shiftAdd	$in_{nr} * (2 + Rf_{nr} + SoWgt1_p)$ $+Rf_{nr} * subpixMv_p$	$in_{nr} * (4 + Rf_{nr})$
holesFilling	$in_{nr} * (1 + holes_p * 24)$	in_{nr}
scale	$in_{nr} * scale_{sr}^2 / precision_{me}^2$	$in_{nr} * scale_{sr}^2 / precision_{me}^2$
reorderBuffer[ab]	$RF_{nr} * 4$	$RF_{nr} * 4$
frameReconstruction[b]	in_{nr}	in_{nr}

[a] access per each MB being processed.
[b] only present in the MBL flow.

iteration— have been created. The resulting equations are presented in Table 4.6, where in_{pels} represents the number of pels in either a MB/FR or SA/FR (for *upHoles*) or a sum of both (for *zeroes2ones*). The equations model the behavior of the steps for the system with fully loaded SFW in saturation state and do not take into account buffering and data reuse schemes that could be deployed by FL and MBL implementations. Accesses to registers are not taken into account for the computation of memory access count required by the processing steps.

4.4.1.3 Equations modeling memory traffic

The equations presented in Table 4.6 do not take into account the differences between the FL and MBL flows (except for the introduction of the entities that encapsulate MB-to-FL adapters). From these equations one can see that the MBL flow memory counts are expected to be (not too much) higher due to the introduction of additional steps and memories required for seamless substitution of the FL SRK.

Nevertheless, only once the differences between the implementations of the flows are taken into account the equations modeling a realistic memory access counts can be constructed. The most important of these differences for the memory access counts are the buffering schemes and data reuse options put in place. The creation of refined equations for both of the flows is the topic of this subsection.

The main characteristic of the FL flow is that it uses a RS that comprise all pels of a frame. As the implementations instantiate local memories for all RS of SFW, for FL implementation this means that once a HR RF is written into local memory it remains readily available till being discarded from the SFW. The need to re-create a once discarded RS never arises as the SFW slides only in one direction and therefore once a frame is discarded and falls out of the scope of the SFW it is not referenced again.

On the contrary, the local HR RS used by the MBL implementation contains only the pels that belong to the execution context of by the MB that is currently being processed. This leads to a significant reduction in memory size requirements, but as the MBs are processed in raster scan order, it also results in the necessity of re-creating from scratch the same (once created and discarded) *holesGrids* for MBs of the subsequent frames. The re-creation of *holesGrids* takes place even if the MBs with the same location within the frame reference the same RF (thus, the same RS).

The investigated MBL implementation does not implement any kind of reference structure pels buffering. Thus, for each processed MB all pels of the RS are received, undergo full *up-holes* transformation and are written to local *shiftAdd* memories. This impacts the steps of the RS/*holesGrids* preparation flow: the *zeroes2ones*, the *upHoles* and the extraction stage of the *shiftAdd* transformation. Instead of preprocessing (on average) one RS of HR FR type per processed FR (as in FL), the MBL flow needs to re-create and copy into local memory RSs of HR SA type $RF_{nr} * MB_{nr}$ times for processing of each frame.

The number of pels contained in the square-shaped neighborhood that need to be read from memory by the *holesFilling* step ranges from 8 to 24 and its statistical distribution depends, among other parameters, on the frame size. For this study, we overestimate the high ceiling value as 24 accesses required per computation of the new value of one hole, which resembles an overestimation of the worst case scenario (since the computation of new values of holes belonging to the frame borders always requires less than 24 accesses).

The data dependencies management necessary for correct execution of the MBL flow requires the introduction of the interpolation buffer (int_{buffer}). This memory further increases the total memory access count. The number of the additional accesses depends on the interpolation radius (int_{window}) and HR macro-block size. The count of additional writes is modeled as $2 * int_{window} * MB_{width} * precision_{me} - int_{window}^2$ which for constant value of int_{window} $(= 2)$ is equal to $4 * (MB_{hr} - 1)$. The introduction of the interpolation buffer does not impact much the total memory access counts.

The equations modeling the memory access counts required by NUGPA steps that take into account buffering schemes deployed by the FL and MBL implementations, the conse-

TABLE 4.7 *Equations modeling the count of accesses of the identified algorithm steps required for processing of one frame.*

Entity \ Accesses	Read	Write

Implementation	Frame-level	
zeroes2ones	FR_{lr}	$2 * FR_{lr}$
upHoles	FR_{lr}	FR_{hr}
upGrid	FR_{lr}	FR_{hr}
shiftAdd	$3 * FR_{hr} + FR_{hr} * SoWgt1_p$ $+FR_{hr} * (RF_{nr} * subpixMv_p$	$4 * FR_{hr}$
holesFilling	$FR_{hr} * (1 + holes_p * 24)$	FR_{hr}
scale	FR_{sr}	FR_{sr}
reorderBuffer	—	—
frameReconstruction	—	—

Implementation	MB-level	
zeroes2ones	$SA_{lr} * RF_{nr} * MB_{nr} + FR_{lr}$	$SA_{lr} * RF_{nr} * MB_{nr} + FR_{lr}$
upHoles	$SA_{lr} * RF_{nr} * MB_{nr}$	$SA_{hr} * RF_{nr} * MB_{nr}$
upGrid	FR_{lr}	FR_{hr}
shiftAdd	$SA_{hr} * RF_{nr} * MB_{nr}$ $+FR_{hr} * RF_{nr} * subpixMv_p$ $+FR_{hr} * (2 + SoWgt1_p)$	$SA_{hr} * RF_{nr} * MB_{nr}$ $+3 * FR_{hr}$
holesFilling	$FR_{hr} * (1 + holes_p * 24)$	$FR_{hr} + 4 * MB_{nr} * MB_{hr}$ $-4 * MB_{nr}$
scale	FR_{sr}	FR_{sr}
reorderBuffer	$4 * RF_{nr} * MB_{nr}$	$4 * RF_{nr} * MB_{nr}$
frameReconstruction	FR_{sr}	FR_{sr}

quences of smaller (that a whole frame) reference structures and workload processing in a MB-by-MB manner are presented in Table 4.7. As before, these equations are defined for the system in saturation state (system with fully loaded SFW). Thus, due to the reuse of RFs for the FL flow on average only one HR RS *holesGrid* needs to be constructed and copied into local *shiftAdd* memories. For MBL all SAs have to be received and all *holesGrid* have to be re-constructed and copied into local *shiftAdd* memories for every processed MBs.

In summary, due to greatly increased number of re-transferred reference pels (and the need to process them again), without considering the implementation specific buffering schemes aimed at minimizing the number of pels being (re)transferred (plenty of room for

TABLE 4.8 *Noticed minimal, maximal and average values of probability figures.*

Sequence	$holes_p$			$subpixMv_p$			$SoWgt1_p$		
	MIN	MAX	AVG	MIN	MAX	AVG	MIN	MAX	AVG
foreman	0.59	0.87	0.76	0.48	0.64	0.55	0.01	0.64	0.22
mobile	0.57	0.87	0.75	0.5	0.63	0.55	0.01	0.64	0.22
paris	0.75	0.91	0.85	0.16	0.33	0.24	0.01	0.33	0.11

optimization), the TMAC of the MBL flow is expected to skyrocket with the increase of SAR and RF_{nr}.

4.4.2 Quantitative evaluation of memory traffic

The equations used for modeling of memory access counts presented in Table 4.7 use three figures to represent probabilities of conditional accesses carried out by the *shiftAdd* and *holesFilling* steps. Thus, the following step of the study was the evaluation of the values of these probabilities.

Once the probability of these accesses has been determined the aforementioned equations have been used to determine the total memory access counts required by the FL and MBL implementations for the same set of 96 combinations of algorithm parameter values. The results of both studies are presented in this section.

4.4.2.1 Evaluation of values of the probability parameters

In order to determine the values of the $holes_p$, $subpixMv_p$, $SoWgt1_p$ several simulations have been carried out for the aforementioned set of 96 combinations of algorithm parameter values for three test sequences, namely, the *foreman*, *mobile* and *paris* sequences. For this purpose 96 simulation runs have been carried out. The tested combinations set has been created by sweeping through the MB_{width} (4, 8 and 16), SAR (2, 4, 8 and 16), number of RFs (2, 4, 8 and 16), and the scale (2 and 4) parameter values. This set is the same as the one used for the study and optimization of memory requirements. The defined probability figures characterize the algorithm rather than its implementations flows. Thereby, simulations have been carried out only for the frame-level flow and have been considered valid for the MBL flow.

The simulations produced 96 values of the $holes_p$, $subpixMv_p$, $SoWgt1_p$ figures. The minimal, maximal and average values observed for the defined probability figures are presented in Table 4.8. These values have been analyzed in order to evaluate the relations be-

TABLE 4.9 *Computed average values of probability figures for groups sharing algorithm parameter value.*

Probability figure	Reference frames				Search area radius				MB size			∇		
	2	4	8	16	2	4	8	16	4	8	16	RF	SAR	MB
Foreman sequence														
$holes_p$	0.86	0.81	0.73	**0.62**	**0.75**	0.76	0.76	0.76	0.76	**0.75**	0.76	**0.079**	0.004	0.01
$subpixMv_p$	0.54	0.54	0.55	**0.57**	**0.58**	0.55	0.54	0.53	0.52	**0.57**	0.56	0.009	0.018	**0.036**
$SoWgt1_p$	0.01	0.07	0.23	**0.57**	**0.24**	0.22	0.21	0.21	0.2	**0.23**	0.22	**0.186**	0.01	0.017
Mobile sequence														
$holes_p$	0.86	0.81	0.73	**0.61**	**0.75**	0.75	0.76	0.76	0.77	0.75	**0.74**	**0.84**	0.004	0.018
$subpixMv_p$	0.53	0.55	0.56	**0.58**	**0.57**	0.56	0.55	0.54	0.53	**0.58**	0.56	0.015	0.008	**0.035**
$SoWgt1_p$	0.02	0.07	0.23	**0.58**	**0.23**	0.23	0.22	0.22	0.22	**0.24**	0.22	**0.186**	0.005	0.016
Paris sequence														
$holes_p$	0.90	0.88	0.84	**0.78**	**0.85**	0.85	0.85	0.85	**0.84**	0.85	0.86	**0.042**	0.001	0.012
$subpixMv_p$	0.2	0.23	0.26	**0.29**	**0.25**	0.25	0.24	0.24	**0.28**	0.25	0.21	0.029	0.003	**0.036**
$SoWgt1_p$	0.00	0.03	0.11	**0.29**	**0.11**	0.11	0.1	0.1	**0.12**	0.11	0.09	**0.186**	0.002	0.012

tween the algorithm parameters and the defined probability figures. In order to determine the parameter whose changes influences the most the defined probability figures values we have defined the ∇ figure that represents the dynamics of the correlation between the changes of the SR parameter values and probability figures values. The ∇ figure values have been computed as follows. First the values obtained from simulations have been grouped based on the value of a chosen algorithm parameter. Then, the average values of probability figures have been computed for each group. Following, absolute differences of the subsequent groups' averages have been computed. The ∇ figure has been assigned the value of the average of these absolute differences. The computed average values of probability and the ∇ figures are presented in Table 4.9, with values that benefit the SR image quality in bold.

The Table 4.9 does not present data for different values of scale parameter as for different scale values we have obtained the same probability values. This is explained by the fact that the *scale* parameter does not influence the control flow of the *shitftAdd* and *holesFilling* steps.

The results highlighted the RF_{nr} parameter as the one that has the greatest impact on the $holes_p$ and $SoWgt1_p$ probabilities. The increase in RF_{nr} leads to significant and progressive decrease (which is good in this case) in the count of *holesGrid* coordinates that are not assigned a value during the fusion stage of the *shiftAdd* step. The dynamic of the correlation changes between RF_{nr} and the aforementioned figures is by far greater than for other investigated parameters. The average values noticed for groups created based on RF_{nr} values are

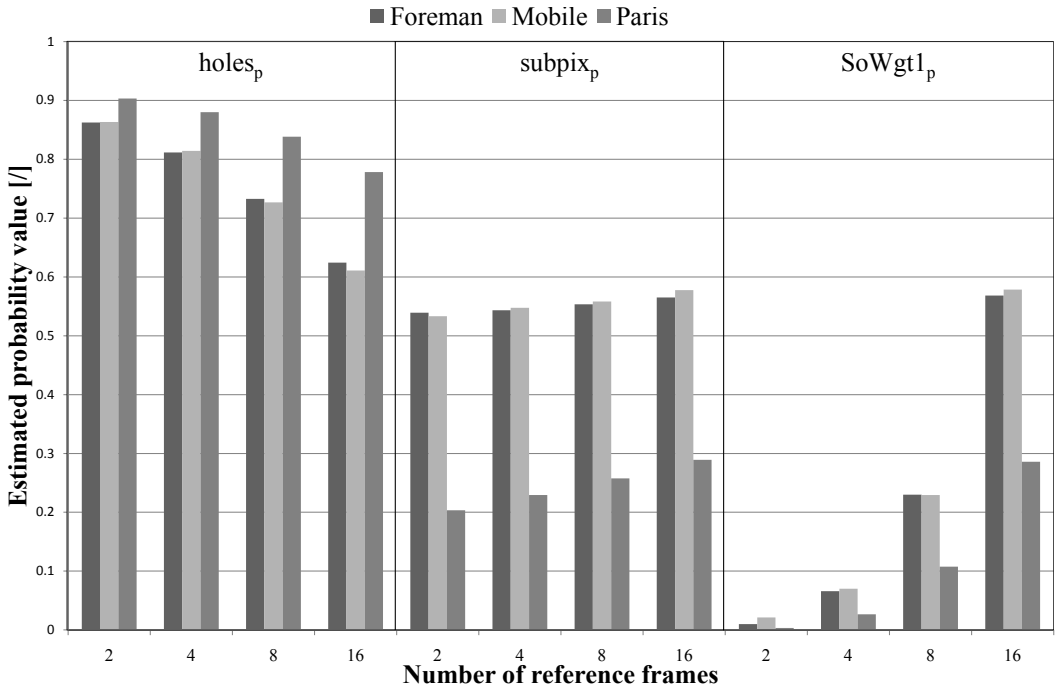


Fig. 4.5 Observed values of probability figures as a function of the number of reference frames.

presented in Fig. 4.5.

The impact of the RF_{nr} on the $subpixMv_p$ is also significant, but, for carried out simulations, the value of this figure has been more susceptible to changes of the MB size. Nevertheless, the influence of the change in the MB size on the change of the $subpixMv_p$ is not easily determined as the effect of the change from one MB size to other results in different direction of changes in $subpixMv_p$ for different test sequences used.

The dynamic of the impact of the SAR value change has been quantified as of the least importance from the three investigated parameters. Even major changes in SAR value did not influence significantly the average values of defined probability figures. This is not a surprise as the size of the SA does not impact the number of pels being merged or the probability of receiving a subpixel mv.

4.4.2.2 Evaluation of memory traffic

Having determined the values of the $holes_p$, $subpixMv_p$, $SoWgt1_p$, TMACs of the FL and MBL have been computed for the same 96 combinations of the algorithm parameter values used in the study on probabilities using the equations presented in Table 4.7.

Computed minimal and maximal values of the TMACs for foreman, mobile and paris sequences are presented in Table 4.10. The results show that for FL the maximal TMACs

TABLE 4.10 *Computed minimal and maximal memory access counts of frame- and MB-level super-resolution kernels.*

Flow	Frame-level		MB-level		MAX/MIN	
Sequence	Min	Max	Min	Max	FL	MBL
Foreman	13561068	15831891	16619173	1689979357	1.17	101.7
Mobile	13589454	15568313	16635409	1690506449	1.16	101.6
Paris	13690830	15028993	16777890	1689999505	1.1	100.7

have been of no more than 1.17 times higher than the minimal TMACs. For MBL the maximal TMACs have been more than 100 times higher than the minimal ones. The remarkably small difference between the minimal and maximal TMACs of the FL implementation is caused by the moderate reuse of RS data. Due to the fact that the RS represents a whole frame, once a RS is loaded all possible reference pels are readily available from memory and do not need to be re-transmitted for subsequent frames that reference this RS. Thus, when working with fully loaded SFW (the state of the systems considered in this study) on average the system has to receive and construct only one RS per processed frame, independently of the maximal number of RFs in the SFW. Therefore due to extensive RSs reuse the increase of the TMAC caused by the RF_{nr} is significantly reduced (still, the RF_{nr} impacts the number of pels projected). Moreover, the influence of the MB size and SAR on the FL TMAC is not direct as these parameters are not referenced by the equations used for modeling of TMACs (Table 4.7). Rather, the values of these parameters influence the value of the probability figures that impact both flows in the same manner.

In case of the MBL flow reference data reuse is limited and troublesome in implementation as the RS only contains the part of the frame that represents the search area of the MB that is being processed. The investigated version of MBL did not implement any data buffering or schemes aimed at data reuse of the RS pels. Hence, for subsequent frames, the RS pels have to be re-transmitted in order to re-construct the *holesGrid* referenced by the currently processed frame MBs resulting in significant changes in TMAC for different algorithm parameter values. Moreover, as the SA of adjacent MBs overlap, the total sum of transmitted SA/RS pels is (much) greater than the number of pels in a RF. For a 4x4 MB and a SAR of 16, the number of pels that are contained within all possible SAs, when compared with equivalent FL RS, is increased by the factor of 81 times. When combined with RF_{nr} of 16 RS, it results in a total increase by the factor of 1296.

In order to quantify the impact of the algorithm parameter values on the FL and MBL TMACs difference the increase noticed in TMACs associated with the change from FL to

MBL processing has been investigated. The increase has been quantified by dividing the computed MBL TMAC by TMAC computed for an equivalent FL implementation. The obtained increase values computed for all the tested combinations (with $scale = 2$) and sequences are presented in Fig. 4.6. For all the tested sequences the TMACs follow the same trend resulting in the observed minimal TMAC increase by a factor of 1.22, 1.23 and 1.22, and the maximal TMAC increase by a factor of 117, 113, and 117, for the foreman, mobile, and paris sequence, respectively. The choice of the MB size is of most importance. The lowest increase (minimal of 1.22, maximal of 14) has been noticed for the highest investigated value of MB width (16). The increase factors noticed for all tested combinations (with $scale = 2$) for MB size of 16x16 are presented in Fig. 4.7. The increase noticed for switching from MB size of 16x16 to MB size of 8x8 and 4x4 is presented in Fig. 4.8. When compared with the TMACs of 16x16 MB the TMACs of 8x8 and 4x4 MBs are increased by a factor of up to 2.7, and up to 8.7, respectively. The moderate differences noted for different MB sizes are caused by the increased number of SA pels that have to be transmitted for smaller MB sizes. The relation between the SAR and RF values and the TMAC can be approximated as linear for all MB sizes. The change from FL to MBL flow always has resulted in significant increase of TMAC.

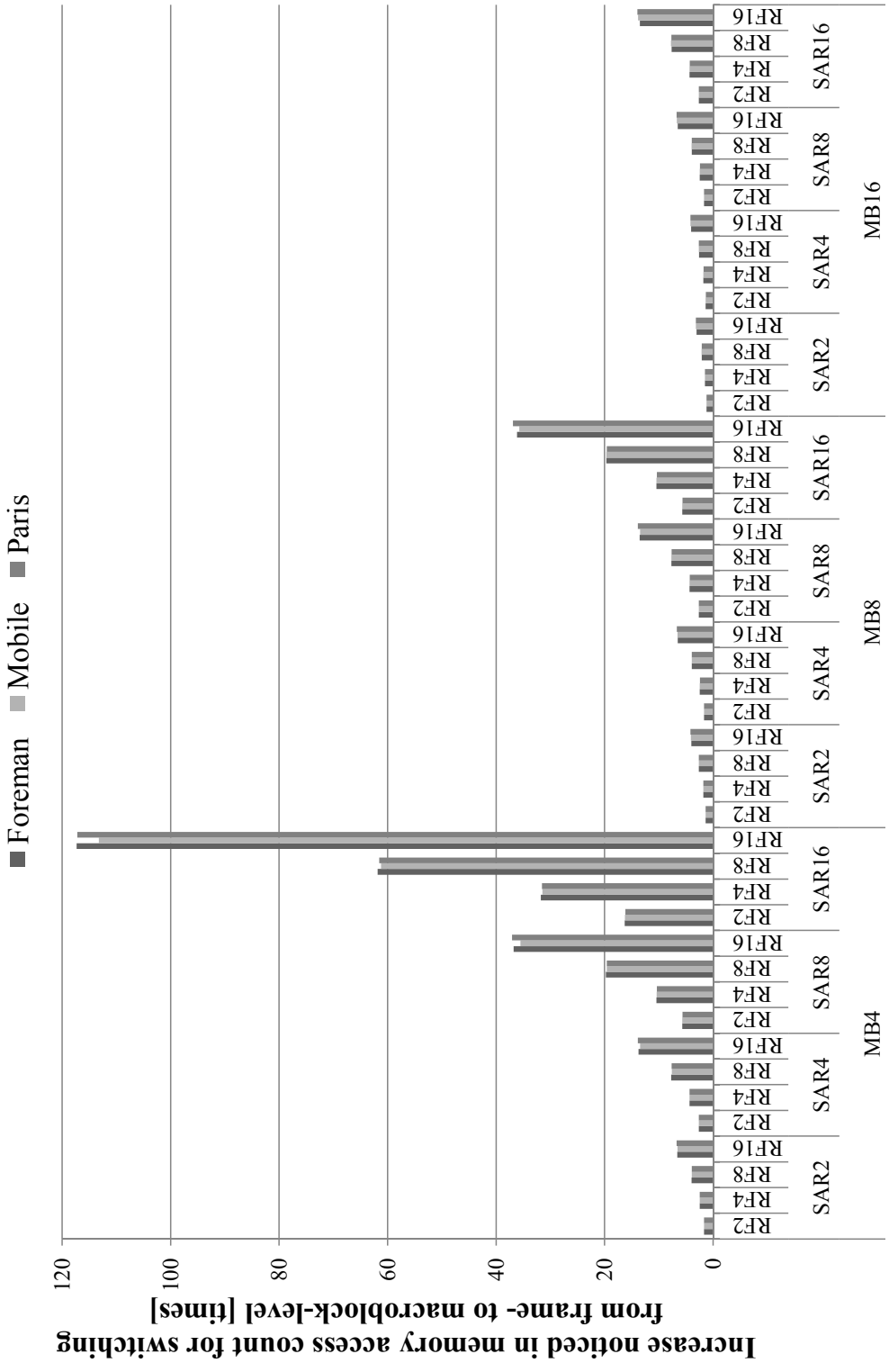


Fig. 4.6 Increase noticed in total memory access count when comparing macroblock-level NUGPA SRKs with equivalent frame-level implementations. MB4, MB8, and MB16 stand for $MB_{width} = 4, 8$ and 16, respectively.

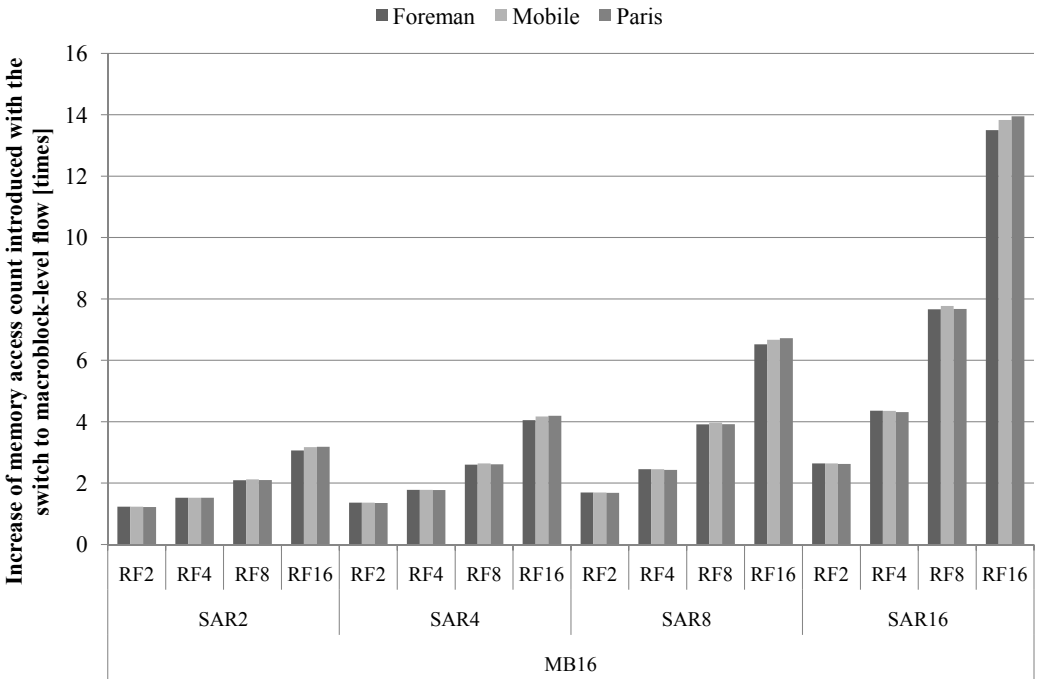


Fig. 4.7 Increase in TMAC observed for the change from FL SRK with MB size of 16x16 (labeled as MB16) to equivalent MBL SRK implementation.

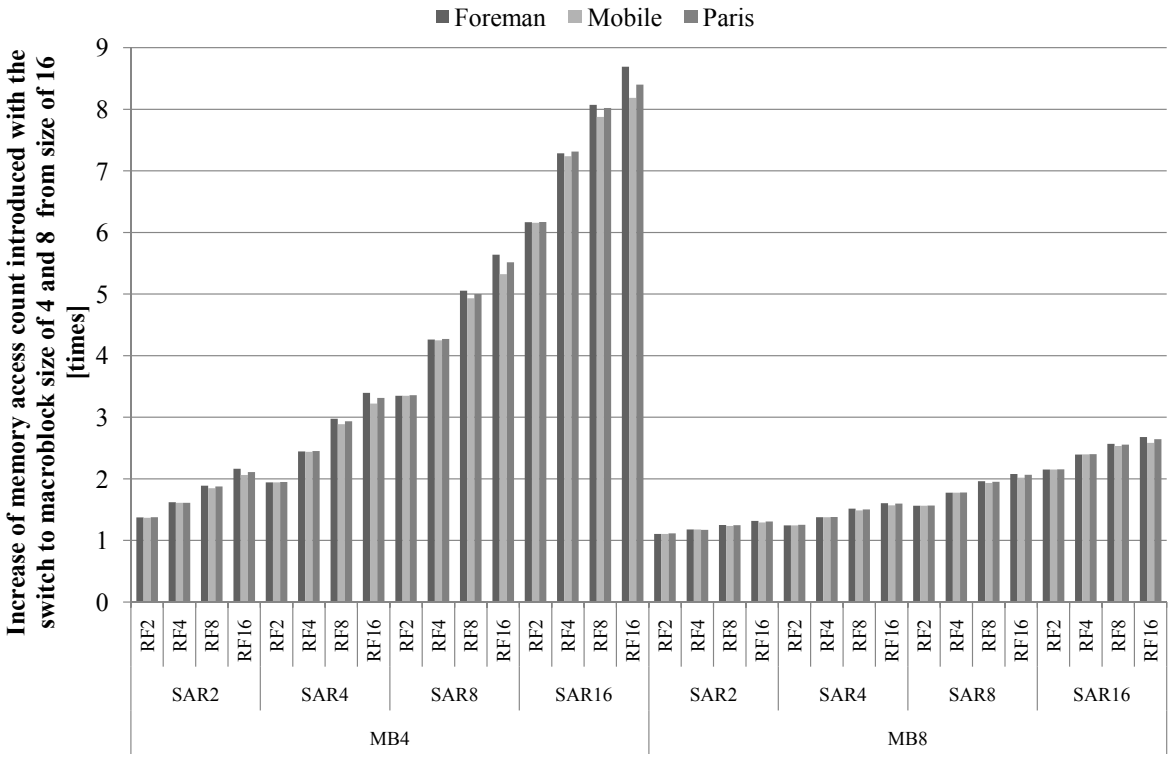


Fig. 4.8 Increase in TMAC observed for the change from MBL SRK with MB size of 16 to equivalent MBL SRK implementation with MB size of 4x4 and 8x8 (labeled as MB4 and MB8, respectively).

TABLE 4.11 *Computed memory access counts of defined entities of the analyzed software implementations.*

Flow	Frame-level			MB-level		
Share / Entity	Min [%]	Max [%]	Avg [%]	Min [%]	Max [%]	Avg [%]
zeroes2ones	0.48	0.56	0.53	1.16	3.89	2.96
upHoles	2.72	3.18	3.03	7.48	33.06	24.33
upGrid	2.72	3.18	3.03	0.03	2.59	0.87
shiftAdd	20.64	48.61	30.17	26.08	62.56	51.93
holesFilling	41.95	70.17	59.72	0.42	57.77	17.85
scale	1.33	5.72	3.52	0.01	4.55	1.01
reorderBuffer	—	—	—	0.01	0.21	0.04
frameReconstruction	—	—	—	0.01	4.55	1.01

4.4.2.3 Share of execution flows' entities in total memory access count

Computation of memory access count has been carried out for every entity making up the SRK and for each of the 96 test combinations. The values of these computations have been used in order to determine the impact of each of the entities on the TMAC of the SRK. The results of the study — the minimal, maximal and average share of the count carried out by the entities in the TMAC —are presented in Table 4.11 and visualized in Fig. 4.9. Presented data show that the total count of accesses carried out by the SRK is mainly influenced by the *shiftAdd*, the *holesFilling*, and, in case of MBL implementation also by the *upHoles*, entities.

The frame-level memory traffic is mostly generated by the accesses carried out by the *holesFilling* entity. As shown in Table 4.11 this entity has been the one with the highest minimal (42%), maximal (70%) and average (59%) share. The average share of this entity is almost two times higher than the average share of the *shiftAdd* entity (30%, second most significant share in TMAC). The third most influential, with average share of 3.52%, is the *scale* entity.

Again, the change from frame-level to macroblock-level flow significantly changes the distribution of entities share in TMAC. The impact of the *holesFilling* entity is significantly lowered. This entity with the minimal, maximal and average share of 0.4%, 58% and 18%, respectively, becomes the third most important contributor (based on average share), after the *shiftAdd* and *upHoles* entities. The *shiftAdd* entity is by far the one that generates the most traffic with minimal, maximal and average share of 26%, 63% and 52%, respectively. The most significant changes have been noticed for the *upHoles* and *zeroes2ones* entities.

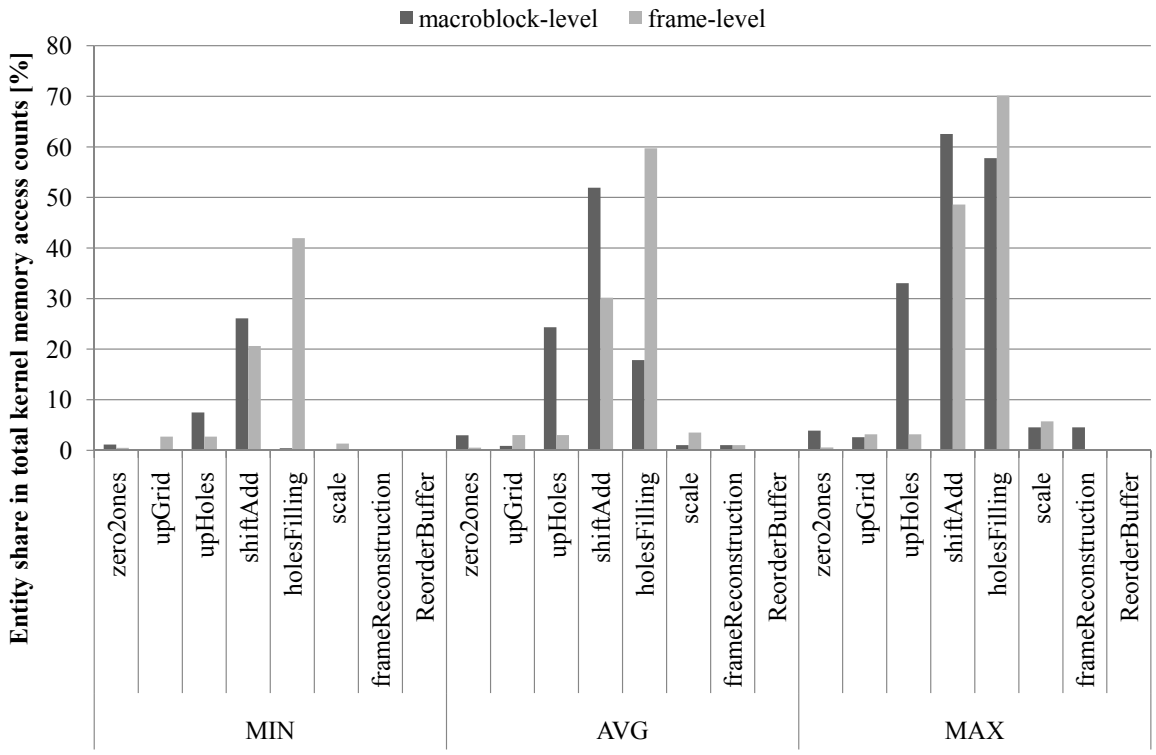


Fig. 4.9 Share of defined entities in the total memory access count of the frame- and macroblock-level NUGPA super-resolution kernels.

The former entity presents an increase in the average share by a factor of 8, and with minimal, maximal and average share of 7.5%, 33% and 24%, it has become the entity with the second highest share in TMAC. The latter entity average share has increased almost 6 times reaching 2.96%.

The main cause of the increased share of the *shifAdd*, *upHoles* and *zeroes2ones* entities of the macroblock-level flow is the necessity of re-transmitting, re-creating and copying of the *holesGrids* for every MB being processed. The amount of data that undergoes this re-processing skyrockets with the increase in values of SAR and RF_{nr}. For 4x4 MB, QCIF frame, SAR=16 and RF_{nr}=16 the count of generated memory accesses observed for the *zeroes2ones*, *upHoles*, *shifAdd* is, respectively, 865, 1296, 212 times the one generated by a FL equivalent (worst case). However, the increase of SAR and RF_{nr} values lowers the probability of the necessity of interpolation, lowering the count of accesses performed by the *holesFilling*. An increase in MB size improves the figure of merit across all test combinations. These are examples of tradeoffs to be made at design time.

The average share of the memory accesses of introduced adapter entities on the TMAC is of 0.04% and 1.01%, respectively for the *reorderBuffer* and *frameReconstruction*, being

almost of no significance. The writes to the int_{buffer} buffer required for data dependency resolution increase the count of accesses carried out by the *holesFilling* entity by no more than 2%.

4.5 Evaluation of the trade-offs of the proposed execution flow

In this section we compare and evaluate the trade-offs of switching from FL to MBL flow by comparing the factors of reduction in memory occupancy and the increase in memory access counts associated with the aforementioned change.

The results of the study presented in Section 4.3.2 have shown that, for the QCIF frame format, the change from the frame-level to the MB-level execution scheme can lead to of a TMAC by a factor between 6.8 and 40, depending on the NUGP algorithm parameter values. The minimal and maximal TMR computed for the MBL flow correspond to 3.7% and 11.5% of memory occupancy of equivalent (having the same values of algorithm parameters) frame-level version. The results have shown that both flows' memory requirements are mostly influenced by the number of reference frames. For the MB-level flow also the size of a macroblock is significant as it determines the requirements of the *adapter* entities.

The results presented in Section 4.4.2 have shown that, for the QCIF frame format, changing from frame-level to MB-level execution flow, without implementing ad-hoc buffering mechanisms, can lead to an increase in total memory access count by a factor between 1.22 and 117. Notice that not implementing ad-hoc buffering mechanisms is a worst case scenario for memory traffic, and hence a stress test, in particular for the macroblock-level flow.

The results shows that for the tested combinations the algorithm parameter that influences the most the memory access count of the frame-level flow is the number of reference frames used. For the macroblock-level flow all of the NUGPA parameters have a major impact on the memory access count. Nevertheless, the most significant impact was observed for changes in MB size value, for which the highest noticed factor of maximal count increase versus equivalent frame-level count ranges from 14 (for MB size of 16x16) to 117 (MB size of 4x4). For the MB-level flow, the noticed maximal count could be as much as 102 times greater than the observed minimal count. For the frame-level flow, the observed maximal was only 1.21 times higher than the noticed minimal. Observed frame-level flow results highlight the efficiency and importance of the deployed memory buffering policy.

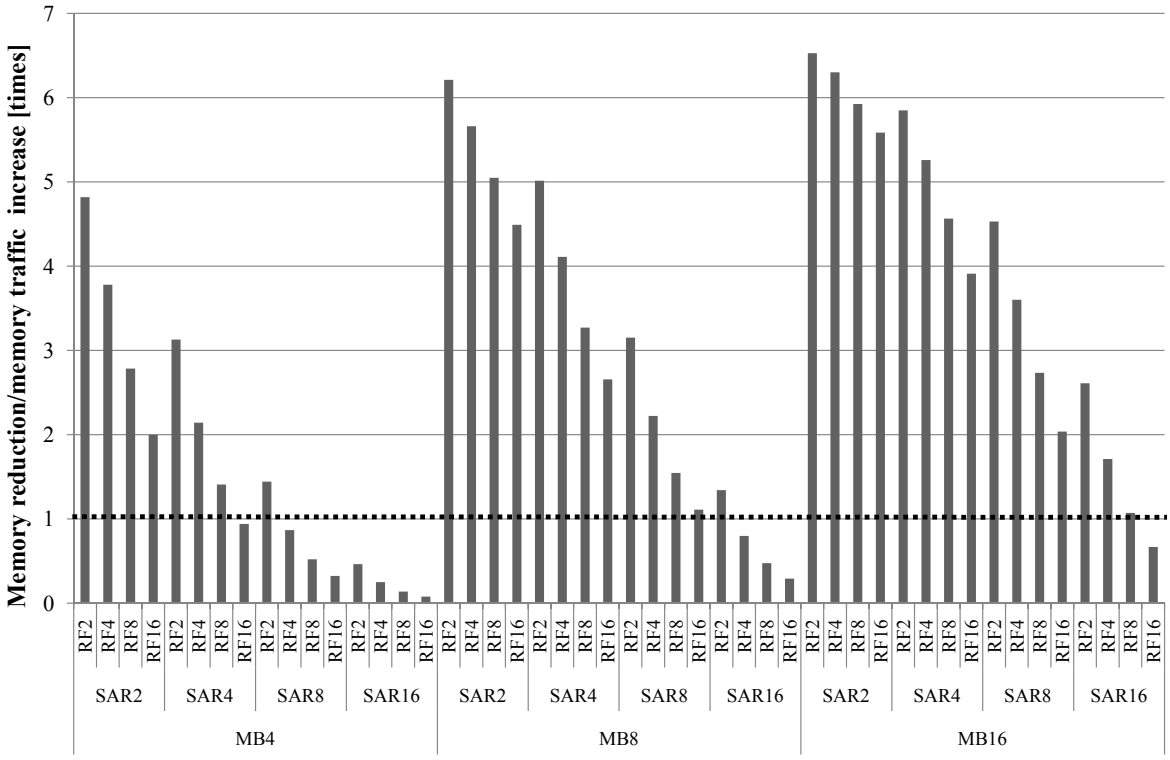


Fig. 4.10 The factor of reduction in memory occupancy versus the factor of increase in memory traffic (for scale=4). MB4, MB8 and MB16 stand for $MB_{width} = 4, 8$ *and* 16*, respectively.*

As the sets of investigated algorithm parameters that has been used in both studies has been the same. Thus, it is possible to confront both factors values and analyze the relation of memory reduction/traffic increase as a function of the algorithm parameter values. This relation, for the results of estimation of memory traffic for the worst case observed and scale parameter set to 4, is presented in Fig. 4.10. The diagram for scale value equal to 2 is similar, with the difference that the memory occupancy reduction versus traffic increase factor (a suitable figure of merit, hereinafter $TMR/TMAC_{ratio}$) is about 2–3 times higher. Scale = 4 is the worst case for this factor.

The computed increase of memory access count associated with the change from frame-level to macroblock-level NUGP algorithm flow is significant. The range of computed values of the factor of increase in memory access count (of 1.22 to 117) is wider than the range of computed values of the factor of reduction observed in total memory size requirements (of 6.8 to 40). Nevertheless, an analysis of the Fig. 4.10 shows that, in spite of the increase in total memory traffic, for 76 out of 96 tested combinations the observed factor of memory occupancy reduction has been greater than the observed factor of increase in memory access count.

Out of the 20 combinations for which the $TMR/TMAC_{ratio}$ is lower than '1', two have been observed for MB size of 16x16, five for MB size of 8x8 and 13 for MB size equal to 4x4. The set of combinations for which the increase in memory traffic is greater than the reduction noticed in memory requirements for the largest MB size only contains combinations with the search area radius (SAR) and reference frame number (RF) values higher than 8. For MB size of 8x8, only combinations with SAR value greater than 8 results in the $TMR/TMAC_{ratio}$ value being lower than '1'. For the smallest evaluated MB size (equal to 4) for all combinations with SAR value greater than 8, almost all with SAR value of 8 and one combination with SAR equal to 4 the increase in memory traffic is greater than noticed factor of memory reduction. It is clear that the advantages brought by finer granularity of the MB-level SRIR flow are limited for cases of the smallest MB size and the highest SAR values. Thus, optimizations of data reuse and buffering policies should focus on lowering the memory traffic particularly for these combinations of SRIR parameters' values.

4.6 Optimization of memory traffic

The study on entities' share in total memory occupancy and memory traffic has identified the *zeroes2ones*, *upHoles/upGrid* and *shiftAdd* entities as the ones that have the greatest influence on the $TMR/TMAC_{ratio}$ precisely for the values of parameters for which the contention in traffic has been observed. The analysis has shown that the observed skyrocketing increase in memory traffic is generated by the *zeroes2ones*, *upHoles* and *shiftAdd* entities. Most of this traffic is caused by the lack of implementation of buffering mechanisms as per our worst case macroblock-level scenario. There is large room for optimization and a high possibility that the impact of optimizations on memory traffic will be significant. Hence, these entities have been chosen to be analyzed in details in order to optimize their implementation and execution and come up with a buffering policy that would significantly reduce the memory traffic generated by the entities.

4.6.1 Optimization technique

As stated before, the *zeroes2ones*, *upGrid*, *upHoles* and *shiftAdd* entities have been identified as the ones with the largest room (and necessity) for optimization, especially as far as number of carried out accesses is concerned. An analysis of memory access patterns of these entities has shown that most of the accesses generated by these entities are used for transmission of search area pels. Thus, the optimization effort has been focused on finding

a mechanism that would lead to reduction of the number of these transmissions.

Once the *zeroes2ones* transformation has been applied on the LR input pels of the to-be-super-resolved MB and its SA, these pels are projected onto HR grids. The projection is carried out in two stages: (i) computation of HR coordinates for the LR pels, and (ii) placement of the LR pels at the computed coordinates. The HR coordinates that have not been assigned a LR pel value during the projection step are initialized with the value of '0'. The LR-to-HR coordinates mapping depends only on the LR coordinates' values (limited, ordered set of fixed values) and the resolution of the motion estimation (fixed value). Thus, it is known as soon as these values are determined.

Following the execution flow of the reference implementation, the majority of accesses to memory made during the process of pels extracted, projection and fusion, are carried out in order to read non-LR pels values (coordinates initialized with the value of '0'). Thus, if the identified coordinates do not point at a LR-pel, the coordinate must be addressing a coordinate initialized with '0' and this value can be assigned without performing the actual access to the memory. Otherwise, the HR coordinate of the to-be-loaded pel can be used to obtain its LR coordinates and the pel value can be loaded directly from the LR representations of the reference structures and the MB being actually processed. This approach allows to use the LR representations of the reference structures in the pre-fusion pert of the execution flow (*zeroes2ones*, *upHoles/upGrids* and the extraction stage of the *shiftAdd*) at the expense of (slightly) increasing the computation complexity of the *shiftAdd* step. The identification of non-LR pels has been implemented by introducing a new addressing scheme that is capable of determining whether or not the to-be-extracted HR coordinate addresses a LR pel.

4.6.2 Optimization results

Introduction of the described addressing scheme has led to a significant reduction in memory traffic of both execution flows. When compared with the reference versions (i.e. without optimizations) the observed minimal, maximal and average memory access count have been reduced by 13.8%, 13.3%, 12%, and 25.6%, 87.4%, 81%, respectively for the FL and MBL flow. The much higher rate of reduction in access count observed for the finer grain flow has lowered the minimal and maximal factor of increase in memory traffic associated with the change to the MBL flow to 1.1 and 16.9, respectively from 1.22 and 117. This is an optimization that offers clear trade-offs for design implementation with fitting local memory and held down communications traffic.

Implementation of the new addressing scheme has allowed to eliminate most of the HR representations and effectively has lowered the memory occupancy of the FL and the MBL flow. The observed maximal memory occupancy has been reduced from 9158 KB and 1051 KB to 2475 KB and 718 KB, respectively for the frame-level and macroblock-level implementations. The minimal occupancy has been reduced from 3317 KB and 122 KB to 1832 KB and 113 KB, respectively. In summary, the introduced optimizations have lowered the minimal, maximal, and average memory occupancy by 73%, 44.8%, 62.5%, and 31.7%, 6.9%, 22.7%, respectively for the FL and MBL flow. As a result the new addressing scheme has also lowered the observed minimal and maximal value of the factor of reduction in memory occupancy associated with the change to the MBL flow to 3.5 and 16, respectively (from 6.8 and 40). Despite the noticed decrease, the change from frame-level to MBL flow is still significant enough, with held down memory traffic, as to likely turning the MBL approach the preferred one.

Additionally, introduction of the new addressing scheme has further reduced the number of test combinations for which the increase in memory access counts is still greater than the reduction in memory occupancy achieved. A superposition of the factors of reduction in memory occupancy versus the factor of increase in memory traffic computed for the reference and the optimized versions is presented in Fig. 4.11. The figure shows that for scale of 4 (worst case) the number of combinations for which the observed increase in memory traffic is greater than the occupancy reduction has been reduced from 12 to 5. Considering all 96 test combinations the number of these combinations have been reduced from 20 to 9. Notice that for these combinations, the memory size is still significantly reduced, nevertheless a moderate surge in traffic is expected that has to be dealt with or tolerated.

4.7 Conclusions

The NUGPA algorithm presented in the previous chapter was expected to pose memory requirements that preclude its hardware implementation. In order to tackle this problem the algorithm's dependency on frame-level buffers had to be eliminated. In this chapter the proposed changes to the NUGPA execution flow that facilitate hardware implementation have been presented and quantitatively evaluated.

The chapter started with a brief description of the reference and the proposed execution flows. Following, in order to quantify the memory occupancy and memory traffic of both flows, and their change associated with switching to the MBL flow, a theoretical study

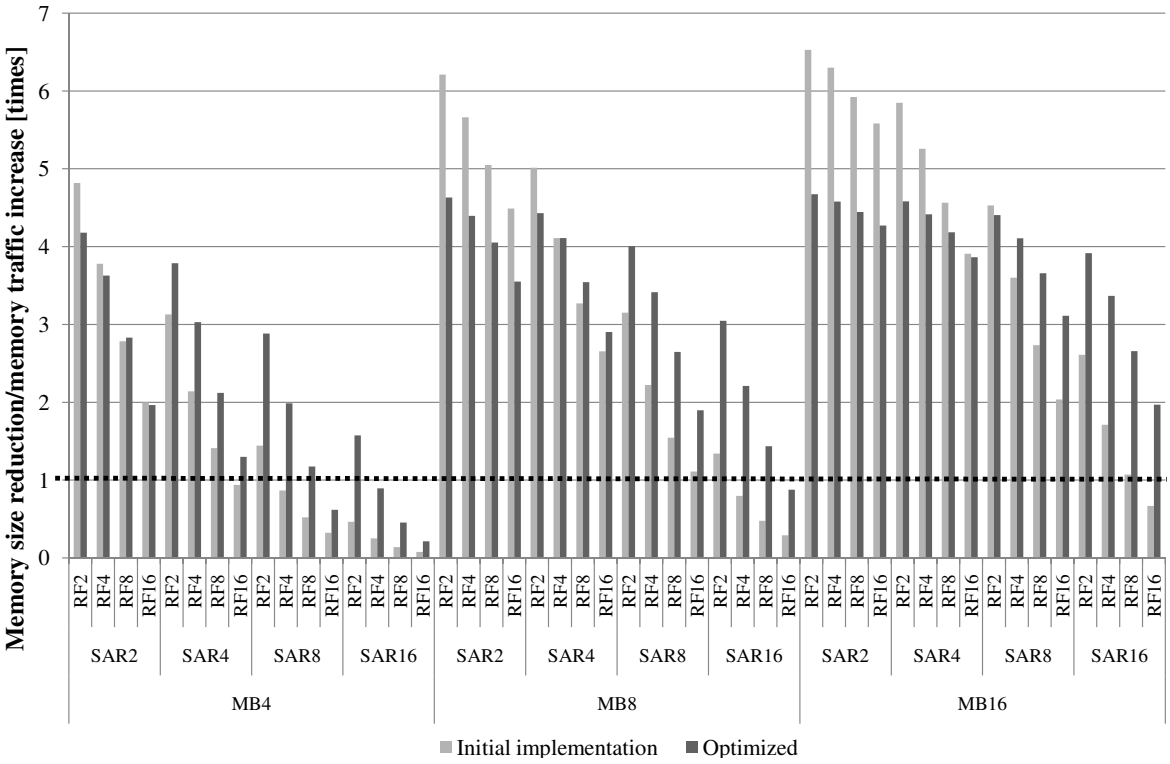


Fig. 4.11 The factor of reduction in memory occupancy versus the factor of increase in memory traffic for initial and optimized macroblock-level implementations (for scale=4). MB4, MB8 and MB16 stand for $MB_{width} = 4, 8$ *and* 16, *respectively.*

based on the available software implementation was carried out. The results of the study, which considered a set of 96 combinations of SR parameters values, have shown that MB-level implementation is expected to lead to a reduction in memory occupancy at the cost of significantly increased memory traffic. For the tested configurations the macroblock-level implementation used no more than 14.7% of the memory occupancy computed for an equivalent frame-level system (for a QCIF input).

Following the initial study, system's bottleneck identification and algorithmic-level optimization of the system has been carried out. The presented optimization was the modified addressing scheme which has led to significant reduction in size of memories used in the pre-fusion part of data flow. The significance of this optimization has been quantitatively estimated using the set-up established for the initial study. The expected minimal and maximal value of the factor of reduction in memory occupancy associated with the change to the MBL flow is of 3.5 and 16. The expected minimal and maximal factor of increase in memory traffic is of 1.1 and 16.9, respectively. Considering all evaluated combinations, the factor of reduction in memory occupancy is expected to be greater than the factor of increase in

memory traffic for 87 out of the 96 configurations. Notice that for the remaining 9 configurations the memory occupancy is still significantly reduced. Nevertheless, a moderate surge in traffic is expected that has to be dealt with or tolerated. The results reported for memory traffic should be treated as the worst case scenarios, as the evaluated MB-level system did not implement nor considered any architecture-level optimization. When introduced, these optimizations (e.g. buffering schemes) are expected to significantly lower the number of memory accesses and boost the expected implementation efficiency of the proposed MBL flow.

Chapter 5

Implementation methodology

5.1 Introduction

The goal of any hardware implementation is to create a system description that, when implemented on the targeted device, results in a hardware system having the same observable behavior as the provided reference, while meeting the performance requirements. In the typical design flow the reference algorithm is first analyzed and architecturally constrained, before being used to create a register-level and pin-accurate description in a hardware design language. This description can then be used to create a gate level description used to configure the targeted hardware. In most hardware implementations the whole system functionality is first prototyped in software. This intermediate step is used to test the algorithm functionality and provide a high-level of abstraction reference to be used in validation and testing.

Historically, the only option to obtain the required register-level/pin-accurate description was to manually code the system directly from the provided algorithmic (or functional) description. Recently, an emerging trend in the design and verification of electronic systems is to provide not one, but a set of system models. The idea behind this flow is to enhance the probability of a successful implementation of functionality in a cost-effective manner by providing a hierarchy of tightly-coupled models that become progressively more accurate.

In our implementation, we have opted to follow the second flow, referenced in literature as the electronic system level (ESL) methodology. In this chapter, we will present the established methodology for implementation and verification of our design in FPGA devices. The provided description will contain details on the design flow, used models of abstraction and refinement steps required to obtain a model that can be used to create the final pin-accurate register-level model. To facilitate re-using of the established methodology and

allow its application in other implementations: (i) the description will be structured and presented in a way that follows the design flow, and (ii) most of the technology-specific details will be generalized, were possible.

5.2 Modeling of hardware-targeted systems

The design flow for system implementation presented in the previous section requires a series of abstractions forming a hierarchy to be created before the targeted hardware can be configured and ready for testing. Typically, the following representations are being created:

1. System description at the algorithmic/functional level (hereafter the representation 1).
2. System description at the structural level (logical (behavioral or structural) description level) (hereafter the representation 2).
3. System description at the physical circuit components level (hereafter the representation 3).

In electronics, the process of creation of a lower-level of abstraction from a higher-level one is referred to as *synthesis*. When the synthesis is automated, the tool/computer program that carries out the synthesis is called a *synthesis tool*. In reference to the above presented representations, there are two types of synthesis processes:

***High-level synthesis*,** the process by which a functional description of a desired behavior is turned into a description in terms of logic functions and processes equivalent to an description in a hardware description language. Encapsulates the transformation of representation 1 into representation 2.

***Logical synthesis*,** the process by which an abstract form of desired circuit behavior (described in terms of logic functions and processes) is turned into a design implementation in terms of logic gates. Common examples of this process include synthesis of the hardware description languages, including VHDL and Verilog, resulting in generation of a hardware configuration file. Encapsulates the transformation of representation 2 into representation 3.

5.2.1 High-level synthesis

The goal of HLS is to increase the efficiency of the process of hardware design and verification, by means of: (i) provision of rapid ways of design space exploration facilitating

the process of design architecture optimization, and (ii) support for multiple levels of abstraction allowing the design to be modeled with various levels of details with the required inter-model transformation of the representations being facilitated by automated HLS tools.

Assuming that the algorithmic level description is provided in software design language, then, the HL synthesis becomes the process of obtaining a hardware level description from software level one. The difference between the so called software and hardware design languages is that the former (for example ANSI C) have no notion of time (nor event sequencing), limited concurrency (hardware is inherently concurrent) and no hardware specific data types (e.g. 'Z' state for tri-state buffers, etc.). Thus, a need for conversion that bridges the two worlds arises.

There are two ways for carrying out the HL synthesis, namely, the HL synthesis can be carried out manually or by using a HL synthesis tool.

5.2.1.1 Manual conversion

In this methodology, hardware description is carried out manually by a designer. The hardware description is created from scratch. The available base code is used only as a source of information on the design functionality and to produce the reference output used in the process of verification. This approach leads to a highly optimized, low level implementation that requires a significant amount of time for system creation and validation. As shown in Fig. 5.1 use of manual conversion decouples the (software) system model and the HDL model. Due to non-propagative relationship between the software and hardware implementation, implementation of changes introduced to the base software, as well as evaluation of their impact on the system performance, is complicated and requires significant amount of effort and time. To recap up, the main drawbacks of manual conversion are as follows:

(i) There is no 'global' (or 'root') system model and the created descriptions are decoupled requiring multiple system tests.

(ii) System evaluation is performed at the end of the flow.

(iii) Design space exploration is carried out using the end of the flow description (time consuming) modeling/simulation.

(iv) Limited re-usability in new designs.

(v) Manual propagation of changes due to lack of conversion automation.

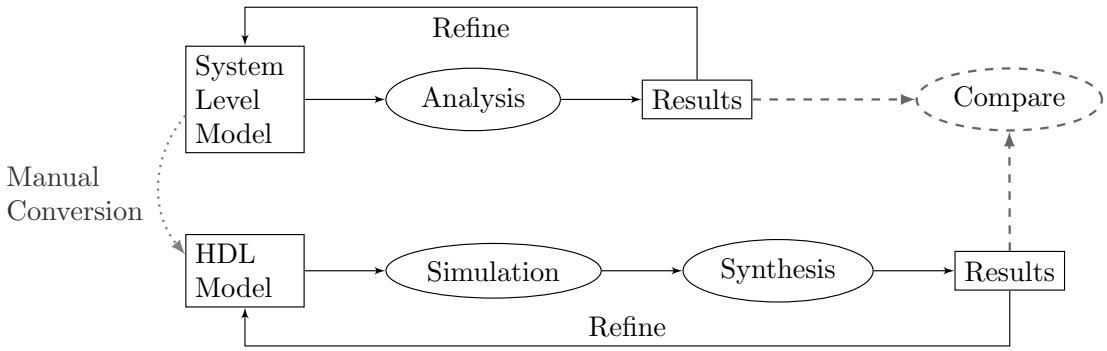


Fig. 5.1 Manual HL synthesis.

5.2.1.2 Computer aided conversion

In this approach the base software code is somewhat used to create the HDL description. In order to bridge the gap between the algorithmic and HDL descriptions the automated HL synthesis operates on an intermediate representation. This representation imposes limitations on the set of allowed software language constructs and introduces additional constructs to represent (some of) the missing HDL constructs. The execution flow and most of the software code remain unchanged during the transformation to the intermediate representation. The coherency between the representations, as presented in Fig. 5.2, facilitates propagation of changes introduced to the base code onto the hardware implementation and allows rapid system modeling using the intermediate representation which can be simulated at much high speeds then HDL. The HDL is generated automatically from the intermediate representation by the HL synthesis tool.

Although, this approach solves many of the issues associated with manual conversion it faces some issues of its own:

- The creation of the intermediate representation requires additional effort. The learning curve is somewhat steep.
- The HDL created by the synthesis tool is human illegible, this significantly limiting the possibility of manual introduction of post-synthesis changes or tweaks.
- Even the smallest change made to the intermediate description requires a synthesis re-spin.
- The relation between the input and the output of the synthesis is somewhat non-deterministic. That is, small changes in the input may result in significant changes

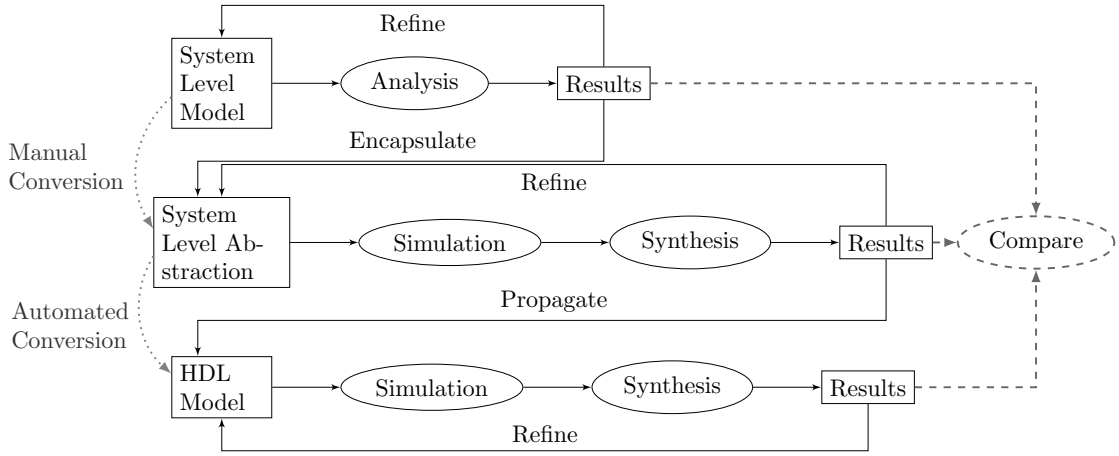


Fig. 5.2 Computer aided HL synthesis.

in the created HDL. The observed changes are not always intuitive.

- The quality of produced HDL is somewhat inferior to the one manually written.

5.2.2 Electronic System-Level design

Contemporary electronic systems have shifted towards Systems-on-Chip (SoCs) and in particular Multi-processor SoCs (MPSoCs). These systems are not only more complex but also call for solutions of additional issues, especially in the context of very competitive industry of consumer electronics. Among most important challenges faced by electronic systems design methodologies, the *European Space Agency* (*ESA*) [Age08] lists the following ones: (i) reduced time-to-market, (ii) increased verification complexity that scales with the system components and the need to use vendor-specific tools, (iii) huge design space, (iv) highly complex software development, and (v) need for methodologies enabling concurrent design of hardware/software.

All of the above issues can be solved (at least to some extent) by the use of tiered hierarchy of high-level abstractions during system specification and during initial phases of electronic system design. Abstraction levels are used to quantify the degree of details simulated in the model and expressed in terms of the accuracy of the representation. In literature, the set of complementary methodologies that enable the design, verification, and debugging through the hardware and software implementation is referenced to as the *ESL design and verification methodology*. The multi-tiered structure allows to have several system descriptions each focusing only on the relevant implementation's details and allowing better

work partitioning, and significantly faster design and simulation times. Due to its multi-tier nature, ESL methodology leverages the computer aided conversions for the tier-to-tier transitions.

5.2.2.1 Representation accuracy

As aforementioned, ESL uses multiple levels (or tiers) of abstraction. The modeled levels of abstraction are characterized by their accuracy understood as the level of details that are included in (or, in other words, abstracted from) the model. The model's accuracy on its own is defined in reference to certain aspects of the model. In practice two main aspects of a model can be distinguished: timing (or temporal) accuracy and structural accuracy. Many models introduce additional components of accuracy by considering different levels of abstractions for functionality, and by separating transmissions (interfaces) from functionality (in this context, computations) in the context of temporal accuracy. A particular level of abstraction is defined by providing accuracy in some or all domains.

In the context of temporal accuracy the following models can be distinguished:

A. **Untimed (UT) models.** Refer to model communication interfaces and functionality. Does not use time to regulate the execution. Assume execution and data transport to be immediate and requiring no time.

B. **Timed Functional (TF) models.** Refer to model interfaces and functionality. Use 'time' to regulate the execution. Implement the notion of time by dealing with system changes at specified boundaries (i.e. cycle, system event and deltas). Latency is modeled with execution and data transport requiring (non-zero) time.

C. **Cycle Accurate (CA) models.** Require that during a cycle all entities that request to act are given an opportunity to execute for one cycle.

C.1 **Bus Cycle Accurate (BCA) models.** Refer to models of interfaces not functionality. Timing is cycle accurate and tied to some global clock. Require complete timing down to the notion of bus transfer, so that the description could be defined down to the exact order that they appear in the real system. Even though, data transport is carried out at a coarse level using transactions, complete information, in terms of when the bus transfers occur, is required. BCA does not infer pin-level accuracy.

C.2 **Pin Cycle Accurate (PCA) models.** Refer to model interfaces not functionality. Timing is cycle accurate, with communication modeled at the level of single pins.

C.3 **Register Transfer (RT) accurate models.** Refers to functionality. Infers low-level accuracy with timing of all components. Complete detailed description with every bus and every bit fully modeled.

D. **Phase accurate models.** Require that during a cycle all entities that request to act are given an opportunity to execute many times during one cycle. The resulting changes to the system are being considered with sub-cycle accuracy by means of additional time quanta called *delta* phases.

In the context of structural accuracy the following levels of details can be distinguished.

A. **Pseudo resources.** Provide description of the system in terms of structural units which do not consider the design as composed of entities which have their direct counterparts in hardware.

B. **Functional units.** Provide description of the system as composed of coarse elements, defined at the architectural level, grouped by their function in the system (i.e. processing unit, memory, register, etc.). At this level units interfaces are established. Behavior of the defined elements is defined in terms of the input-output relationships.

C. **Pipelines.** Behavior of the defined elements is defined for each stage of their execution pipeline.

D. **Pins.** The state of the system is defined in terms of pins and bits states, also in context of the sub-cycle events (i.e. interrupts).

Both, the temporal and structural aspects of representation accuracy are interdependent, that is, in order for the abstraction accuracy to be meaningful the model has to simulate time with a required fidelity and use structures that operate at the given time resolution level, and *vice versa*. This interdependence is illustrated in Fig. 5.3, where pseudo-instructions represent the notion of untimed execution.

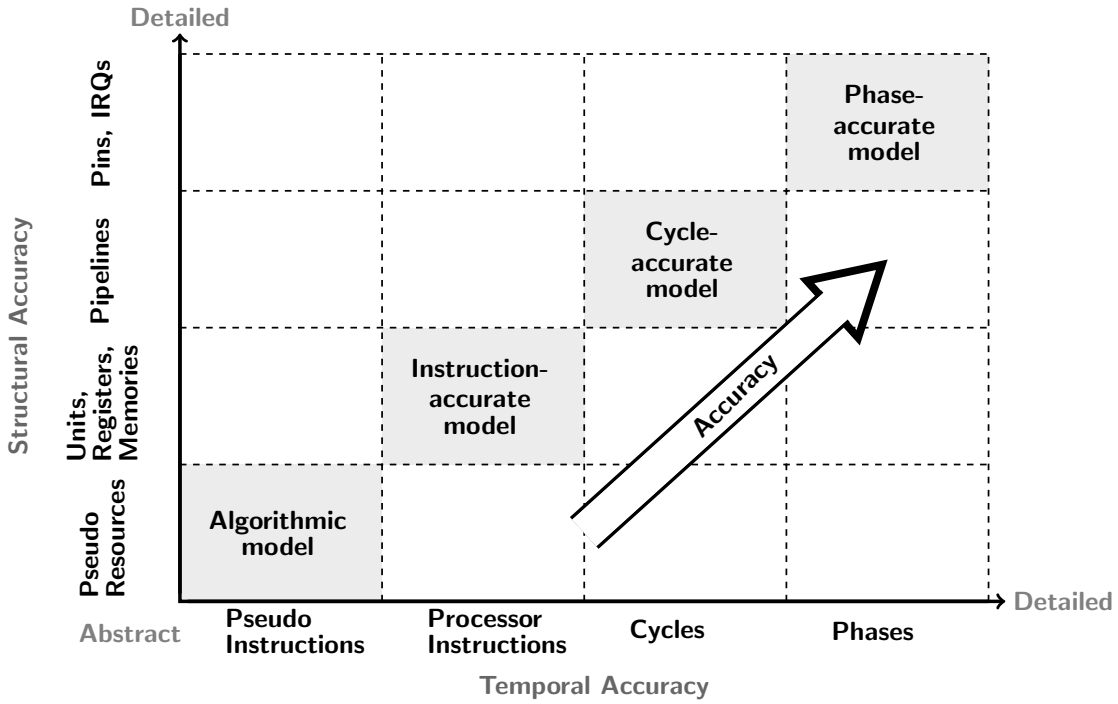


Fig. 5.3 Abstraction levels accuracy in system modeling. Based on [Fos11].

5.2.2.2 Models of abstraction

The bare-metal details of a hardware design can be abstracted using a variety of models. The used levels of abstraction can fall somewhere in between the two opposing extremes: the gate level and algorithmic level models. Considering the aforementioned accuracy models and dependencies the most commonly used levels of abstraction are the ones presented in Fig. 5.4:

A. **Register Transfer Level (RTL) model.** RTL is a timed model that specifies the design at the level of bit-accurate logical entities. The details of physical implementations are abstracted by using logic functions and registers. The notion of time is abstracted by dealing with system changes on clock cycle boundaries. RTL communications are carried out by wiggling pins over a period of time. Transfers are pin-cycle-accurate and execution is fully timed.

B. **Bus Functional Model (BFM).** Non-synthesizable timed model that abstracts the details of computations and puts the emphasis on verification of correctness of communication interfaces and system bus transfers. Mainly used for verification of testbench communication. The transfers are usually pin-cycle-accurate

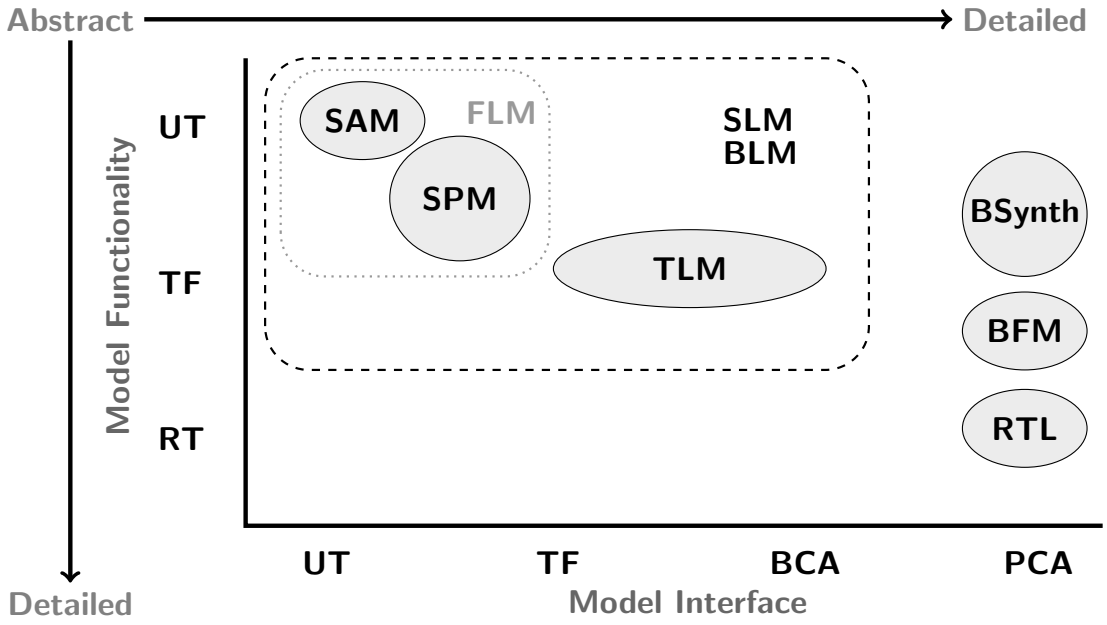


Fig. 5.4 Abstraction levels in system modeling in function of their accuracy.

and carried out with using pin-level interfaces. Variations of the BFM that carry out interface verification at (a higher level) of transactions are known as *Transaction Verification Models* (*TVM*).

C. **Behavioral Synthesizable (BSynth).** BSynth is a timed model that specifies the design at the functional/behavioral level using only synthesizable constructs. This model can be used to obtain RTL/gate level description using an appropriate HL synthesis tool. Transfers are pin-cycle-accurate and fully timed. Computations execution is carried out with some notion of time. Various temporal accuracy specifications can be used.

D. **Transaction Level Model (TLM).** TLM is typically used in reference to modeling of hardware only. TLM abstracts communication (pins and events of the communication) by introduction of transactions to model data transfers and system behavior. A single transaction abstracts a collection of individual events occurring over some period of time. Transaction level communication is carried out by function calls. Communications, as well as functionality can be timed functional but not cycle accurate.

E. **System Level Model (SLM).** SLM encapsulates models with level of abstraction higher

than RTL.

F. **Functional Level Model (FLM).** FLM encapsulates models with level of abstraction higher than TLM. These models represent an executable specification of hardware and software components.

F.1 **System Architectural Model (SAM).** Used for design architecture exploration, algorithm determination and functional verification. Defined interfaces and functionality are modeled as untimed, with no pin-level details used for modeling communications. Computations are modeled using sequential execution.

F.2 **System Performance Model (SPM).** Used for design space exploration, time budgeting and architecture benchmarking. Introduces some high-level notion of time and concurrent behavior modeling.

5.2.2.3 Implementation considerations

Independently of the abstraction-tiers used, the final representation of the design is the device configuration file. In the past, this file would be created by means of using the manual conversion approach and logic synthesis from RTL model. Nevertheless, manual creation of the RTL model results in decoupling of the reference and implementation. A contemporary alternative, one that allows designers to work at a higher level of abstraction and leverage the advantages of high-level synthesis, is to follow the ESL design flow. The ESL flow starts with an algorithmic description in a software design language (e.g. C++). The designer typically begins by developing the System Architectural Model and interconnects protocol in a high-level of abstraction hardware design language (e.g. SystemC). Defined Functional Level Models are refined until the results obtained from the System Performance Model are satisfactory. Then high-level synthesis tools handle the micro-architecture and transform untimed (or partially timed) functional code into fully timed RTL implementations, automatically creating cycle-by-cycle detail for hardware implementation. Optionally, the System Level Model (e.g. TLM) is developed and refined before the high level synthesis is carried out. The resulting (RTL) implementations are then used directly in the logic synthesis to create a configuration file. This means that independently of used hierarchy of abstraction levels an HDL (RTL) description of the design will be generated at some point.

The direct manual conversion flow would most likely use one of two of the standardized hardware design languages: VHDL [IEE02] or Verilog [IEE06]. These hardware design languages provide specialized means for creation of accurate HDL/RTL level descriptions in

an efficient manner. Nevertheless, the scope of the levels of abstractions used by ESL is very broad, reaching well beyond the scope of abstraction provided by these languages. Thus, lately, there has been a growing interest in complementary languages that could be employed for descriptions at a much higher level of abstraction. The motivation behind such efforts is the simultaneous increase in design complexity, with multi-million gate designs, and the increase in pressure to get designs out faster with first-time design success. Among those high-level languages, SystemC [Ins06], SystemVerilog [IEE09], and SpecC [GJZGSZ00] represent the most widespread. These languages fuse a well-known syntax with powerful constructs, enabling the modeling and simulation of complex systems; in particular SystemC has grown more and more popular. Most HLA languages are implemented as a set of classes and macros which extend the base language by providing an event-driven simulation kernel, together with signals, events, and synchronization primitives, deliberately mimicking the hardware description languages like VHDL and Verilog. In fact, these languages not only reduce the gap between software and hardware descriptions but also allow much faster and robust verification flow.

As presented in Fig. 5.5 the scope of supported levels of abstraction differs significantly among different hardware design languages. Up to the date, no programming language is capable of handling all of the defined levels of abstraction. Thus, the implementation work-flow should consider the tiers of abstraction that are planned to be used, and use the appropriate language(s) to drive the design down to the bare-metal configuration file.

5.2.3 The SystemC language

SystemC, also known as the IEEE standard 1666 [Ins06], is a set of C++ classes and macros that deliberately mimic the functionality found in hardware description languages (i.e. VHDL and Verilog). While the latter are often used for Register Transfer Level descriptions, SystemC is generally applied to system-level modeling, architectural exploration, software development, functional verification, and high-level synthesis. While HDL languages are often used for Register Transfer Level descriptions, SystemC is generally applied to system-level modeling, architectural exploration, software development, functional verification, and high-level synthesis [Age08]. The architecture of the SystemC language is shown in Fig. 5.6. The main components of the standard are events, signals, hardware-level data, the simulation kernel (written in C++) and synchronization primitives. When combined, these components allow the execution of an event-driven simulation of concurrent systems at a higher-than-RTL level of abstraction [Age08]. This allows the introduction of

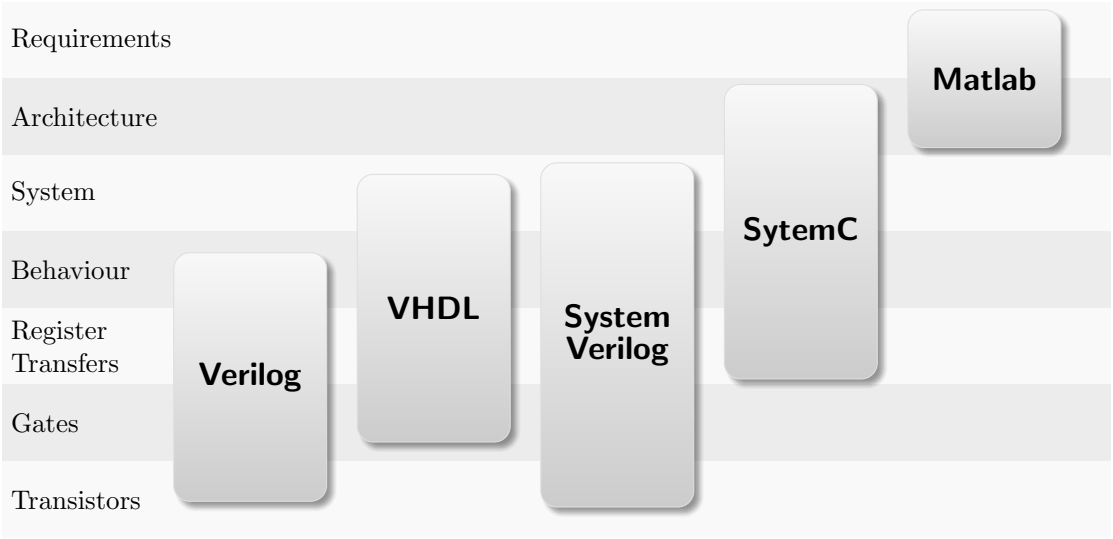


Fig. 5.5 Abstraction levels offered by the most popular hardware description languages. Based on [Age08].

System-Level Models which focus on the actual functionality of the system more than on its implementation details.

From all available levels of modeling, SystemC emphasizes the use of two levels and two description modes. These two levels of modeling are the high abstraction *Transaction Level Modeling* ([Ghe06], [CG03]), and the *Register Transfer Level*. The description mode is either a behavioral *untimed mode*, or a low level cycle-accurate *timed mode* ([Gro02], [CMMC08]). (All of these modes/levels have been already described in Section 5.2.2.2.) SystemC permits all combinations of modes and levels, but only timed descriptions, using a specific subset of the language, lead to reasonable quality of the hardware-level description obtained using high-level synthesis. In literature these descriptions are referred to as being *synthesizable*.

In the context of the SystemC language, the Transaction Level Modeling can be defined as the modeling style where, at least one of the two system components, the communication and computation, introduces an approximate concept of time. In 2008, the OSCI committee has proposed a TLM library composed of a set of SystemC primitives that allow designers to implement several transaction level communication protocols with different degrees of accuracy (i.e. Programmers View, Programmers View with timing, etc. [CG03]). This nomenclature was superseded in 2009 with the release of TLM 2.0 [Gro09] which specifies the following two terms to describe the accuracy of the timed mode [BM10]:

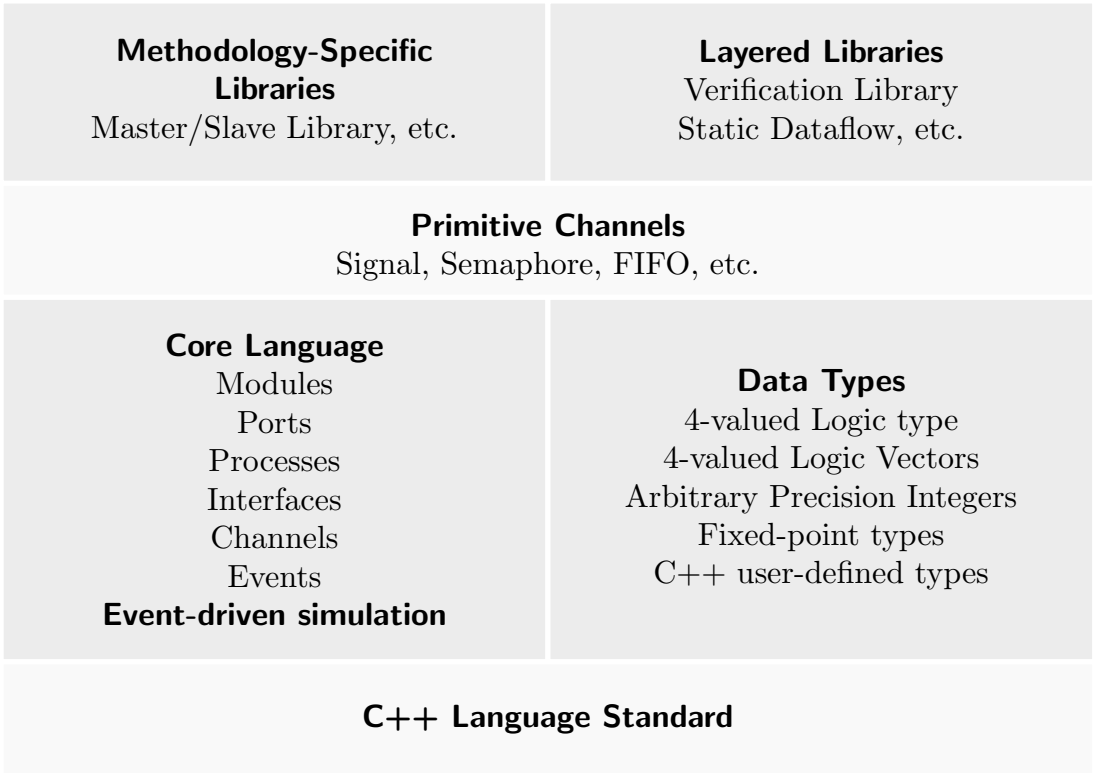


Fig. 5.6 SystemC language architecture. Based on [Swa01, Figure 1].

A. **Loosely timed (LT).** Implements temporal decoupling of system models with blocking interface. The process that is given the control executes all tasks till it blocks. Then the control is passed back to the scheduler and passed onto another module that requests it. This allows some models to run ahead in their own time, up to the quantum boundary. There is no common time reference. Time is tracked locally (in quantums) or not at all. This mode requires explicit synchronization in order to work.

B. **Approximately timed (AT).** Every model is synchronized to a common simulation time. The time elapsed when the module runs corresponds to the time of the system. Processes are kept in lock-step in simulation time using *wait()* and/or *notify()* statements allowing the use of non-blocking interfaces. Each process is synchronized with a SystemC scheduler. The level of the modeled delays is customizable (ranges from approximate to detailed).

The core concept of TLM is to model only the level of detail that is needed. The elim-

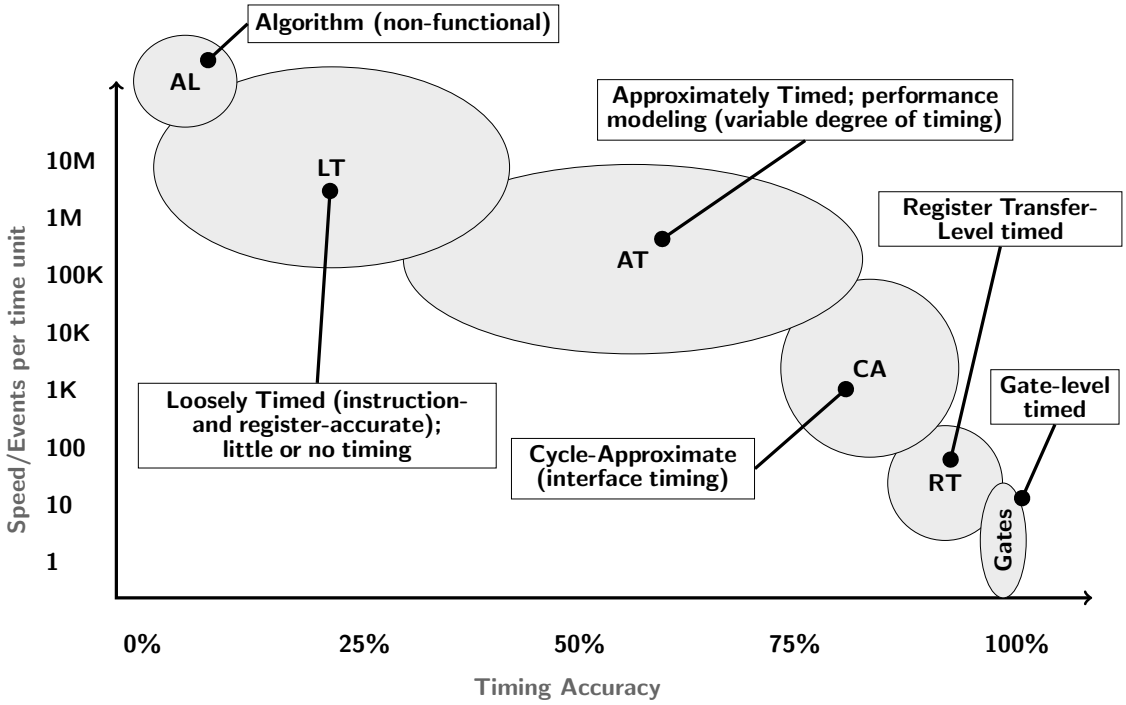


Fig. 5.7 Expected simulation speed for different levels of abstraction.

ination of the un-necessary details leads to huge gains in simulation speed [Fos10, Gro02]. Moreover, since the detailed implementation has not been finalized yet, it is still possible to perform consistent changes to the design, enabling an effective evaluation of different architecture alternatives (including the partitioning of the functionality between hardware and software). An coarse-grained approximation of the simulation performance offered at different levels of abstraction is presented in Fig 5.7. From this figure it can be seen that by using a higher level of abstraction and reduction of timing accuracy a significant speed up can be obtained. The relative speed up obtained from carrying out of the initial simulation at the above-presented level of accuracy of timing vs the RTL simulation time is of orders of magnitude. This difference has an immense impact on the time and quality of the verification stage, allowing faster and more thorough validation.

5.3 Design and implementation methodology

To accelerate hardware implementation of the super-resolution kernel we have opted for the design verification methodology presented in Fig. 5.8. The established methodology has been based on the methodologies co-developed by the author and successfully used in the

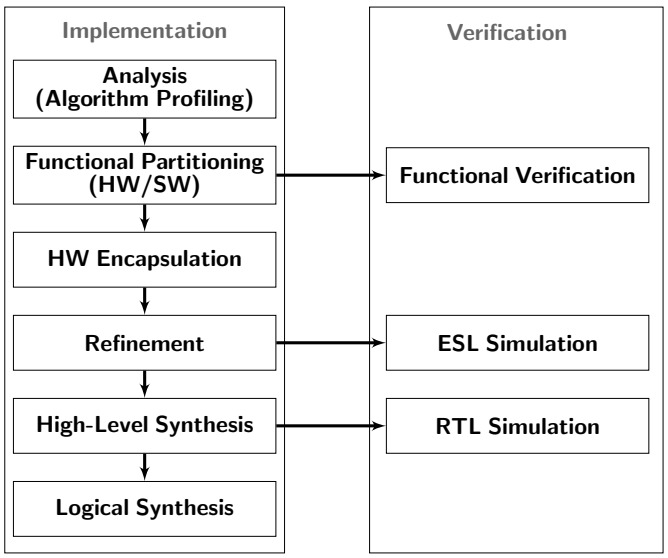


Fig. 5.8 Implementation and verification methodology.

IUMA projects described in [SHFC+08, TSC+09, TCH+09, CEN+13b, CEN+13a]. The methodology follows the typical ESL flow. First, the provided reference code is used to create a functional high-level of abstraction model. Then, this model is iteratively refined, and, using high-level synthesis methodology, transformed into RTL microarchitecture. The latter is used to create the required hardware configuration file by means of logical synthesis.

In the case of NUGPA hardware implementation, the provided base code had been written in ANSI C. This code has been instrumented and used as the base for functional level analysis. The results of the analysis stage allowed to determine the workload of each functional structure and guided the specification of hardware modules prototypes and hardware/software partitioning. Following, the C functions have been grouped by functionality and encapsulated as SystemC modules. In this work encapsulation is defined as the process of defining a hardware entity prototype. In the case of the super-resolution core, it has been decided that only the super-resolution kernel will be implemented in hardware, leaving block-matching, a well-defined and known problem, for a standalone implementation [HN10]. During functional partitioning it has been decided that the processing of the chroma components will not be implemented using FPGA resources, but rather it will be carried out in parallel using one of the SoC's microprocessors. As pointed out in [IP91] chroma component values are not worth being super-resolved and hence they are being simply interpolated.

The functionality encapsulated by the super-resolution kernel (luma) was to be iteratively refined in order to allow its high-level synthesis. On the contrary, the functional part of block matching and chroma processing were not refined. After initial encapsulation, only the part of the code carrying out the communication was being changed in order to reflect the modifications introduced in the super-resolution kernel. Most of the design decisions and profiling have been made during the decomposition and refinement stages, including the analysis of the impact on resources occupancy and critical path latency. The refinement re-iterations have been guided with the performance measurements (latency, cycle time and resource utilization) obtained for the high-level synthesis results. This assured the highest quality of results of the final system.

In order to obtain the HDL representation of the system, the TLM description had to be refined into a synthesizable one. A SystemC description is considered synthesizable if it utilizes a synthesizable subset of the SystemC language. The set of constructs considered synthesizable is vendor-specific. In our case, the preparation of synthesizable model for the high-level synthesis involved substitution of language constructs not supported by the chosen SystemC compiler by the ones being supported. Among the required modifications the most significant ones were: dynamic memory allocation elimination, array/memory dimensionality reduction, pointer arithmetic elimination, combinational loops breaking, inlining of functions, allocation of arrays in RAMs and floating point data elimination. Once the code was synthesizable, a RTL description of the core was synthesized and used in logical synthesis. More details on encapsulation, refinement and strategies used for our implementation are provided in the following sections. Verification methodology is presented in Section 5.4.

In the established implementation methodology C code development and verification (we had to implement the MB-level flow) has been carried out using Microsoft Visual Studio. This environment was also used to carry out SystemC encapsulation and simulations using the *Open System C Initiative* (*OSCI*) simulator (now hosted on Accellera website) [Acc08]. The SystemC code has been synthesized using the Celoxica Agility Complier [Cel06]. The result was revised using the tool and the created RTL was further optimized using Synplify 2009.06 Premier. The optimized HDL/EDIF was then used to obtain the vendor-specific configuration file using the *Xilinx Synthesis Technology* (XST) tool from the Xilinx ISE 14.6. The mixed-language and RTL simulations were carried out using ModelSim 6.4. It is important to note that ModelSim uses its own version of the SystemC simulator (not the reference OSCI one). The used tools and their role in the established implementation methodology are presented in Table 5.1.

Later in the development, a second set of tools has been used. The first set, the one centered around the use of the Celoxica Agility Compiler had to be abandoned due to this tool being discontinued in 2012 (with license expiring in 2013). The set of tools to which the work have been migrated is presented in Table 5.2. In this environment the HL-synthesis has been handled by the Cadence C-to-Silicon tool. For compatibility reasons the Cadence Incisive Simulator was used for SystemC/mixed-language simulations. The rest of the tools deployed in the flow remained unchanged. From our experience, the Cadence-based deployment proved to be much more reliable and more efficient in terms of execution/simulation speed, quality of results and ease of use.

5.3.1 Established ESL design flow

One of the objectives of the algorithm analysis and refinement was the determination of code functional groups. These groups formed the base for initial system decomposition into prototypes of hardware entities. These entities encapsulated the functionality of the code, allowing it to be used in ESL simulations. The SystemC description of the system was used in modeling and underwent the refinement carried out following the methodology proposed in [TSC+09].

1. As a first step, a transaction level model of the super-resolution kernel was created, using the TLM-2.0 standard. This was a high level functional model in which inter-block communication involves the exchange of transaction objects using TLM-2.0 interfaces. The TLM-2.0 model was verified using the same test cases applied to the C model and was used as the reference model for the subsequent micro-architecture exploration and RTL implementation. Each of the kernel functional blocks was refined, implemented and verified independently.

2. In the modeling phase, the code was made synthesizable, that is, it was modified for an initial high level synthesis in the SystemC editor. Among the changes made was the replacement of the TLM-2.0 socket interfaces with SystemC signal channels, creating a *pin-level* interface, and the addition of timing information using *wait()* statements. The refinement methodology employed involves a modeling phase, high level synthesis, optimization and logic synthesis.

3. High level synthesis was performed next using HLA-synthesis tool (Agility Compiler or C-to-Silicon), in order to obtain an area and delay estimation summary.

TABLE 5.1 *Tools and their role in the Agility Complier centered deployment.*

Tool	Purpose	Description
Microsoft Visual Studio	Analysis	C code execution.
	Functional verification	
	ESL simulation	SystemC code execution using SystemC 2.2 and TLM 1.0 [Acc08]
Celoxica Agility Compiler 1.2/1.3	HL-synthesis	SystemC refinement and RTL creation.
Mentor Graphics ModelSim 6.4	ESL simulation	Functional and performance checks.
	RTL simulation	Mixed language simulation.
Synopsys Synplify 2009.06 Premier	Logic synthesis	RTL optimization and EDIF creation.
Xilinx ISE 14.6	Placement and mapping	Final results assessment.

TABLE 5.2 *Tools and their role in the C-to-Silicon centered deployment.*

Tool	Purpose	Description
Microsoft Visual Studio	Analysis	C code execution.
	Functional verification	
Cadence C-to-Silicon 12	HL-synthesis	SystemC refinement and RTL creation.
Cadence Incisive Simulator	ESL simulation	Functional and performance verification.
	RTL simulation	Mixed language simulation.
Synopsys Synplify 2009.06 Premier	Logic synthesis	RTL optimization and EDIF creation.
Xilinx ISE 14.6	Placement and mapping	Final results assessment.

4. In order to determine if the results were satisfactory at this stage, initial reference values were set for the target operation frequency and for the FPGA area consumption, depending on functional block complexity and other factors. If the area and delay results obtained did not meet the requirements, an iterative process involving the optimization of the SystemC code in the editor and the execution of the HLA-synthesis tool to obtain area and delay results was performed.

5. Once the synthesis results obtained with the HLA-synthesis tool were satisfactory, logic synthesis was performed next, leading to a further improvement in area and delay results. The performance constraints were check again after obtaining final area and delay results based on the FPGA programming information. Unsatisfactory results led to further modeling/refinement iterations.

5.3.2 Structural decomposition and encapsulation

In order to take advantage of the parallel nature of hardware implementation, the software sequential execution flow had to be remodeled. This process involved two main tasks: (i) system decomposition into modules (code encapsulation), and (ii) definition of inter–module communication schemes. Partitioning a design into modules has the advantage of allowing (but not guaranteeing) parallel execution. Module partitioning and refinement formed a crucial part of the performance optimization process. In the context of hardware implementation, the inter-module communication should be planned with the view of facilitating non-blocking execution of communicating modules. (Inter-module communication is the topic of Section 6.3.2.2).

The process of code encapsulation into modules should be carried out in a fashion that assures the lowest number of synchronization points and non-blocking processing of the newly created modules. For the purpose of code encapsulation, we defined two types of inter-module relationships: the *unidirectional* and the *bidirectional* relationship. The classification was based on the data passing flow: if modules interchanged data mutually, that is both modules acted as initiators and targets, their relationships were denominated as bidirectional. Otherwise, the relationship was classified as unidirectional, meaning that, considering a pair of modules, only one module acted as the initiator and the other was acting only as the target. Generic examples of uni- and bidirectional relationships are presented in Fig. 5.9.

Execution flow of modules with bidirectional relationship is characterized by the fact that one of the modules outsources some amount of workload to the other and waits for

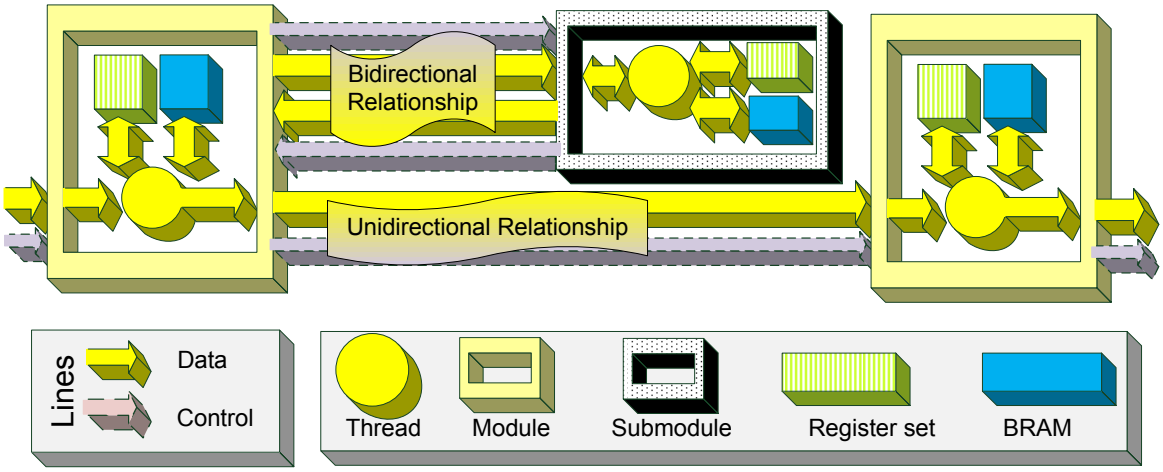


Fig. 5.9 Generic inter-module relationships.

its response. If the remaining workload of the sending module cannot be carried out concurrently during the time necessary for response reception, the module is blocked until the response is received. This is likely to result in lowering the amount of exploitable parallelism. For this kind of relationship, work redistribution and module inter-communication implementation is troublesome. Usually, a better solution is to convert the module relationship to a unidirectional one. This is done by decomposition of the waiting module into two modules, of which one carries out the computation and sends the data, and the other receives the data and post-processes them.

Considering the above-presented classification, it was established that the system after the decomposition should comprise only modules that feature unidirectional relationships. In order to enforce unidirectional relationships, a module should be created only from parts of code that have been *isolated* from the rest of the code. The isolation is a process of assuring that the code can be executed on its own, once provided, in a single synchronization event, with the required input. First, module synchronization points in the code had to be determined, along with data dependencies. Then, global variables referenced by the code had to be replicated as local ones, added to the list of input arguments, and, if changed by the code to be encapsulated, also to the list of the output arguments. Finally, the code referenced by function calls had to be made local or inlined. Once the code has been isolated it constituted an independent part of functionality and could be *encapsulated* as a SystemC module.

The goal of the encapsulation was to obtain a SystemC module that internally executes the code being encapsulated. In SystemC a module is an isolated piece of code that com-

municates with the rest of the code by means of SystemC communication constructs. The module is required to define at least one thread process that can be mapped to execute the encapsulated functionality, communications or both. Otherwise the module will be considered to be purely combinational. The thread process execution has to be implemented as an infinite loop. Once instantiated, the module reads input data from the input interfaces and stores then in the input data structure of the functional block. The original C code included in the SystemC module is invoked to process the data stored in the input structure. The C code output is stored in the output data structure. This data are written to the output interface and received by a subsequent recipient.

In this work, the SystemC modules were defined by using a template that comprises two files whose structure is shown in Listings 5.1 and 5.2. The encapsulation process was carried out as follows.

1. The 'to-be-encapsulated' code was copied and used to define an internal method of the module.

2. The missing sections of the template file were filled out based on the isolated code (and its dependencies). In particular:

    a. communication interfaces and code to pass (and parse) the input/output arguments were defined,

    b. a wrapper function that invoked the input/output management code and the isolated functionality was created,

    c. the module's constructor was defined in such manner that it initialized module's variables and created (and mapped) a separate thread for the execution of the wrapper function(s),

    d. required files for implementation of the above were added in the file include section.

Requirements of isolation made functions a natural candidate for modules, as their entry points and input/output arguments in the code are easily determined. Thus, first system partitioning into modules was carried out in a top-down fashion, based on software partitioning into functions, roughly representing the steps of software execution flow presented in Fig. 4.2. Nevertheless, this partitioning turned out to be too coarse grained for most of the defined functional groups. In particular, the computationally intensive (i.e. *ShiftAdd* and *holesFilling*) functions required further refinement.

Listing 5.1 SystemC encapsulation template header file for module *MODULE_NAME*.

```
1  ///This file name: MODULE_NAME.h
2  /* ----------File Include Section---------- */
3  #include "systemc.h"  ///< SystemC library.
4  #include "tlm.h"      ///< TLM library.
5  (...)          ///< i.e. TLM namespace constructs.
6
7  SC_MODULE( MODULE_NAME ) { ///< Module definition.
8
9  /* ----------Interfaces Section---------- */
10 // Should correspond to the input/output arguments
11   #ifdef _TLM_ ///< TLM.
12     // Fill with TLM interface declaration.
13   #endif
14
15   #ifdef _PIN_LEVEL_ ///< Pin-level.
16     // Fill with pin-level interface declaration.
17   #endif
18
19   #ifdef _TIMED_SIMULATION_
20    sc_in<bool> clk;    ///< Required clock signal
21   #endif
22
23 /* ----------Internal Variables Section---------- */
24   (...)    ///< Fill with variables declaration.
25
26 /* ----------Functions Section---------- */
27   (...)      ///< Fill with functions declaration.
28   void encapsulatedCode(); ///< Encapsulate code declaration.
29   void encapsulatedCode_wrapper(); ///< Wrapper declaration.
30
31 /* ----------Constructor Section---------- */
32    SC_CTOR(MODULE_NAME):{ ///< Module constructor.
33     ///< Fill with variables initialization
34     (...)
35
36     SC_THREAD(MODULE_NAME_main); ///< Function/thread mapping.
37     #ifdef _TIMED_SIMULATION_
38       sensitive << clk.pos();     ///< Clock sensitivity mapping.
39     #endif
40     }
41 };
```

Listing 5.2 SystemC encapsulation template source file for module *MODULE_NAME*.

```
1  ///This file name: MODULE_NAME.cc
2  /* ------------File Include Section---------------- */
3  #include "MODULE_NAME.h"  ///< Module header file.
4  (...)
5
6  void MODULE_NAME::encapsulatedCode(input, output) //
7  {
8    /* -------Computations ------ */
9    (...)
10 }
11
12 void MODULE_NAME::encapsulatedCode_wrapper()
13 {
14 ///
15   /* -----Variables Definition---------- */
16   (...) /// e.g. allocation of arrays for output.
17
18   /* ----Initialization----*/
19   (...)
20 ///^The above executes only once.
21
22   /* -------Main Loop-------- */
23   while(true){ ///< executes indefinitely.
24
25     /* ----------Receive Data ------ */
26     /// Get the input arguments
27
28     (...)
29     /* ----Execute the Encapsulated Code---- */
30     encapsulatedCode(arguments);
31     (...)
32
33     /* ----------Send Data -------- */
34     /// Return the output arguments.
35
36   } // End of main loop
37   /* -------Memory Clean Up ------ */
38
39 } // End of the wrapper method
```

When deciding on further partitioning of a module, one should consider the tradeoff between small and larger module implementations. The smaller the module, the smaller the workload it encapsulates, and hence, the time needed for processing. This significantly reduces the time necessary for module synthesis, optimization re-spins, simulation, and validation. Nevertheless, even though the overall execution time is lowered, the part of the execution time spent on inter-module communication is higher. Moreover, systems employing communication through shared memories, if composed of smaller modules, tend to require more memory, both, in overall size and number of instances. This is mainly caused by the increased number of shared memories used for data passing, whose size cannot be made smaller than a technology specific fundamental BRAM construct. Also, one should not forget that carrying out the isolation process for each module can add up to a significant amount of men-hours.

Final factor to be taken into account at the time of system partitioning was system's customization. In this work customization is defined as the capability of the system to support a range of super-resolution parameters, allowing their customization at (HL) synthesis time. In our approach this support was implemented by defining a base system organization and a strategy of its adaptation to suit different parameters values at synthesis time. System's performance scalability was planned to be achieved by means of modules replication and require only minor changes in internal code of system's modules. To make this possible, system partitioning and code encapsulation had to be made aware of, and take into account, code dependency on super-resolution parameters. By identifying and taking into account data dependencies at the time of encapsulation, the code exhibiting the same data dependencies could be grouped and arranged in a way that effectively has led to reduction of the number of modules in which data dependencies were present.

Taking into consideration the above, the final methodology used for system decomposition into modules was as follows.

1. For a given function, identify main execution steps and groups of (sub-)functions that carry them out.

2. Evaluate groups' dependencies on the algorithm parameters. If it would relax the dependencies, partition the groups based on the dependencies.

3. For each identified (sub-)functions group:

   (a) *Isolate* the group's code and *encapsulate* it as a SystemC module.

(b) Check if its relation with the rest of the code is unidirectional. If so, pass on to the step 3e.

(c) If the relation is not unidirectional, analyze the possibilities of non-blocking concurrent execution for the bidirectional relationship. If non-blocking concurrent execution is viable, pass on to the step 3(e).

(d) Pass on to the step 1 to start recursive partitioning of the function.

(e) Analyze the created module performance and optimization possibilities. If the performance meets the targeted one, consider this group code encapsulation as concluded. If more groups are to be processed at this level of recursion, pass on to the next group. Otherwise return to the entry point for this recursion.

The presented partitioning scheme was used for all non-iterative functions. Due to their nature, implementation of iterative functions required a different approach. If execution time of a module encapsulating an iterative function surpassed the targeted limits, estimation of time necessary for processing a single iteration was carried out. Only if the resulting time still surpassed the limit, the module was to be partitioned. Otherwise, the module was to be replicated. The iteration time estimation was used to determine the maximal number of iterations that one module could carry out within the time limit. Knowing the number of iterations and the iteration per module ratio, the number of replications necessary for targeted performance was determined and instantiated.

5.3.3 Untimed/loosely timed TLM model

TLM refers to an abstraction level in the description of a system that provides a model of the communications among elements that describe the behavior of the system in a functional, not pin-accurate, way. That is, in a TLM model the focus is on the data that are passed between two modules, rather than on the detailed way the transfer is accomplished or the detailed functionality inside the communicating elements. Since the detail of the timing-accuracy and pin-accuracy level is omitted, these models are fast for system simulation and prove themselves useful to validate the circuit functionality and explore high level architectural options.

In order to create a TLM-2.0 model of the super-resolution kernel from the C model described in the previous section, the basic features of the TLM-2.0 standard were used: initiators, targets, transaction objects and sockets [Gro09]. Initiator modules initiate new transactions, while target modules respond to transactions initiated by other modules. A

transaction object is a data structure passed between initiators and targets. Sockets are used to pass transactions objects between initiators and targets. Transaction objects are sent through initiator sockets in initiator modules and received through target sockets in target modules. Transaction objects are of generic payload type, which is a general purpose transaction type implemented in the TLM-2.0 standard, and feature a series of attributes. For the purpose of creating the core model the following standard attributes were used: command, address, data pointer, data length and response status. The socket interface through which the transaction objects are sent can be blocking or non-blocking. Our model uses the blocking interface, which is associated with the loosely-timed coding style, which focuses on functional execution with minimal or no timing detail. In this case, the model is untimed (as no clock is used) since the goal is the creation of a model whose functionality can be verified with minimal simulation overhead and which serves as the reference model for the subsequent RTL implementation.

5.3.4 ESL refinement methodology

As discussed in the previous section, the functional blocks that comprise the TLM model of the core were SystemC modules with socket interfaces. These socket interfaces are used for data transfer between functional blocks. Within the SystemC modules, the original C code is used to perform the block's function. In order to create an optimized HDL model, a refinement methodology that involves a *modeling* phase, *high level synthesis*, *optimization* and *logic synthesis* is used. This methodology is applied to each of the functional blocks, which were refined, implemented and verified independently. The following subsections will discuss each of these steps.

5.3.4.1 Modeling

The ultimate objective of the modeling phase was the creation of a synthesizable SystemC model of a functional block and a corresponding testbench. The system functional blocks were verified independently. The tool used in this phase was the Microsoft Visual studio 2010 with version 2.2 of the OSCI SystemC library [Acc08]. Provision of a synthesizable model can be narrowed down to making sure that the description uses only synthesizable constructs and data types. Nevertheless, a series of additional steps are required in order to obtain meaningful high level synthesis results. These steps are discussed in the following paragraphs. Only after these steps have been implemented, the functional block was verified using the same testbench used to verify the C model. The testbench was modified to use the

same (refined) interface used by the functional block. Verification using this testbench was repeated for each modification introduced to the model.

5.3.4.1.1 Addition of clock and reset signals. The first step on the road to synthesizable description was the addition of input ports for the clock and reset signals. In order to implement the reset signal, a construct of the high level synthesis tool was used. In Agility Compiler the function *ag_global_async_reset_is()* allows the definition of a global asynchronous reset for the whole design. This signal is to be invoked in the constructor of the top level module of a functional block. C-to-Silicon allows to define synchronous reset at (sub)module level using *reset_signal_is()* function. Reset signal constructs are vendor-specific extensions of the SystemC syntax not included in the OSCI standard. Thus, reset signal was not used for simulation with Microsoft Visual Studio which uses the OSCI simulator. In the functional block modules, internal and output signals are initialized outside the thread process loop, while variables are initialized in the module constructor. After adding the clock signal, the thread processes within the modules were made sensitive to the rising edge of this clock signal. This was assured by immediately following the thread to function mapping declaration with the following line of code: *sensitivie « clock_signal.pos();*.

5.3.4.1.2 Refinement to synthesizable communication interfaces. The used HL synthesis tools are not capable of synthesizing the used TLM communication interfaces. Thus, the next step of the modeling phase was the replacement of the TLM socket interfaces with synthesizable SystemC signal channels. In our implementation, the pin-level interface comprises a set of three channels: a request, validate and data signal. Using this interface, the module that transfers data is a *source* module, and the module that receives data is a *sink* module. The sink module asserts the request signal, and waits until the source module is ready to transfer the data. When the source module is ready, it asserts the validate signal, meaning that the data are valid. When the sink module detects that the validate signal has been asserted, it de-asserts the request signal and reads the data signal. Likewise, when the source module detects that the request signal has been de-asserted, it de-asserts the validate signal. The validate signal can remain asserted for more than one clock cycle if multiple data words need to be transferred. In this case the source module will not de-assert the validate signal when the request signal is de-asserted, only when the transfer of all the data words is complete.

Each TLM-2.0 socket interface of the functional block was replaced with this request, validate and data signal interface. If the module acted as a source module (it had an output

socket) the request signal would be an input signal, while the validate and data signals would be output signals. If the module acted as a sink module (it had an input socket) the request signal would be an output signal, while the validate and data signal would be input signals.

When a functional block acted as a sink to receive data, the data word or words received would be stored in an input structure. Each input data port had an associated structure, as data reception happens in parallel using point to point interfaces. The structures were used in a way that permitted only one write per cycle. Likewise, when a functional block acted as a source to send data, data words were read from output structures and sent through the corresponding interfaces. (Memory related implementation challenges are the topic of Section 6.3.2.)

5.3.4.1.3 Introduction of timing information. Although, synthesis is possible without doing this, the entire core module would be implemented as a combinational circuit. For such a case area requirements of the module usually exceeded the used device FPGA resources. Thus, the next step performed in order to prepare the SystemC module of the functional block for the initial high level synthesis, was the addition of timing information to the computational part of the module. Adding timing information involves adding SystemC *wait()* statements to the code. (Cadence C-to-Silicon offers an option for automated *wait()* statements introduction.) The pin-level interface used to replace the TLM-2.0 socket interfaces was already cycle-accurate as it required introduction of the necessary timing information so that the input and output interfaces could work as described earlier. Therefore, in this section we will focus on the computational part of the modules.

Several factors must be taken into account when adding *wait()* statements to the code. Every *wait()* statement added to the code means that the functional block will require an additional clock cycle to carry out its processing, as the processes are sensitive to the rising edge of the clock (the common case) and a *wait()* statement means the processes will wait for the next clock cycle and therefore data will be registered. Therefore, the number of cycles required to process a macroblock can be controlled directly, with the addition or removal of *wait()* statements. Also, since the operation frequency is determined by the critical path of the design, code lines with many arithmetic or logic operations done serially were split by adding *wait()* statements and creating intermediate registers. Results of the synthesis encouraged limiting the number of arithmetic and logic operations done in between two *wait()* invocations (representing one cycle) in order to better utilize the hardware resources of the targeted device.

5.3.4.2 High level synthesis and optimization

Once the functional code of an module was modified during the modeling phase and made synthesizable, the HL-synthesis tool was run next in order to perform high level synthesis. The HL-synthesis tool was configured to use EDIF as the output format and to target a specific FPGA device, with default optimizations enabled. Once high level synthesis was successfully performed, the tool generated an area and delay estimation summary. Examples of HL-synthesis reports generated by the HL-synthesis tools are presented in Fig. 5.10 and 5.11. This report provided a breakdown of FPGA resources used for each file that comprised a functional block, and the longest combinational paths in the design. We used the number of LUTs required to estimate the area requirements for the functional block, and the (expected) maximum logic and routing delay from flip-flop to flip-flop (critical path) to estimate the frequency of operation. The area requirement was compared to a reference value that depended on block complexity. The results observed for this methodology presented in [TSC+09] suggested that the used logic synthesis tool is able to optimize the critical path to reach the targeted frequency for values up to 150 % of the targeted one. Thus, in the process of refinement, the critical path latency at this stage was allowed to reach up to 150% of the targeted value. If either, the area requirement or critical path were above the permitted values for the functional block, an iterative process involving the optimization of the SystemC code in the editor and the execution of HL-synthesis tool to obtain area and delay results was performed, in order to bring the values obtained in the report as close as possible to the reference values. In this optimization phase, the goal was not only to adjust the frequency but also the number of cycles, so that the performance goal of achieving over 24 frames per second for QCIF video sequences was met, while at the same time satisfying area requirements.

5.3.4.2.1 Resource utilization optimization. In order to reduce resource occupancy, additional changes were made to the code of each functional block in the optimization stage. The most impactful optimizations of area proved to be memory related. In particular, implementation of arrays using device block RAM memories instead of LUTs/distributed memory resulted in the greatest logic utilization reduction.

After the transformation to timed mode, the core module code was simplified by turning multidimensional arrays into single-dimensional arrays. There were two main reasons for this modification. The first reason was that operating with arrays of more than one dimension required nested control flow loops to generate indices for each dimension, and more registers for the index values and address generation, complicating the hardware required

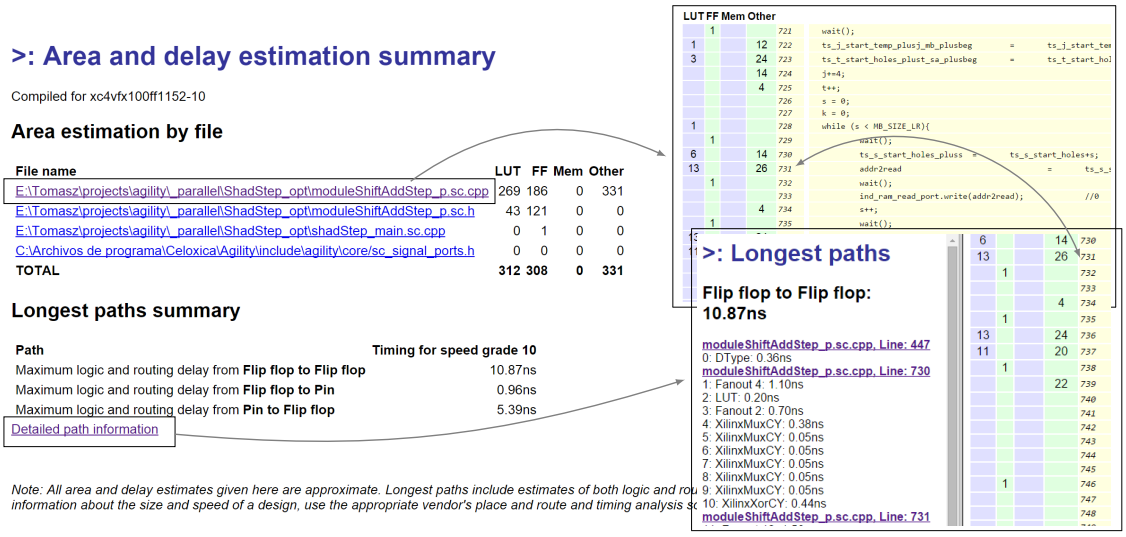


Fig. 5.10 Example of high level synthesis report generated by Agility Compiler.

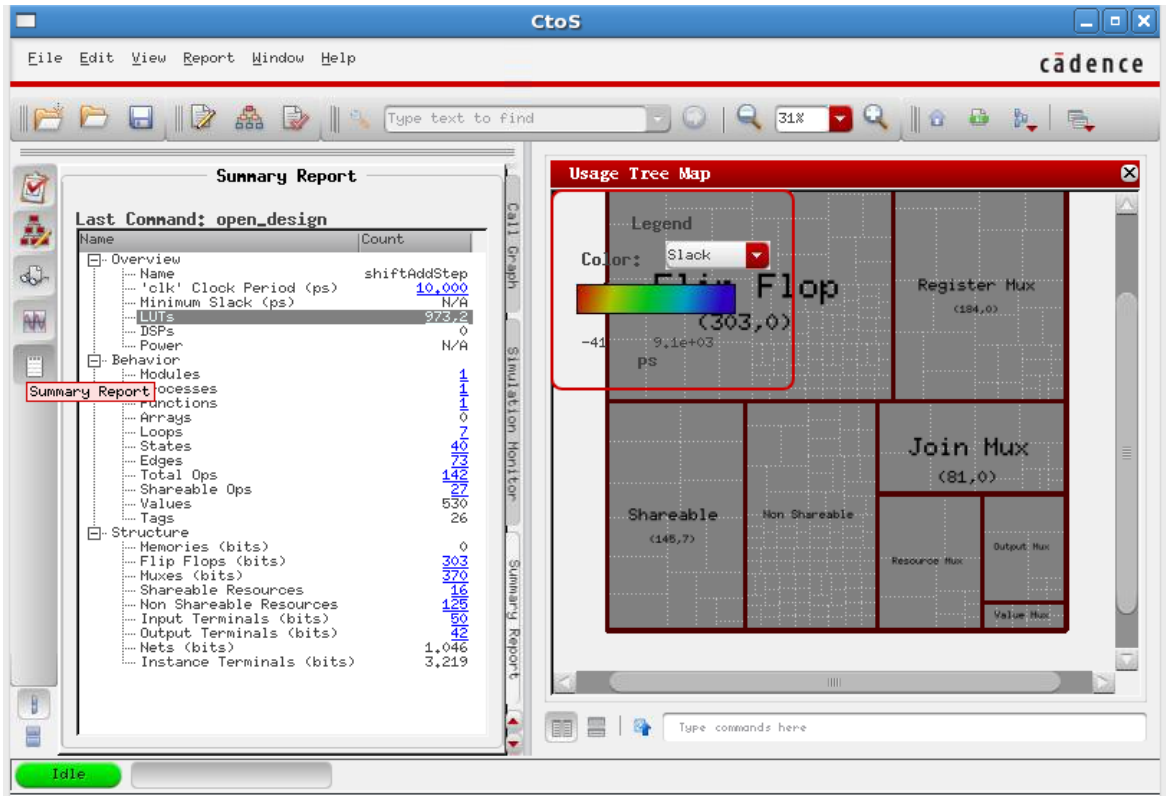


Fig. 5.11 Example of high level synthesis report generated by C-to-Silicon Compiler.

and leading to worse synthesis results. The second reason was the fact that Agility Compiler does not provide templates for multi-dimensional array mapping to block RAM memories. Additionally, it is of particular interest to convert multidimensional arrays that hold constant values to single-dimensional arrays, as this would allow the specification of these arrays as *read only memories* (ROM) in the high level synthesis stage, to further reduce area requirements.

The synthesis tools used required guidance in inferring the use of the targeted device memory resources. In order for Agility Compiler to infer array-to-RAM mapping the mapping has to be explicitly expressed using a dedicated function (*ag_constrain_ram(the_array, read_access)*). As this function cannot be invoked within a (thread) process, in practice memory mapping requires for the array to be encapsulated as a separate module. This process was carried out using the template presented in Listing 5.3. In C-to-Silicon the designer is presented with a table of inferred memories/arrays and has the possibility to choose the preferred implementation using the *Allocate IP* dialog and choosing the appropriate option (i.e. builtin, prototype) from a drop-down list of the action column. For batch execution of synthesis the designer needs to explicitly specify the mapping in the synthesis invocation script by means of the attribute *allocate_builtin_ram /path_to_array/array_name*. It is worth noticing that the basic memory element supported by these tools allows one sample to be read from memory and/or written to memory in each clock cycle. This means that the input and output interfaces will receive or transfer one sample per cycle (after the read address is stable), and that the core module will read from or write one sample to memory per clock cycle for processing. Multiple write ports are not supported by the used tools. Multiple read ports are implemented by replication of the basic memory construct.

All memories have been isolated from their respective functional blocks and encapsulated as an independent SystemC module. The memories used in our design are synchronous and feature one read and one write port. Memory synthesis requires significantly more time to complete than synthesis of logic. Isolation and encapsulation of large memories as separated modules played a significant role in reducing the time of synthesis of the modules carrying out computational part of the core. All instantiated memory instances (implemented as block RAMs) have been grouped into one module instantiated in the top level module of the design.

Other optimizations and modifications of the code aimed at resources utilization optimization were associated with the replacement of the input and output structures of each functional block with input and output memories. Sections of code that read from or write to these structures would instead access the corresponding memory using functions that have

Listing 5.3 Declaration of the array ARRAY_NAME as an isolated module MODULE_NAME_RAM that results in Agility Compiler inferring the use of device BRAMs.

```
1  ///This file name: MODULE_NAME_RAM.h
2  /* ------------File Include Section---------------- */
3  #include "systemc.h"  ///< SystemC library.
4  #include "agility.h"  ///< Agility extensions.
5  SC_MODULE( MODULE_NAME_RAM ) { ///< Module definition.
6  /* ------------Interfaces Section------------------ */
7   sc_in<bool> Clock;
8   sc_in<bool> Reset;
9   sc_out<sc_uint<8> > Output;
10  sc_in<sc_uint<8> > Input;
11  sc_in<sc_uint<3> > Address_Input;
12  sc_in<sc_uint<3> > Address_Output;
13 /* ------------Internal Variables Section------- */
14  sc_uint<8> ARRAY_NAME[8]; // Declare RAM
15 /* ------------Functions Section------------------ */
16 // Define the access interface.
17  void write(){
18    sc_uint<3> Index;
19    wait();
20    while ( 1 ){
21      index = Address_Input.read();
22      ARRAY_NAME[Index] = Input.read(); // Mapped to "WritePort".
23      wait();
24    }
25  }
26  void read(){
27    sc_uint<3> Index;
28    wait();
29    while ( 1 ){
30      index = Address_Output.read();
31      Result = ARRAY_NAME[Index]; // Mapped to "ReadPort".
32      wait();
33    }
34  }
35  /* ------------Constructor Section---------------- */
36  SC_CTOR( MODULE_NAME_RAM ) { ///< Module constructor.
37    SC_CTHREAD( read, Clock.pos() ); ///< Clock sensitivity mapping.
38    SC_CTHREAD( write, Clock.pos() );///< Clock sensitivity mapping.
39    // Declare array as a memory with synchronous read access
40    ag_constrain_ram( ARRAY_NAME, ag_ram_synchronous );
41    ag_add_ram_port( ARRAY_NAME, ag_port_readonly, "ReadPort" )(&
          MODULE_NAME_RAM::read ); // Add and map ports accessed by read
42    ag_add_ram_port( ARRAY_NAME, ag_port_writeonly, "WritePort" )(&
          MODULE_NAME_RAM::write ); // Add and map ports accessed by write
43   ///< Initialise the RAM
44   for( int i=0; i<8; i++ ){
45     ARRAY_NAME[i] = i;
46   }
47  }
48 };
```

been created for this purpose. The number of variables and arrays used for temporal results was reduced, whenever possible, by writing to memories directly in order to reduce area requirements. Also, for separate memory spaces the writes/reads carried out by subsequent modules for different processing iterations could be overlapped.

Final area optimization was obtained after replacement of the synthesizable C-data types and operations, with their SystemC hardware-friendly counterparts. The SystemC types (i.e. *sc_uint, sc_int*, etc.) have been found to yield better high-level synthesis results, as they allow the specification of the exact register widths in bits. Use of multiplications (combined with optional summation) and shifts instead of division and modulo, where possible, led to further optimization as the former operations can be directly mapped to specialized hardware resources/accelerators.

5.3.4.2.2 Critical path optimization. In the original C code, shared variables and arrays were read in many different branches, meaning that the resultant hardware registers would have high fan-out. This applies in particular to the more complex blocks and/or blocks that carry out their processing serially, re-utilizing the same variables in many cases. This leads to an increase in area requirements and makes it harder to optimize the critical path. To alleviate these problems, in some cases (i.e. the modules encapsulating the *ShiftAdd* and the *holesFilling* functionality), further decomposition into sub-modules/multiple modules was required. Finer-grain modules improve synthesis results and times as it is easier for HL synthesis tools to optimize smaller functionality. Also, decomposition lowers the costs of module replication which allows to process the data in parallel which has the potential to lower the overall execution time expressed in elapsed number of cycles.

Once area has been optimized to bring it as close as possible to the reference value, the critical path was optimized. To optimize the critical path, the HL-synthesis tool report is used, as it provides a list of the code lines in the longest combinational path, the FPGA resources in the path and the delays between them. It must be taken into account that optimizing one value might lead to the worsening of another, as reducing the area requirement and obtaining a high operation frequency are frequently mutually exclusive goals. In order to reduce this critical path, the synthesis report was studied to determine which code lines contribute to the critical path. For example, let us consider a line that features a calculation involving four consecutive arithmetic operations which were done combinationally. In the case of the code shown in Listing 5.4, the latency of the combinational path could be optimized by breaking into smaller load and registering intermediate results.

Listing 5.4 Critical path optimization example.

```
/* ----------Original C-code---------- */
ik_ = (((i+minK)*SAMPLE_MAP_ROWS_COLS + base_addr_sm + j1) >> 2;

/* ----------Optimized latency---------- */
  ik = i+minK;
wait();
  ik_sa_aux = ik*SAMPLE_MAP_ROWS_COLS;
wait();
  ik_sa = ik_sa_aux + base_addr_sm;
wait();
  ik_sa_j10 = ik_sa +j1;
wait();
  ik_ = ik_sa_j10 >>2;
```

5.3.4.2.3 Execution time optimization. After optimizing both area and critical path, the number of cycles required to process a macroblock was optimized. In order to determine the total number of cycles that a functional block requires to process a macroblock, code is added to the functional block to subtract the start and finish SystemC simulation times and divide by the clock period, obtaining the number of cycles. Reducing the number of cycles involves removing *wait()* statements that are not in the critical path. *Wait()* statements that represent an unnecessary wait for the next clock cycle, can be removed. Let us consider the code from Listing 5.4. This code was optimized and provides lower latency then the original code. Nevertheless, in this case, the HL synthesis tool could infer the hardware elements to perform the associated operations in parallel, but the hardware elements are used serially due to the addition of *wait()* statements. Using the code presented in Listing 5.5 results in slightly higher latency while reducing the execution time by two cycles. This is possible as the multiply and accumulate (as well as add and shift) operations can be combined to one operation when arguments are shared.

5.3.4.3 Logic synthesis and implementation

Once the functional blocks have been optimized and area and delay estimation values were satisfactory, HL synthesis tool was run again, this time using VHDL as the output format. As for the case when EDIF was set as the output format, default optimizations were enabled. An additional option specified was flatten hierarchy, so that only one VHDL file was generated for the functional block, instead of one for each of the modules that comprise the functional block.

The VHDL code obtained was used to perform logic synthesis with Synplify 2009.06

Listing 5.5 Execution time optimization example.

```
/* ----------Original C-code---------- */
ik_ = (((i+minK)*SAMPLE_MAP_ROWS_COLS + base_addr_sm + j1) >> 2;

/* ----------Optimized latency---------- */
  ik = i+minK;
wait();
  ik_sa_aux = ik*SAMPLE_MAP_ROWS_COLS;
wait();
  ik_sa = ik_sa_aux + base_addr_sm;
wait();
  ik_sa_j10 = ik_sa +j1;
wait();
  ik_ = ik_sa_j10 >>2;

  /* ----------Optimized execution time---------- */
  ik = i+minK;
  ik_sa_aux = ik*SAMPLE_MAP_ROWS_COLS;
wait();
  ik_sa = ik_sa_aux + base_addr_sm;
wait();
  ik_sa_j10 = ik_sa +j1;
  ik_ = ik_sa_j10 >>2;
```

Premier. This tool is configured to use the target FPGA. For synthesis, the *resource sharing* option is disabled, *retiming* is enabled and *no timing constraints* are set, so that the tool will attempt to obtain the highest frequency possible. These settings have been found to give the best results for the project discussed in [TCH+09].

After obtaining logic synthesis results, the EDIF output file generated by Synplify was used to obtain final area and delay results using XST. The tool is configured to use the target FPGA and other options are set to default values.

5.4 Verification methodology

The used implementation methodology, allowed us to take advantage of the multi-tier verification typical for ESL. Our methodology established three levels at which validation was to be carried out, presented in Fig 5.8. These levels correspond to three levels of abstraction defined in our methodology. The first level of verification, is aimed at validation of a functional model implementing the MB-level flow. Passing this level validates the correctness of the MB-level flow description. The second level of validation is meant to assure the functional correctness of the created SystemC description of the MB-level implemen-

tation. Passing this level also validates the correctness of the encapsulation process. The final, third level is used to validate the RTL model and verify that it is meeting the expected performance.

Each of the defined levels of verification involved carrying out simulations and verification of their results by means of comparison of the output against the reference. Independently of the level, the base for comparison were file 'dumps' representing the state of processing at a determined point of execution. Depending on the level of the description being validated different simulation tools were used. The tools used in the simulations and their role in the flow have already been presented in Tables 5.1 and 5.2. The reference image (and thus the output) format has been chosen to be a YUV 4:2:0p 8bit CIF (288x352 pels) sequence (we use only the Y component in comparisons). The LR input resolution depends on the value of the super-resolution zoom (*scale*). The set of sequences used in tests was the exact same set used in the study on reference software quality. This set comprised three sequences, namely, the *foreman*, *mobile* and *paris* sequence, whose description has been already provided in Table 3.2. These sequences have been chosen as they constitute a representative set of possible real life sequences of interest.

5.4.1 Algorithmic model validation

At this stage, system description was represented by definitions of functions, classes and sequential control flow coded in C language. Verification required the simulation data of the frame-level and MB-level software implementations. These data, once obtained, were compared. In case of output inconsistency, the output produced by each of the functions of the MBL implementation was revised and compared with its FL counterpart output. Once the whereabouts of the location of the occurrence of the first point of error were roughly identified, the cause was determined using lock-step run-time debugging. In some cases, debugging sessions of both implementations were concurrent advanced in lock-step, allowing for the values being computed to be investigated and compared at run-time.

The introduction of the frame-to-MB adapters allowed to seamlessly substitute the frame-level implementation of the super-resolution kernel with a MB-level one. This facilitated the re-use the remaining code of the frame-level implementation with the code including the functions responsible for input/output load/store, image decimation and block matching being the most important part reused. This led to the setup used at this stage of validation which is presented in Fig. 5.12. As aforementioned the reference image (and thus the output) format was the YUV 4:2:0p 8bit CIF (288x352 pels) sequence. The LR input res-

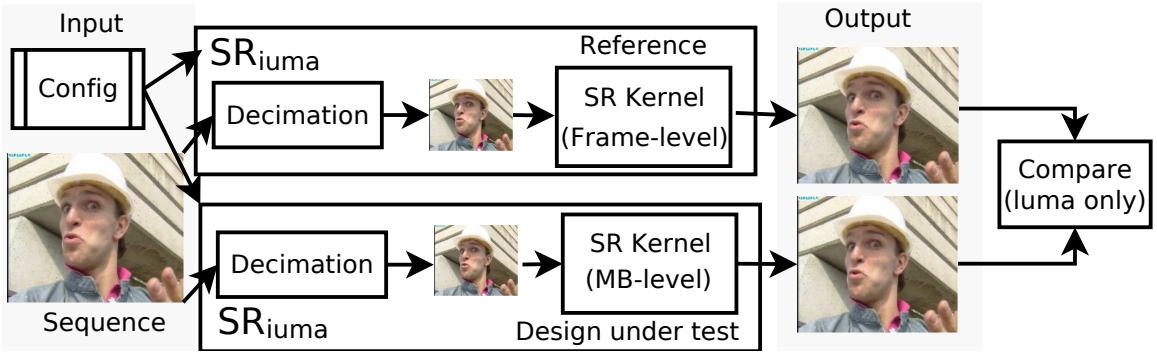


Fig. 5.12 Verification data flow used to assess the correctness of the implementation.

olution depended on the value of the super-resolution zoom ($scale_{sr}$). The validation and verification tasks were carried out in the following steps:

1. First the YUV 4:2:0p 8bit low resolution QCIF and 72x88 pels sequences were obtained from the CIF reference sequence. In order to do so, the CIF sequence was loaded, and decimated, forming a QCIF (or 72x88 pels) sequence that was then stored.

2. The LR sequences were used as input for the reference software implementation, resulting in CIF YUV 4:2:0p 8bit output. This output was considered as the expected output and was used in verification of the algorithmic model of the MB-level implementation.

3. The LR sequences were used as input for the tested system model. The execution of the model was simulated resulting in CIF YUV 4:2:0p 8bit output.

4. The output luma pels of the implementation being tested and the reference software pels were compared. If the observed results were identical for each and every pel and for all frames of the obtained sequence, the verification iteration was considered successful. In comparison we treat both images as lexicographically ordered sets containing all pels in the frame. Only after the outputs generated by all simulations were identical, the current system model was considered as verified.

5.4.2 ESL verification

Once the functional correctness of the C code of the MB-level software implementation was validated, SystemC description of that system was prepared. The C code used to carry out the super-resolution process on the luma component was encapsulated as a SystemC module representing the super-resolution kernel. This code was to be validated and verified based on

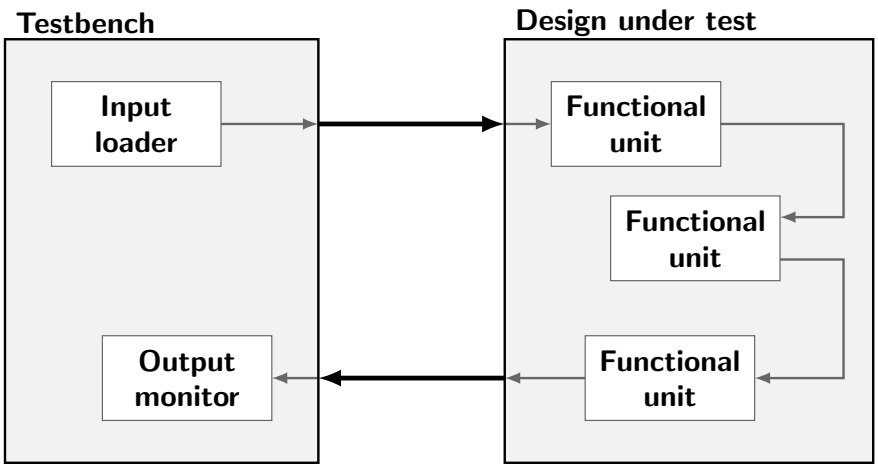


Fig. 5.13 Generic validation setup used for HDL validation.

the files obtained during validation. The validation was carried out following a generic HDL validation setup. In this setup, presented in Fig. 5.13, at the highest level of organization, the system comprises two modules, namely, the module being verified and the *TestBench* (*TB*) module. The former is usually referenced to as the *Design Under Test* (*DUT*). The latter encapsulates the rest of the code used in validation. It is important to notice that the testbench is not required to be synthesizable at any point in the development.

In our case, the testbench code was responsible of: (i) loading the motion estimation metrics, input pels (macroblock and search area), and design configuration, (ii) their propagation to the DUT, (iii) reception and storage of the data produced by the DUT, and (iv) monitoring of DUT execution with (optional) provision of real-time feedback to the designer. Most code implementing this functionality was copied from the C code used in functional validation of the algorithmic model. In order to reduce the simulation time, the block matching was not implemented in code but emulated by loading the ME metrics from the file obtained from the MB-level algorithmic simulations. The output obtained during the step 3 of the algorithmic model validation was considered as the expected output and used in verification of the ESL and RTL models of the MB-level implementation. The rest of the changes were required in order to adapt to the communication protocol using the TLM/pin-level interfaces.

Debugging of the design was based on storing and revising intermediate results of the processing. In order to do so, additional output interfaces were instantiated through which data collected at so called '*validation points*' were transferred to the testbench. In most cases, the validation points were placed in between subsequent modules, collecting the data

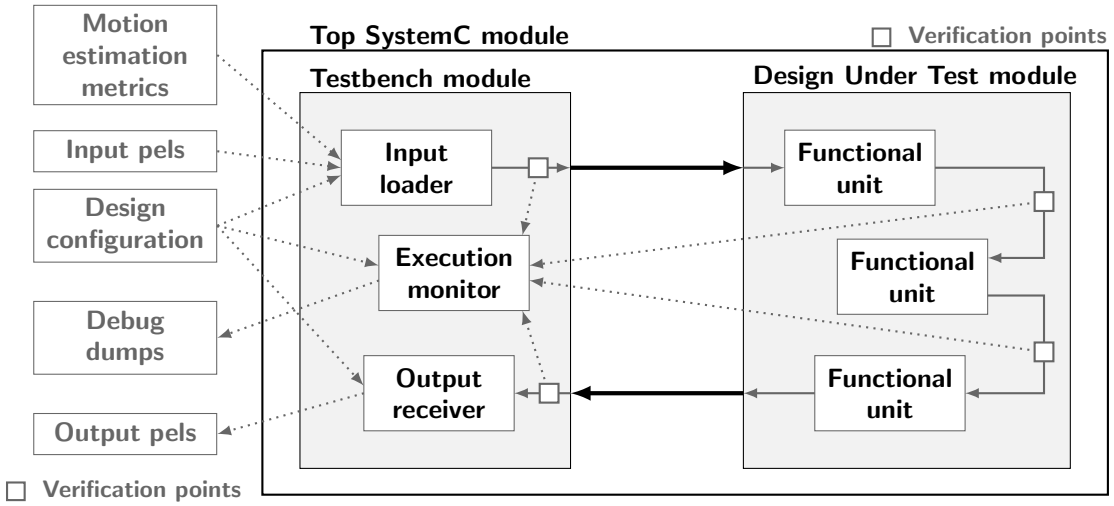


Fig. 5.14 Example of the distribution of validation points in the used ESL level validation setup.

transferred from one onto the other. These points were implemented by replication of the transfer channels to the external interfaces. In some cases, in order to track the progress of module's internal processing and gain access to its local variables, validation points were instantiated as a part of the module. Implementation of these points required (i) definition of additional communication interface in the DUT's sub-module and their connection to the external interface of the DUT, and (ii) introduction of additional code to manage the communication over the instantiated interface. The code implementing the validation points in the testbench and DUT modules was placed in between the `#ifdef (...) #endif` compiler preprocessor statements. This allowed the code to be ignored by the compiler during HL synthesis. The receiving end of the transfers from validation points was the monitoring functionality implemented by the testbench. Once received, the data were checked to assert correct execution, provide feedback to the designer console, and/or store the received data as file dumps. A high level view of the used validation setup after introduction of validation points is shown in Fig. 5.14.

In case of observed inconsistencies the debugging was carried out as follows:

1. The generated inconsistencies were analyzed in order to approximate the time of their first occurrence, system state at that time and possible whereabouts of the point of error in the data flow.

2. The data loaded and transferred to the DUT were revised and compared with the expected ones.

3. Validation points were introduced in between the modules indicated by the results of the first step. In case of no indications of the whereabouts of the point of error, the validation points were activated starting from the last in the flow in an effort to identify the point of first noticeable inconsistency.

4. Additional validation points were added in the execution flow of the module where the first inconsistencies were generated. The execution data captured at validation points were being compared with the expected ones and the design was being modified to assure consistency of these data.

5. If the modifications of the module resulted in DUT output consistency, the functional correctness of the module was considered as attained. Otherwise, the data from the validation point located in between of modules were revised, starting from the point corresponding to the output of the module recently modified. Once the first inconsistencies were observed, procedures from step 4 were carried out for the module responsible for generating the data.

5.4.3 RTL verification

Once verified, the ESL description was made synthesizable and used as the input in the HL synthesis. The generated system description was the RTL description in VHDL (or EDIF). In order to validate the correctness of the HL synthesis translation, RTL simulations were performed. The above-presented ESL validation setup was (re)used also in RTL simulations. This was made possible by means of mixed-language simulation in which VHDL design units were instantiated into the SystemC design.

In order for the HDL entity to be correctly instantiated in the SystemC code, the former has to: (i) be declared as a foreign module, (ii) have its interfaces bound to the HDL implementation, and (iii) be instantiated in the SystemC source. The first two of these tasks are implemented using a header file, referenced hereafter as the module's *wrapper*. The HDL module instantiation in the SystemC source is carried out by referencing the declared wrapper's constructor. Internally the constructor instantiates a foreign module and links it to the appropriate HDL entity (specified using constructor arguments (Cadence) or hardcoded into the constructor definition (ModelSim)).

Both of the used flows provide tools allowing automated creation of the foreign module declaration. Using ModelSim console a foreign module declaration can be generated using invoking `scgenmod` MODULE_HDL, where MODULE_HDL is the HDL entity name.

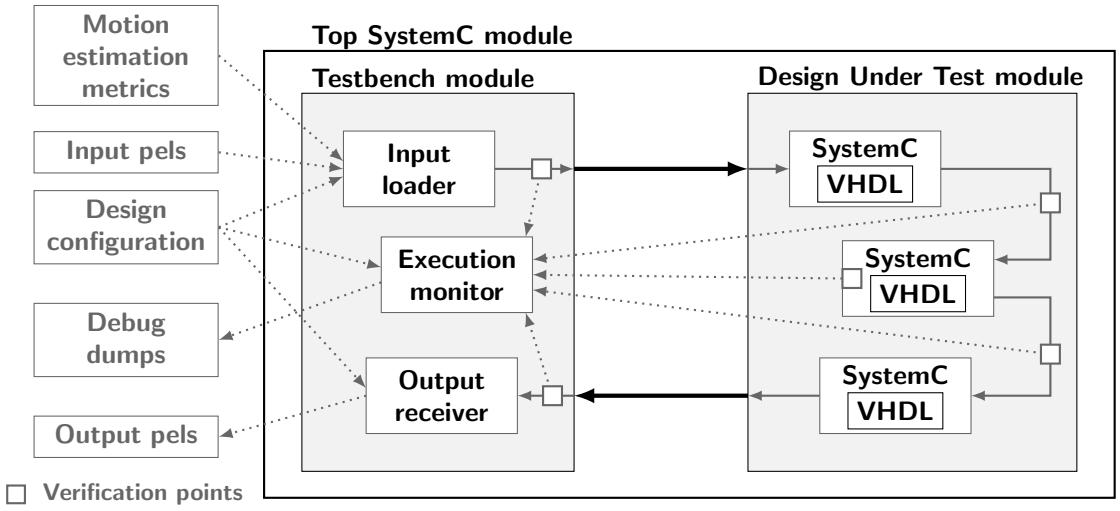


Fig. 5.15 High level view of the setup with SystemC wrappers used in RTL level validation.

This command execution results in the foreign module declaration for the specified entity being written to the ModelSim console. A generic example of a wrapper created using the `scgenmod` for the VHDL entity defined in Listing 5.6 is presented in Listing 5.7.

For the Cadence-based flow, verification wrappers are automatically generated after a successful HL synthesis. The wrapper is written to the *model_dir* specified in the simulation configuration. The filename of the wrapper is *module-name_ctos_wrapper.h*. The wrapper can be recreated by using the completing the *Generate Verification Wrapper* dialog invoked in CtoS GUI. Wrappers used by CtoS have similar structure to the ones used in ModelSim. However, CtoS wrappers contain additional code providing additional functionality that significantly facilitates the verification process. Most importantly, in addition to the design under test (DUT), the original SystemC model can also be instantiated as a reference model (REF) inside the wrapper allowing cycle-by-cycle comparison of the DUT output and the original SystemC model to be carried out.

In the described case, wrappers implementation required only minor modifications, namely, changes resulting in inclusion of the wrapper declaration header and passing the correct instantiation arguments (full name of the HDL entity and verification arguments (Cadence only; described in [Cad10])). A generic example of these modifications is presented in Listing 5.8. Wrappers deployment resulted in the RTL verification setup presented in Fig. 5.15. Once the HDL description has been compiled and the wrappers readily available, this setup allowed the RTL to be verified using SystemC verification code.

RTL verification was carried out in three stages. In the first stage, the whole super-

Listing 5.6 A sample VHDL design unit to be instantiated in a SystemC design.

```
-- File: counter.vhdl
-- ---------Entity declaration ---------------
entity counter is -- Pin-level interfaces using HDL types
port( count : buffer bit_vector(8 downto 1);
      clk : in bit;
      reset : in bit);
end;
-- ---------Architecture definition ---------------
architecture only of counter is
...
end only;
```

Listing 5.7 SystemC foreign module declaration for the HDL module defined in Listing 5.6.

```
/// File: counter_mti_wrapper.h
/* ----------SystemC module declaration---------- */
counter_mti_wrapper
    : public sc_foreign_module {  // declared as foreign module
    /* ----------Interface declaration---------- */
public: // Pin-level interfaces using corresponding synthesizable types
    sc_in<bool> clk;
    sc_in<bool> reset;
    sc_out<sc_logic> count;
    /* ----------Constructor definition---------- */
counter_mti_wrapper(sc_module_name nm)
    : sc_foreign_module(nm, "work.counter(only)"),
    //^ Link to HDL design: "library.enitity(architecture)"
    clk("clk"),          // Binding of interfaces: SystemC("HDL")
    reset("reset"),      // Type compatibility is required
    count("count") {}
};
```

Listing 5.8 Instantiation of HDL entity using ModelSim/CtoS SystemC wrappers.

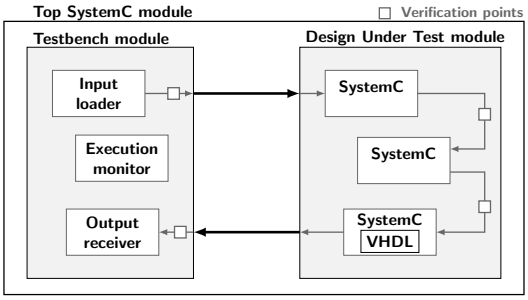
```
#ifdef     MTI_MODEL /* ---HDL instantiation using ModelSim wrapper--- */
  #include "counter_wrapper_mti.h"          // Module as foreign module
  ...
  counter_mti_wrapper dut("dut");           // Wrapper instantiation
#elseif CTOS_MODEL /* ---HDL instantiation using CtoS wrapper--- */
    #include "counter_ctos_wrapper.h"       // Module as foreign module
    ...
    counter_ctos_wrapper dut("dut",(...));  // Wrapper instantiation
#else              /* ----------SystemC module instantiation-- */
     #include "counter.h"                   // SystemC definition
     ...
     counter dut();                         // SystemC module instantiation
#endif
```

resolution core was considered the DUT with only one HDL-to-SystemC wrapper being instantiated. In a progressive manner, each of the DUT's sub-modules was instantiated using a wrapper and its corresponding HDL and verified separately using the testbench of the whole system. In the second stage, instantiation of multiple wrapper modules was allowed. The number of modules whose functionality was implemented in HDL was progressively extended, starting with the module located at the end of the data path. This stage ended with all of the internal (sub)modules of the DUT being instantiated using wrappers. This verification order of module instantiation was meant to increase the effectiveness of mismatches detection (by having HDL DUT output being always the output of the last module) and facilitate faster debugging re-spins (mainly used for the somewhat limited verification options offered by the Agility-based flow). The final, third stage of validation followed the flow of the second stage, with the exception that only one module, encapsulating the functionality of various (subsequent) modules, was instantiated. The modules to be substituted with their HDL description were instantiated as submodules of a newly created SystemC module. After ESL verification, this entity was synthesized and its wrapper used to substitute its SystemC counterpart. The scope of the modules encapsulated by the HDL instantiated in the wrapper was progressively extended till it included the complete functionality of the DUT. An example of the tested system's organization for the above-described stages for the defined RTL validation setup, presented in Fig. 5.15, is shown in Fig. 5.16.

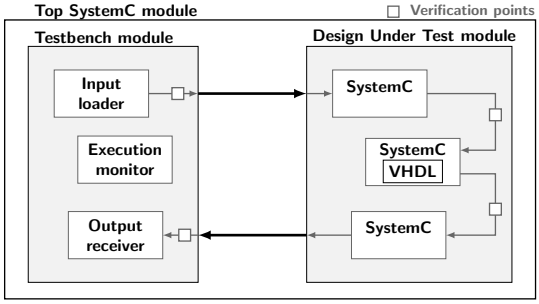
5.5 Conclusions

In this chapter, the used implementation, design and verification methodology has been presented. The established methodology defines three tiers of modeling, namely the functional, ESL and RTL level. These tiers form a hierarchy in which each of the used models is obtained based on the model being one level above it. This renders the models tightly-coupled, facilitating inter-model modifications propagation and code re-use. Even though, the established implementation flow internally distinguishes between design and verification flows, both of these flows deploy the same hierarchy of abstractions.

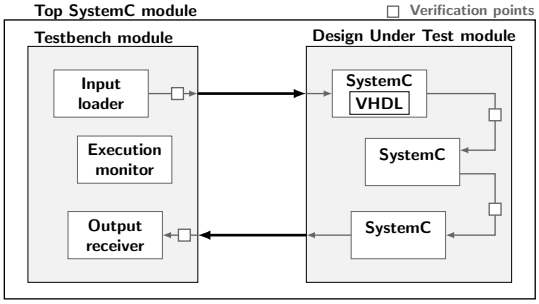
This chapter started with a brief introduction of the basic methodology and modeling concepts considered relevant to the ESL electronics systems design and verification. Next, an overview of the established implementation flow and used toolchain was presented. The tasks and stages of the design and verification flow, as well as the inter-dependencies between these flows, were defined and described. Following, the design flow was described in details. The used models' and refinement steps carried out at each stage of implementa-



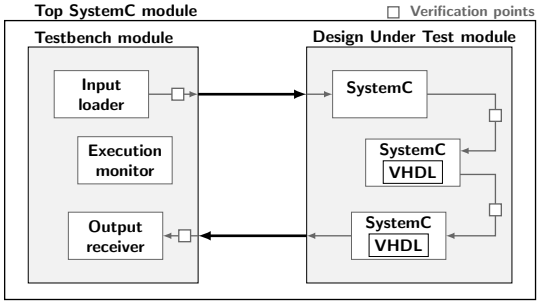
(a) All stages: initial iteration.



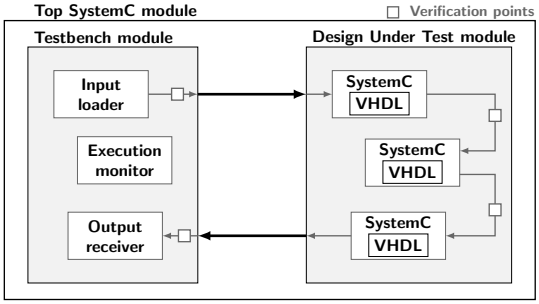
(b) Stage one: second iteration.



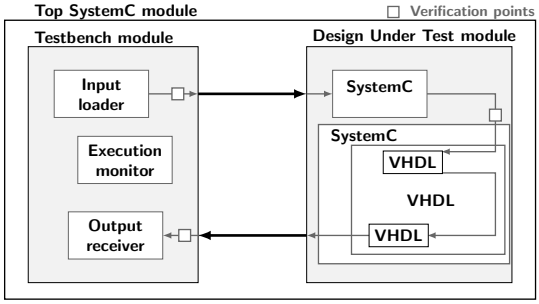
(c) Stage one: third iteration.



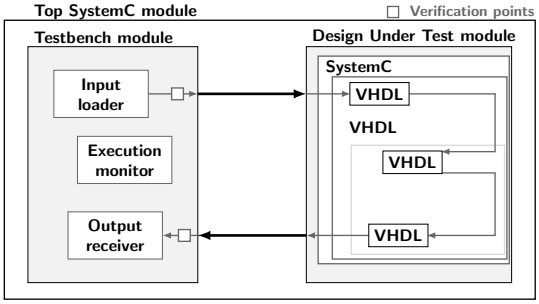
(d) Stage two: second iteration.



(e) Stage two: third iteration.



(f) Stage three: second iteration.



(g) Stage three: third iteration.

Fig. 5.16 Illustration of the defined RTL verification stages for the setup presented in Fig. 5.15: (a) initial iteration (shared by all stages), (b)–(c) iterations of the single wrapper verification stage (stage one), (d)–(e) iterations of the multiple wrappers verification stage (stage two), and, (f)–(g) iterations of the progressive wrapper verification (stage three).

tion were organized and presented in the order that follows the implementation flow stages. Description of the most important stages, that is: modules decomposition, provision of synthesizable SystemC model, and synthesis results optimization, was extensive and provided with real-life examples. Finally, details of the established verification flow were presented. For each of the defined verification levels a short description of the used verification methodology, setup and verification goals were provided. The presented description provided details on the flow of setting-up the verification environment in a way that leverages mixed-language simulation allowing the same environment to be used for both, the ESL and RTL level simulations.

Chapter 6

Hardware implementation

6.1 Introduction

Software super-resolution implementations usually work with all SFW frames available from memory, and output a super-resolved frame only after it has been completely processed. This scheme is meant to minimize memory communications, and is known as the frame-level processing. Memory communications are minimized by coalesced memory accesses, and by high reuse of loaded data. Nevertheless, this approach requires significant amount of memory storage, and thus, results in implementations that tend to make extensive use of external memory. Moreover, when targeting programmable logic devices (PLDs), like FPGAs, high register re-utilization may result in high fan-out. This increases both the logical complexity and the critical route latency.

The SR_{iuma} reference software operated at frame-level. As aforementioned, frame-level processing implementation in FPGA device is troublesome, if not unviable for an implementation that restrains itself from using off-device memory. Due to the high memory requirements implementation of the SR kernel working at frame-level, targeting the available technology and using only on-device memory, was considered. In Section 4.2.2 of this work we have proposed a possible solution to the problem of high memory requirements of the frame-level flow, namely the MB-level flow. The proposed flow internally works with a smaller set of pels, allowing to significantly reduce memory occupancy at the cost of increasing memory traffic. The expected memory reduction and traffic increase associated with the switch to the proposed flow has been theoretically modeled and quantitatively evaluated in Section 3.4.

Hardware implementation of the MB-level flow is the focus of this chapter. The implementation in the targeted FPGA device was carried out using the methodology established

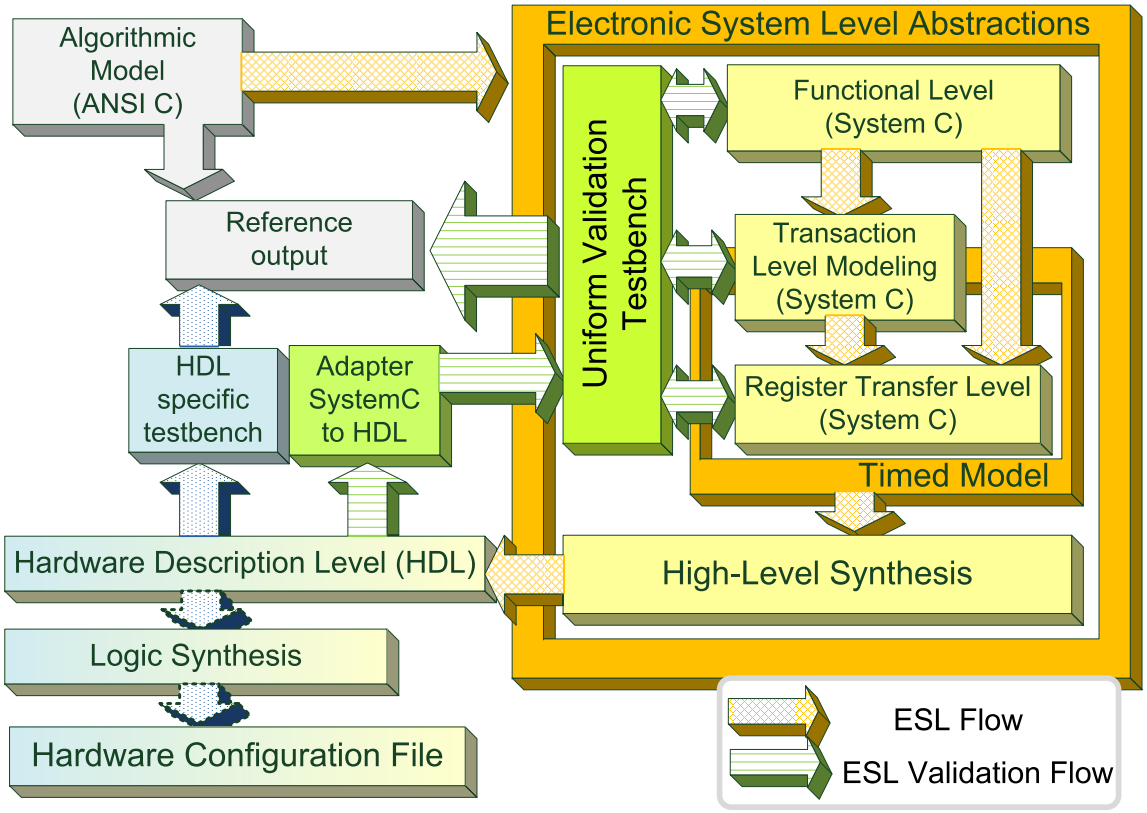


Fig. 6.1 Established design flow for computer aided implementation based on ANSI C code encapsulation and refinement.

in chapter 5. Following this methodology, we have modeled several levels of abstraction before reaching the pin-accurate level used for synthesis. The results of synthesis allowed to determine system bottlenecks and to guide the process of architecture and organization refinement. Reaching real-time performance required multiple refinement iterations and tackling and solving a number of implementation challenges. Details of these tasks are presented in this chapter.

6.2 Hardware implementation of the NUGP algorithm using FPGAs

The base for the hardware implementation was the ANSI C SR_{iuma} software code. Having available a high level software, the implementation followed the computer aided HDL creation using ESL methodology established in Section 5.2.2. This methodology is presented in Fig. 6.1. The implementation flow starts from the existing code and transforms it into an

ESL description. This description after additional refinement becomes synthesizable allowing automatic HDL generation. For our implementation the SystemC (version 2.2, [Ins06]) ESL description language was used. In our implementation we have chosen to model the system using the following three levels of abstraction: (i) the functional level (untimed), (ii) the loosely-timed Transaction Level Modeling ([Ghe06], [CG03]), and (iii) the timed pin-accurate Register Transfer Level.

Functional level description contains regular software that has been isolated and encapsulated as SystemC modules. This description was used to validate the proposed MB-level flow and to provide a golden model for the TLM and RTL levels of abstraction. This model was the one used for algorithmic optimization validation and evaluation.

The TLM description focuses on accelerating the architecture design space exploration by implementing coalesced function-based transfers of coarse grained data. The TLM level description is used for rapid architecture modeling with minimal time spent on inter-modular communication and synchronization design. The created TLM description in *loosely timed* mode implemented synchronization is based on events and blocking communication over high level interfaces i.e. sockets. Thereby, it is not capable of cycle-accurate execution and should be used only for functional validation of the system. For system's quantitative performance estimation the pin-accurate timed mode was used.

The pin-accurate register-level modeling is characterized by fine grain (pin level) communication interfaces based on basic SystemC types, custom inter-module synchronization, and global execution synchronization with the simulation kernel. This mode offers cycle-accurate execution (in lock-step) allowing accurate system performance evaluation.

The HDL description was generated automatically from the pin-accurate model using the HLS tools. The resulting HDL was validated by means of mixed-language simulation, using the verification setup used for ESL verification.

6.2.1 Super-resolution system overview

The NUGPA super-resolution system at the highest level can be seen as comprising three blocks: (i) the block matching, (ii) the super resolution kernel, and (iii) the I/O management code. As aforementioned, BM implementation is considered out of the scope of this work. For better simulation performance, BM has been emulated by introduction of additional I/O management code that loads motion estimation output from files. As aforementioned, hardware implementation of the BM algorithm to be used with our implementation was being developed concurrently [HN10]. The SRK module comprises the logic that carries out

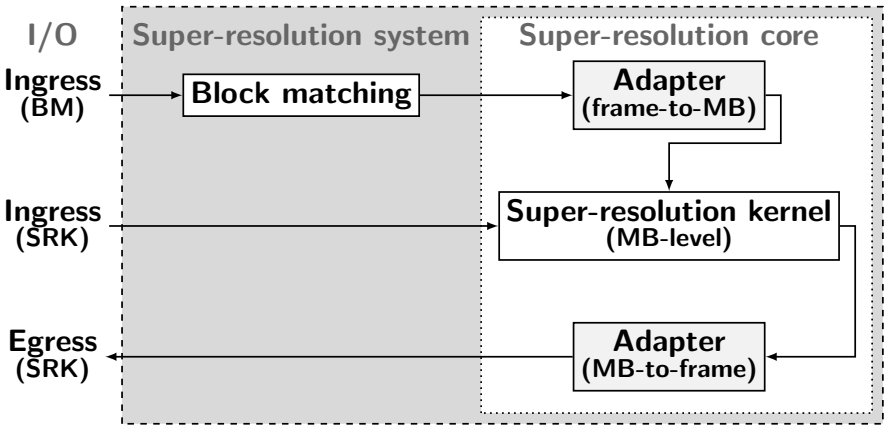


Fig. 6.2 High-level organization of the NUGPA system operating at macroblock-level.

the NUGPA, that is, given the macro-block to-be-processed and the data associated with it (the ME metrics, search areas, etc.), it outputs the MB's super-resolved representation. The original SRK code was transformed in order to operate at MB-level. The deployed block matching algorithm operated in a frame-by-frame manner, meaning that, after the modifications, the SRK code had become incompatible with the rest of the system. To address this issue two adapter modules were designed, one to receive and reorder the output of the BM module, and the other to reconstruct the super-resolved frame from MB pels output by SRK. Implementation of the above-mentioned modifications led to the super-resolution reference code structure presented in Fig. 6.2. Using the nomenclature established in Section 5.4.2, the super resolution kernel and the adapters code formed the design under test, and the I/O management code with BM emulation code was encapsulated forming a prototype of the testbench module. As aforementioned, the implementation of the above presented architecture was carried out following the methodology presented in Section 5.2.2.

6.2.2 Super-resolution core: architecture evolution

The initial functional model of the kernel was based on the decomposition established during the study on the proposed flow's memory occupancy and memory traffic. In that study, the super-resolution kernel comprised the following code sections:

1. The *zeroes2ones* section, which carries out the *zeroes2ones* transformation.
2. The *upHoles* section, which constructs the high resolution grid of a search area.
3. The *upGrid* section, which constructs the high resolution grid of a macroblock.

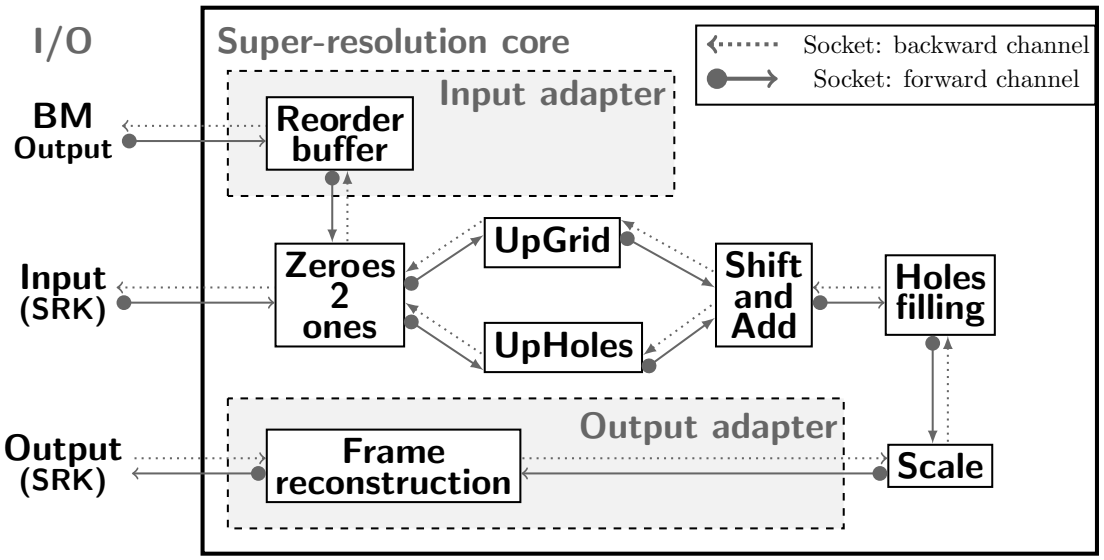


Fig. 6.3 TLM model of the initial architecture of the super-resolution core operating at macro-block level.

4. The *shiftAndAdd* section, which extracts the pels from the created high resolution grid, fuses them, and projects them onto a single high resolution grid of a macroblock.

5. The *holesFilling* section, which estimates values of the missing data of a grid.

6. The *scale* section, which projects a subset of pels from a high resolution grid onto another grid in order to adjust the grid's size to the expected one.

Following the decomposition process presented in Section 5.3.2 a functional SystemC model was created. In this model each of the above-described code sections was encapsulated as a separate SystemC module. Encapsulation of the code into SystemC modules allowed the functional model to be used for system prototyping, design space exploration, and functional verification of architectural changes. In addition, the functional model allowed the impact that the modifications at the algorithmic level have on the system's architecture to be rapidly prototyped and checked.

Once refined and validated, the functional model system was transformed into a higher-accuracy TLM model following the methodology presented in Section 5.3.1. As implied by the name, TLM is focused on modeling (inter-module) communications. In comparison with the functional model, the TLM model allowed to:

- Refine and validate inter-module communications as all data communication is required to be carried out using instantiated SystemC channels.

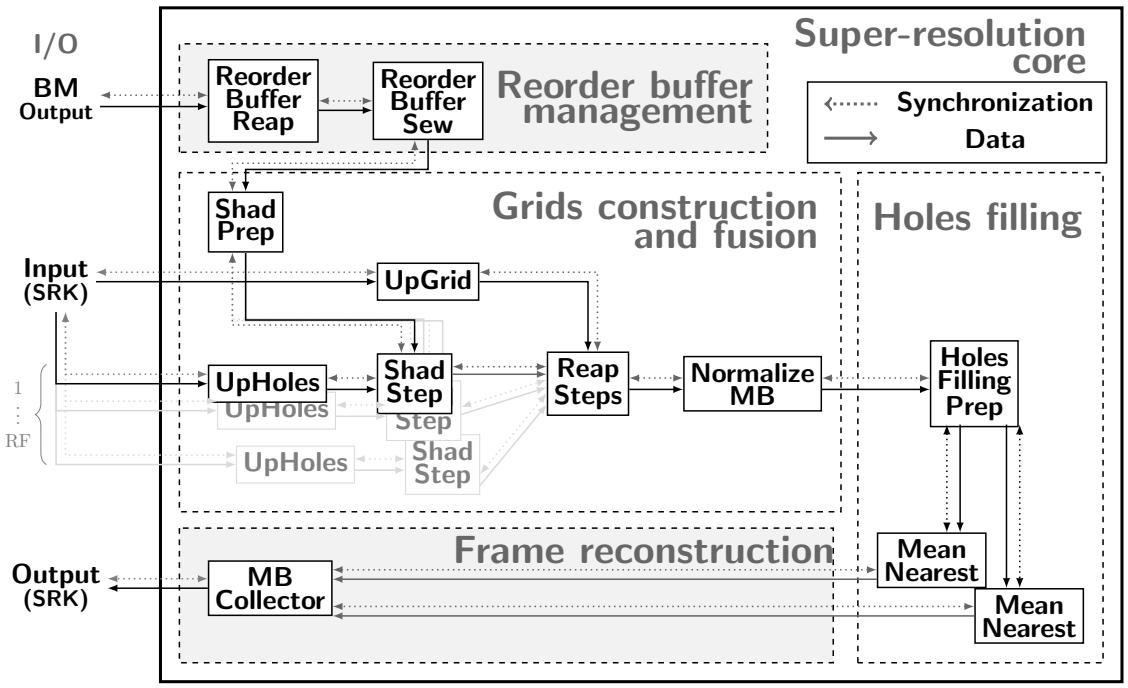


Fig. 6.4 High-level architecture of the pin-accurate model.

- Represent and test more refined memory architecture as at this stage the design memories were no longer modeled using abstract structures instantiated in global memory space, but rather instantiated as separate modules.

The TLM model, whose initial architecture is presented in Fig. 6.3 was used to test SRK's modules parallel execution, inter-module communication and synchronization. Once these aspects of the design were validated, further refinement was carried out in order to obtain synthesizable pin-accurate timed model. This model allows evaluation of the performance and efficiency of the obtained implementation and identify the parts of the design which needed further optimization. During the optimization stage all modifications to system's architecture, organization and inter-module communication scheme implemented at the lowest-level were being propagated to the TLM model. By keeping these models tightly coupled, the lower-level model verification was accelerated by performing partial validation at the TLM level.

The architecture of the timed pin-accurate model is presented in Fig. 6.4. The organization that was validated for implementation is described in the following section. Design of the pin-level model, which when synthesized, results in an implementation that meets the targeted performance required a significant amount of effort. Implementation challenges

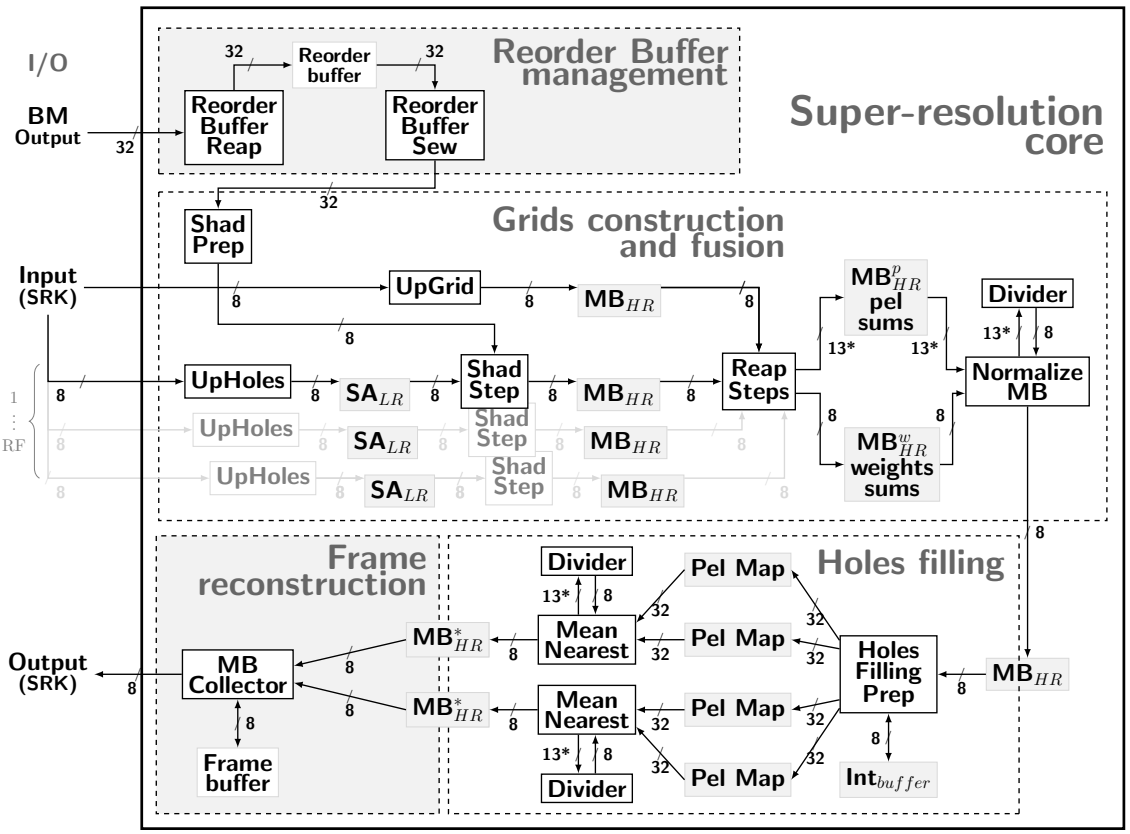


Fig. 6.5 Organization of the pin-accurate model.

associated with this stage of the super-resolution core implementation are presented in Section 6.3.

6.2.3 Super-resolution core: implemented/timed model architecture and organization

The functional architecture of the super-resolution core that meets the targeted performance and functionality is presented in Fig. 6.5. Internally the core maintains the division into three main parts: (i) the reorder buffer management, (ii) super-resolution kernel, and (iii) frame reconstruction. Nevertheless, in order to fit within the established execution cycle budget (computed in Section 6.3), many of the functional blocks had to be further decomposed into smaller modules. The *Reorder Buffer management* task had to be divided in two stages encapsulated as two modules: the *ReorderBufferReap* and the *ReorderBufferSew*. The former module operates at frame-level storing the ME outcome in memories in an order suitable for MB-level processing. Thus, the *ReorderBufferReap* module controls the reception of

the motion estimation metrics, data reordering and storing accesses to the *Reorder buffer* memory. Only once all the reference frames data of the current SFW has been received, the *ReorderBufferSew* is allowed to process them. The *ReorderBufferSew* manages data requests arising from within the super-resolution kernel, carries out data loads from the *Reorder buffer* memory and feeds read data to the *ShadPrep* module of the SRK.

The super-resolution kernel carries out two main tasks: (i) grids construction and fusion, and (ii) interpolation. Thereby, the modules that make up the SRK can be seen as belonging to one of two groups, depending on their tasks. The *ShadPrep*, the *UpHoles*, the *ShadStep*, and the *UpGrid* modules carry out grids construction. Grids fusion is performed by the *ReapSteps* and *NormalizeMB* modules. The process of *holes filling* is carried out by the *HolesFillingPrep* and *MeanNearest* modules.

The input of the SRK comprises the LR pels of the macro-block and its search areas (one per each frame in the SFW), and a set of flags. These flags describe the macroblock and search area location within the frame, and identify the inter macro-block data dependencies. The *ShadPrep* receives the flags and requests the ME outcome (MVs and SADs) from the *ReorderBufferSew* module. Once these parameters are available, current macroblock location within the search area is determined and the fusion weights are computed for each search area from the current SFW. Computed location coordinates and weights, along with the MVs and flags, are passed on to the *ShadStep* modules.

The LR macroblock pels are fed from the super-resolution core input to the *UpGrid* module. Each received pel undergoes the *zeroes2ones* transformation and is stored in the input memory of the *ReapSteps* module. As a result, the input memory of *ReapSteps* holds the HR (4x up-scalled) representation of the LR MB. Having finished its task, the *UpGrid* module sends a synchronization signal to the *ReapSteps* module.

Search areas pels are received by the *UpHoles* modules. Each *UpHoles* module's task is to store received pels in the input memory of the corresponding *ShadStep* module. The *ShadStep*s examine the flags, identify (based on the MVs and coordinates received from the *ShadPrep*) and extract the pels that are used for *SA Grid* creation. The constructed *SA Grid* is stored in the input memory of the *ReapSteps* module. Having finished its task, each *ShadStep* sends a synchronization signal to the *ReapSteps* module.

Having received the synchronization signals from the *ShadStep*s and the *UpGrid* modules, *ReapSteps* starts its operation. The *ReapSteps* module task consists of loading data from grids, fusing them, and storing the outcome in input memories of the *NormalizeMB*. First, the module receives the flags and weights from one of the *ShadStep*s modules. Then, for each HR coordinate it loads pels from the input memories (*HR MB Grid* and *SA Grids*)

and fuses them. At the level of a singular HR coordinate the outcome of the fusion operation are (i) the values of the weighted sum of pels loaded from the input memories, and (ii) the sum of weights of non-zero pels that contributed to the fusion operation. Those values are stored in the input memory of the *NormalizeMB* module. For each HR coordinate the *NormalizeMB* module loads the two values produced by the *ReapSteps*. Then, it divides the weighted sum by the sum of weights and stores the outcome in the input memory of the *HolesFillingPrep* module.

The *HolesFillingPrep* and *MeanNearest* modules carry out the process of *holes filling*. These modules estimate new values for the HR grid coordinates that have not been assigned a value during the fusion process. The outcome produced by the *grids construction and fusion* part of the SRK is passed on to the *HolesFillingPrep* module. The data are used to construct a structure called *pelmap*. This structure is composed of pels of the macroblock currently being processed and some pels of the previously processed MBs. The pels that are used for pelmap construction are determined based on the macroblock location within the frame. The inter macroblock data dependencies are managed by excluding some parts of the pelmap from processing. Pels that are to be processed by the *MeanNearest* modules from a continuous region called the *work region*. The work region spread is identified by the coordinates of its upper-left and lower-right corner. Pels with not resolved inter macroblock dependencies are excluded from the work region. A copy of these pels is stored in local memories until the data dependencies are resolved. Only then these pels are included in the work region and are processed by the *MeanNearest* modules.

Once created, the pelmap is stored in the input memories of *MeanNearest* modules. Based on the received coordinates, each *MeanNearest* determines its particular work region and processes it in a pel-by-pel manner. For each loaded pel the coordinates at which it will be stored in the input memory of the next module are computed. If pel's value is non-zero, it is stored in the input memory of the *MBcollector*, and a next pel is loaded. Otherwise, before being stored the pel is assigned a new value computed by means of mean nearest neighbors interpolation.

The *MBcollector* module receives the super-resolved pels and carries out frame reconstruction. Due to the inter macroblock dependencies, the number of super-resolved pels in the input memories of the *MBcollector* is variable. The exact number of super-resolved pels to extract, their location in the input memory and the frame buffer memory are determined based on received flags and the value of the *scale* parameter. Having received and processed all MB belonging to a frame, the module outputs the content of the frame buffer memories through the core's output ports.

6.3 Implementation challenges

The performance target for the implementation had been set for the system to be capable of super-resolving 24 QCIF progressive formatted YUV4:2:0 frames per second (*fps*). This roughly corresponds to the 24p format [Ski05]. The targeted frequency $f_{targeted}$ was assigned the value of 109 MHz. This value was the one reported in [TSC+09, TCH+09] for an implementation of an H.264 baseline decoder that followed the established ESL design flow and targeted the same family of FPGA devices. The maximal number of cycles $limit_{cycles}$ available for one 4x4 MB processing was estimated by dividing the targeted frequency estimate by the number of MBs (MBs_{nr}) to be processed in one second, as in (6.1). Based on the targeted performance of 1584 4x4 MBs per second and an operation frequency estimate of 109 MHz, the $limit_{cycles}$ value was estimated to be 2867 cycles. The idea of one MB processing encapsulates: internal module processing, synchronization, and inter-module communication.

$$limit_{cycles} = \left\lfloor \frac{f_{targeted}}{MBs_{nr} * fps} \right\rfloor = \left\lfloor \frac{109 * 10^6}{1584 * 24} \right\rfloor = 2867 \tag{6.1}$$

This section presents the implementation design challenges tackled in order to fit within the established execution budget.

6.3.1 Provision of synthesis-time customization

The performance and enhancement capabilities of the super-resolution process depend on the values of the super resolution parameters. Apart from the impact that these values have on the super-resolved image quality, investigated in Section 3.4.2.3, the super resolution parameters also significantly influence the efficiency of NUGPA hardware implementation understood as a function of target device resources occupancy and execution time. There is a clear trade-off between the algorithm performance and the cost of the resulting implementation. Provision of efficient implementation is greatly facilitated if the designer is allowed to balance these trade-offs in order to deploy an architecture that provides the expected performance while maximizing the implementation efficiency in the particular deployment context.

To allow the implementation to fit various execution contexts efficiently we have opted for allowing HL synthesis-time architecture customization. To do so, the design's code had to take into account the ranges of possible parameters values, and their impact on the internal organization, synchronization and performance. Support for a range of execution parame-

ters have introduced further constraints on the design, requiring that: (i) the modules are granted enough execution cycle budget to carry out the processing for the most demanding parameters configuration, that is the one with the greatest workload, and (ii) the module's control flow is aware of the current internal organization. More specifically, the memories, communication ports (pins), processing loops flow (number of iterations and break conditions), workload distribution, and performance had to allow a range of different parameter values. This requirement was met by means of parameterization of the design code, and conditional modules instantiation.

Modules parameterization required evaluation of the control flow dependencies on the parameter values and a synchronization scheme that facilitates customizable modules organizations (variable number of internal modules). Code parameterization is a somewhat straightforward process whose purpose is to assure that the processing (or part of processing) that is found to be dependable on a parameter value has its control flow coded based on macros (#define PARAMETER_VALUE) which actual values are set at synthesis-time. Nevertheless, the process is error prune and its correct implementation requires evaluation of the workload dependencies on the parameters and parallelization possibilities. In our case, the evaluation was based on the memory occupancy and traffic estimation presented in Sections 4.3 and 4.4.

6.3.1.1 Customization challenges.

The SRK core's data path, hardware resources occupancy and attainable performance, are all determined by the run-time values of the SRK parameters. These parameters influence the following three aspects of the SRK hardware implementation: (i) hardware resources occupancy (encapsulates utilization of the targeted device memory, specific accelerators and generic (logic) resources), (ii) execution time (measured in cycles), and (iii) critical path latency. Increase in value of all of the SRK parameters, with the exception of the macroblock width, leads to increase in the amount of workload that requires to be processed (at some point in the data path). If the additional workload is to be processed sequentially the execution time will increase proportionally. Due to higher resources reuse, sequential processing results in additional latency introduced by increased fan-out and higher description complexity at logical level (more variables in equations).

Memory occupancy growth has been already estimated using theoretical equations describing each of the modules memory requirements as a function of SRK parameters. The equations used in our study are the ones that have been established in Section 4.3.1.1. The theoretical estimation did not consider the impact of data segmentation and fragmenta-

tion caused by encapsulation into BRAMs, nor other memory-related implementation challenges. Nevertheless, the carried out study provides sufficient information on how the SR parameters impact the overall memory occupancy. Estimation of the impact that the algorithm parameters have on the device specific accelerators and logical resources is complicated without knowing the actual implementation organization/architecture and synthesis constraints. For the sake of portability, our implementation does not explicitly use any of the hardware accelerators (e.g. Xilinx DSP48 blocks), but leave the room for the synthesis tool to infer its use from generic HDL description.

The impact on the critical path is determined by the introduction of the additional resources and/or sharing of the already instantiated ones. As this process is carried out by the synthesis tools, which optimize the design globally, its outcome cannot be reliably predicted. The use of higher level(s) of abstraction further complicates this estimation. Thus, the impact of the parameters on the critical path latency had to be determined empirically by reviewing the results of the synthesis for different SRK parameter values combinations.

The impact of the SR parameters values on the execution time (measured in cycles) can be, with certain accuracy, estimated and optimized using approximately timed high-level abstraction models.

6.3.1.2 System-level implications of the SRK parameters values

The parameters that impact the execution performance and implementation efficiency of the NUGPA super-resolution kernel are the ones that specify: (i) macroblock size (MB_{width}), (ii) input and output frame size (FR_{cols}, FR_{rows} and $scale$) (iii) internal processing accuracy ($precision_{me}$), and (iv) search area size and reference frame number (SAR and RF). (For more information on these parameters refer to Table 3.1 on page 90.) The impact that the above presented parameters have on the core's organization is discussed below. The presented SR kernel currently allows only one $precision_{me}$ to be specified (=4). Thus, this parameter impact on the design implementation is not discussed here.

- **Macroblock size.**

  In our design, all modules latency and memory occupancy are dependent on the macroblock size. This is mainly due to the fact that the macroblock is the base processing element and most memories and processing loops are defined based on its size. Increasing macroblock size does not increase the overall workload to be processed and does not decrease the cycle/pixel budget. Some variations of system performance is expected due to increased memory sizes (and address space) which could result in

changed memory latency, fan-out, etc. Nevertheless, this could be partially compensated by more coalesced memory accesses and less frequent inter-module synchronization. To recap, changes of MB size have a significant impact on the resources occupancy of all modules and have been handled using parameterized description of memories and execution loops, requiring no changes in the internal organization.

- **Frame size.**

  Number of rows and columns of the input frame has a direct impact on the amount of workload and thus the cycle/pel budget. Moreover, the adapters memories occupancy and access latency is determined by the number of MBs to be processed, which is derived from the frame and MB size. Apart from these dependencies, the core part of the MB-level processing kernel can be considered free of dependencies on the former parameter as it operates on macroblocks in a way that is unaware of the input/output frame size. To recap, frame dimensionality has a significant impact on the adapter modules and has been handled using parameterized description.

  The *scale* value defines the spatial resolution of the super-resolved frame in reference to the input frame size. Its value is used to decide which subsets of the super-resolved samples are to be discarded in order to meet the specified outcome spatial resolution. Customization of this process is provided by parameterization of the decimation loop, the frame buffer definition and the process of pels placement in the frame buffer.

  In the presented implementation, the luma component of the super-resolved frame is fully reconstructed in hardware. This scheme requires significant amount of internal memory for frame buffering. In fact these storage requirements are the ones limiting the maximal supported output resolution to 352x288 pels.

- **Search area radius.**

  The increase of the search area radius results in an increase of the number of candidates for comparison, effectively increasing the number of cycles needed for template matching part of the motion estimation process. For SRK, this parameter, along with macroblock width, defines the search area size and the number of reference pels that have to be received from the input and stored internally. Thus, search area radius has a significant impact on initial (pre-fusion) steps of the kernel, determining their memory occupancy and number of cycles spent on data passing/memory accesses. The increase noticed in these requirements with the increase of the search area radius value is significant. Provision of support for the targeted range of values of this

parameter required modifications at the architectural level. In order to fit within the cycle budget the reception of pels from the input and their extraction for fusion had to be isolated into two stages instantiated by separate modules joined by shared memory. This separation effectively doubled the available cycle execution budget and allowed further optimization (data packing, memory replication) to be introduced. To recap, support for larger search areas required decomposition of the search area-related tasks into smaller chunks instantiated as separate modules. The carried out decomposition allowed to fit within the execution cycle budget.

- **Number of reference frames.**

  For each additional frame in the SFW, new data have to be received (and stored), processed and included in the fusion process. Additional data to be received and processed includes the search area pels and ME metrics. Thus, this parameter impacts modules responsible for performing the pre-processing (reception of input data, parameter estimation and grids creation), and performing the initial stages of the fusion task.

  Reference frames are represented by search areas, which are used to build grids from which the pels to be used in the fusion process are extracted. Due to the elimination of the HR grids, the increase of the number of cycles needed for communication and extraction, rather than the increase of memory occupancy, is the primary factor limiting the customization options. Having the execution time in mind, the architecture was designed in such a way that introduction of additional frames into the SFW could be handled by module replication with conditional instantiation without the necessity of increasing the data path depth or module re-designing. To be able to fit within the assigned cycle budget the processing of multiple reference frames had to be carried out concurrently. As aforementioned, parallelism has been exploited by isolation of the reception and extraction tasks, their encapsulation and (conditional) replication. Increased number of references also impacts the post-fusion normalization process as the pre-normalization sums can hold greater values. This is handled by parameterization of the division module. One thing to notice is that greater values need more bits for their representation, resulting in increased memory requirements and division latency. To recap, customizable number of reference frames has been handled by separation of the pels reception and pels extraction tasks, encapsulation of these tasks into separate modules, and parallelization of processing by replication of these modules (one pair per reference).

The above-presented considerations have led to the creation of a customization-friendly organization in which each SFW frame has a dedicated pair of *UpHoles* and *ShadStep* modules being instantiated. The pels of each search area are received by a dedicated *upHoles* module and processed by a dedicated *ShadStep*. All *ShadStep* modules are synchronized with the *ShadPrep* and *ReapSteps* modules which are aware of the total number of reference frames in the pipeline (one *UpHoles* + *ShadStep* pair per frame in the SFW). The instantiation and synchronization is conditioned on the synthesis values of the number of the reference frames. Memory definition and accesses are carried out using parameterized code and loops.

6.3.2 Memory related challenges

The achievable performance for an implementation in FPGA devices of a computer vision/image processing algorithm is more likely to be limited by the available memory resources than by the computational ones. This is true also in the case of super-resolution. An example of such an implementation is the state-of-the-art implementation presented in [BB08], in which the number of refinement iterations was limited due to insufficient memory resources, leading to sub-optimal quality of the results (measured in PSNR). Even though, in case of insufficient internal storage, the memory can be extended by using external memory. This option still remains a costly solution, in terms of area, power and men-hours required for implementation. Apart from the additional resources, this solution requires significant changes in the data path in order to manage the memory wall problem [NFMM13]. These changes, i.e. vectorization and/or pipelining of computations, introduce another level of complexity to the design and verification process. Thus, memory hierarchy and organization plays a fundamental role in the process of provision of high level of performance, requiring much consideration at the time of implementation.

This section presents some of the memory related concerns that were taken into consideration at the time of implementation of the super-resolution kernel and that have proved to play a crucial part in the process of meeting the performance goals. Among others, the topics that required most attention were the following ones: (1) efficient use of resources for the implementation of arrays, (2) assurance of parallel accesses to memory by subsequent modules which limits the need for serialization of execution, and (3) provision of sufficient memory throughput for the computation intensive modules to meet the targeted memory budget.

6.3.2.1 Resource efficient memory implementation

First synthesis re-spins brought to our attention the fact that the used compilers (Agility Compiler in particular) tend to make an extensive use of look-up tables for arrays implementation. This problem was solved by using compiler specific *Random Access Memory* (RAM) instantiation macros, as described in Section 5.3.4.2.1. The use of these macros allowed the synthesizer to correctly infer the use of available device Block RAM (BRAM) resources.

Block RAMs are commonly available in most of the commercial FPGA devices and their utilization is encouraged. Nevertheless, it must be noticed that RAM macros use limits the portability of system description, as they require that the targeted device offers dedicated on-chip memories and that the compiler supports the used vendor-specific SystemC extensions. In order to relax the latter, in our implementation the used RAM (macros) were constrained to have only one read and one write port per memory. This limitation complicates the design process and limits memory throughput offered by one RAM instance, but has the advantage of facilitating migration to other FPGA technologies as one port memories are always present in BRAM-enabled FPGA devices. Furthermore, one port could not be shared between modules, and BRAM read accesses are characterized by a latency of two clock cycles (one to latch the address, and a subsequent one to latch data on the output). Those limitations required a new memory access scheme and rethinking of system division into modules. In the final scheme, when more than one read per clock was necessary, the memory had to be partitioned and/or replicated. Where possible, memory accesses were grouped, so that they could be carried out in bursts. Burst accesses are characterized by the fact that in each cycle (except the initial cycle of the first access, and last cycle of the last access in the burst) both the address and data latches store new values. The one port per module limitation resulted in enforcement of a policy of *exclusive memory accesses*. In accord with this policy, each module was required either to have an exclusive access (also called two-way exclusiveness) to a memory or to perform only and exclusively either the read or the write accesses (shared exclusiveness).

6.3.2.2 Inter-module communication

In order to take advantage of the parallelism offered by hardware, a system has to be decomposed into modules which have execute in parallel. The task of decomposition has already been described in Section 5.3.2. The task of choosing the inter-module communication scheme that leads to maximization of exploitable parallelism is the focus of this section.

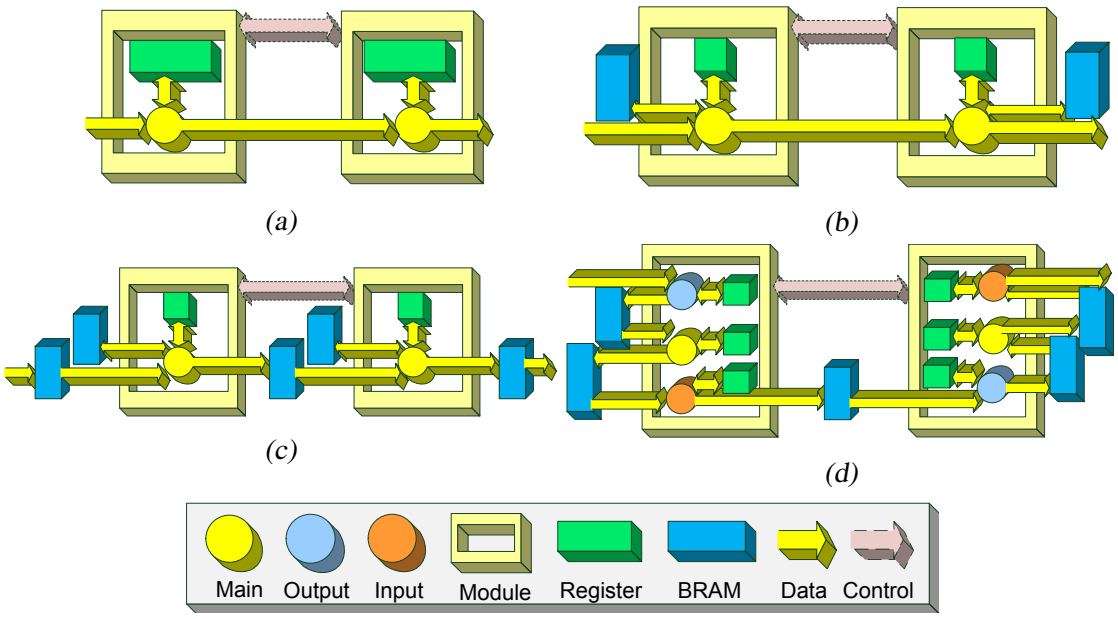


Fig. 6.6 Evaluated inter-module communication schemes: (a) direct transmission with local storage in registers, (b) direct transmission with local storage in memories, (c) indirect transmission with shared input/output memories and, (d) indirect transmission with shared input/output memories and dedicated input/output submodules.

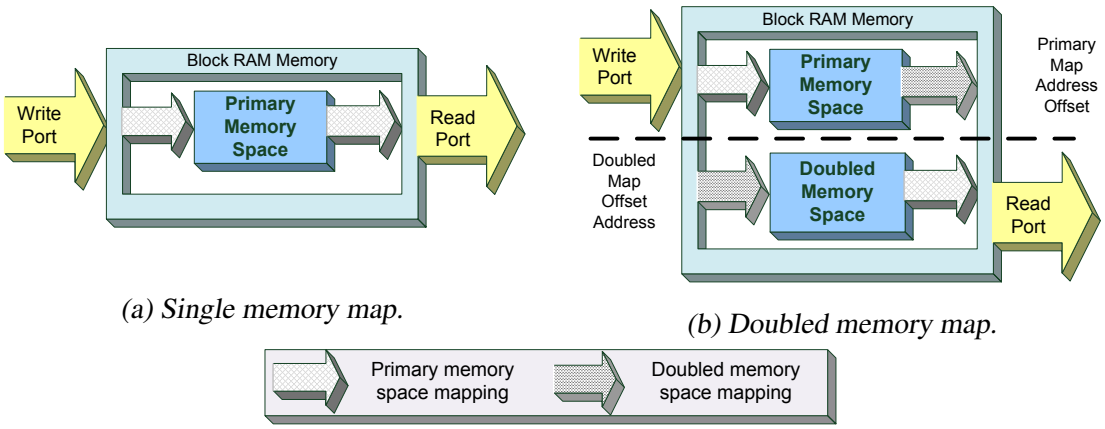
For our implementation, deployment of the inter-module communication schemes that are presented in Fig. 6.6 has been considered. First system implementations deployed a data passing scheme based on direct transactions from source (the *initiator* in SystemC nomenclature) to the sink (the *target*) module. This scheme, shown in Fig. 6.6(a), required long synchronization periods lasting a significant number of cycles that were directly related to the number of data elements to send. This became a probable system performance bottleneck in case most of the internal register arrays of the module, due to prohibitively high implementation costs, were to be eliminated and implemented in BRAM memories (Fig. 6.6(b)). In many cases, received data no longer fit in the reduced register set, and therefore they had to be, either used directly after being received, or stored in BRAM memories. Storing data in memories meant that the data had to be loaded before being used, thereby it effectively doubled the data reception cost for a direct transmission scheme.

Furthermore, for modules implemented using only one thread, direct transmission required that during the synchronization period all other processing of modules participating in the transmission had to be suspended. This problem was solved by the introduction of an indirect transmission scheme based on shared input/output memories (Fig. 6.6(c)). In this line, a shared memory is defined and accessed by more than one module. In this scheme, data are stored before, and read after the synchronization takes place. Thus, duration of syn-

chronization periods are minimized and their duration does not depend on the cardinality of the set of transmitted data.

In order to match requirements of modules with highest workload, a version of the indirect transmission scheme with separated input/output management threads, presented in [TSC+09], could be used. In this scheme a total of three threads are present, as shown in Fig. 6.6(d). The input thread maintained synchronization with the previous module and copied the data into memory accessible by the main thread. The main thread processed the data and stores the outcome of the carried out processing in a memory accessible by the output thread. The output thread stored data in the shared output memory and managed synchronization with the following module (input) thread. Due to the exclusive memory accesses policy, each of module's threads was granted with only one-way memory access rights. Thus, implementations that made use of separate communication threads required dedicated doubled memories used for communication between the input/output threads and the main thread. In this scheme the output thread is granted one-way write exclusiveness, allowing it to write data directly into the input memory of the next module. Once the output accesses are concluded, the thread marks data in shared memory as valid and synchronizes with next module's input thread waiting for data reception acknowledgement. When the input thread reaches the synchronization point, it immediately signals data reception acknowledgement to the output thread, updates data offset value and marks the data as ready for processing by the main thread. Once a reception acknowledgement is received by the output thread, it ends the synchronization period, the thread updates its data offset value and resumes processing.

In order to allow the above-described type of processing, our implementation of indirect communication uses shared memories with separated memory spaces employing the *exclusive* memory accesses policy in order to pass data from the source to the sink module. A simplified architecture of a shared memory with unified and separated memory spaces (hereafter *doubled memory map*) is presented, respectively, in Fig. 6.7(a) and Fig. 6.7(b). It is important to notice that this scheme assumes that the modules relation is of unidirectional type. That is, only one module, of the pair that accesses it, is accessing the write port and the read port is accessed exclusively by the other module. The write and read address spaces of a doubled memory are separated. These address spaces are swapped after each modules synchronization. Having processed one macro-block, the source module stores the outcome in shared memory and synchronizes before switching to the alternative (doubled) memory space mapping. The sink module synchronizes, swaps its memory space, so that it coincides with source's previous memory mapping and starts processing the input data. This scheme



(a) Single memory map.

(b) Doubled memory map.

Fig. 6.7 Defined memory mappings: (a) single and (b) doubled.

prevents data hazards, minimizes the number of cycles spent on synchronization that could be used for processing, and allows simultaneous read and write accesses. Furthermore, as the data are stored in memory, there is no direct transmission and the module is free to load and process the data at its pace.

Moreover, BRAM memories have a fixed (vendor and technology dependable) minimal allocation size, which in most cases was greater than the required memory size, even after doubling. Thereby, memory size doubling did not result in a significant increase of the number of BRAM instances required for implementation. In the case of the targeted Virtex 5 FPGA technology the Block RAMs are fundamentally 36 Kbits in size.

The final version of the system employed the direct transmission scheme only for data reception from the input. The rest of the system employed the indirect transmission scheme. The variant with separated input/output threads for most modules turned out to be an overkill and a waste of memory resources, and was not implemented. The sole exception to this rule was the *MBcollector* which implements a two-threads architecture with the additional thread used to handle the output communications. The memories used for passing data between the modules implemented doubled memory mapping.

6.3.2.3 Memory throughput assurance

In order to increase memory throughput, our implementation uses a mix of memory replication and data coalescing techniques. The former is based on using multiple memory instances storing additional copies of data that are readily available for access. This allows multiple simultaneous loads at the cost of multiplied memory storage requirements. Data coalescing allows accessing several data elements during one transmission cycle by their

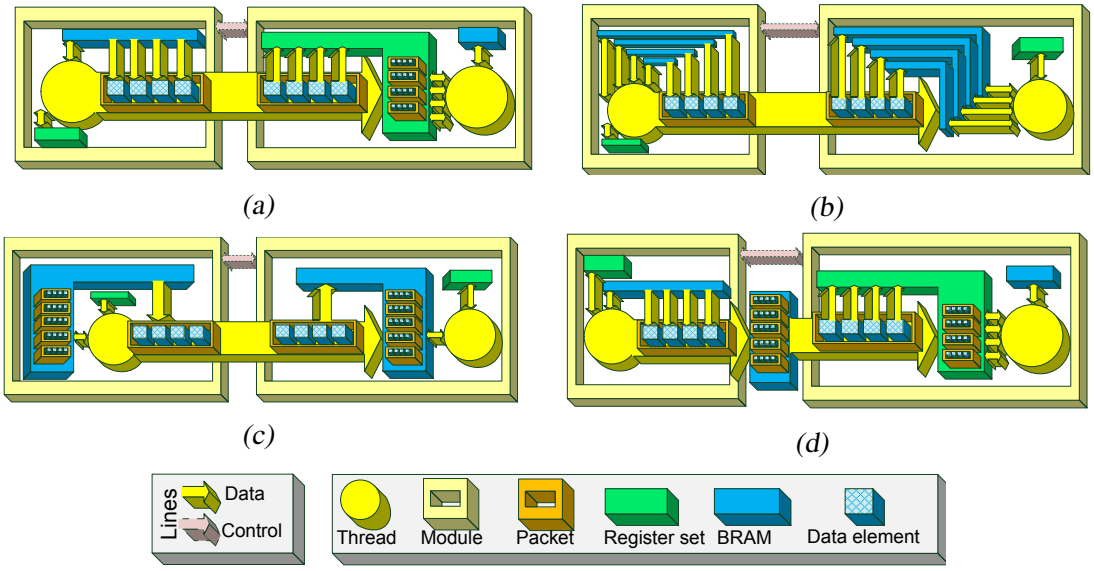


Fig. 6.8 Examples of data coalescing and memory replication: (a) data coalescing without memory replication, (b) data coalescing with four-way memory replication, (c) data coalescing with extended width of memory elements, and (d) data coalescing and data passing using shared memories.

concatenation and transmission over wide (in the sense of number of bits) communication buses. The drawbacks of data coalescing are: increased complexity of data elements extraction, and, for irregular memory accesses, increased complexity of address generation. Moreover, multiple data sources and wide data buses may result in higher resource utilization, fan-out and critical path latency.

In Fig. 6.8 some variants of data coalescing and memory replication investigated for implementation by the source (initiator) and sink (target) modules are presented. For direct connections (Fig. 6.8(a), 6.8(b), and 6.8(c)), that is when the two modules transfer the data over point-to-point interfaces without storing them in any intermediate structures, the target modules have to be capable of receiving and processing the coalesced (or extended) data word in the time reserved for one transmission. As soon as the processing is terminated the outcome is stored in memory, and the registers are used for reception of subsequent data. The storage element can be either the coalesced data word or a singular data element. The choice of storage element granularity is made based on the received data bit width and the number of the words (to be) received. If the data can be processed in that time and the data fit in the register sets they can be received and locally stored in registers. Otherwise, they have to be stored in memory before processing can be applied.

The internal architecture of a module can be characterized as either with single, or with replicated input/output memories. Considering the scenario in which these memories have

the same width, the number of subsequent accesses and cycles used for synchronization needed for the latter scheme is reduced. Thereby, for a coalesced data word comprising four data elements, the initiator module with only one memory of data element width requires four memory accesses in order to create a data word and send it (as shown in Fig. 6.8(a)). If the initiator could access multiple memories, the time needed for data loading would be reduced proportionally to the number of memories that can be accessed concurrently. For the given scenario, for the initiator to be capable of loading a word that comprises four data elements/chunks in time of one memory access, four-way replication, resulting in four available memories, would be sufficient.

Considering the case of direct transmission and assuming that memory access time is equal to data transmission period, in order to send/receive one coalesced data word per data transmission period, the data word has to be created-from/broken-up-into chunks during the cycle budget reserved for data transmission. In order to meed this requirement the total module memory throughput of at least one data word per data transmission period is needed. The above can be achieved by means of memory replication as shown in Fig. 6.8(b) allowing concurrent accesses to replicated memories in order to load all the necessary data elements, which will be concatenated in order to form the coalesced word, during the time reserved for one data word transmission. In this case, on the target side the received word is broken up into into data chunks, with each data element being immediately stored in a different memory instance. Data word creation is performed during the data transmission period, thus, no time dedicated for processing is consumed. Nevertheless, data word creation just before its sending may result in longer latency for data sending, as the data to be sent may need more time to stabilize due to data elements concatenation. Moreover, memory replication multiplies the number of memory resources needed for instantiation.

In order to alleviate the drawbacks of memory replication, a scheme based on memories storing data elements of bit-width equal to the coalesced data word bit width were introduced where necessary. In this scheme, modules also use internal memories/registers of packet width. When used in both the initiator and target modules, as presented in Fig. 6.8(c), this scheme allows loading, sending, and storing of a whole coalesced data word per one data transmission period. The main disadvantage of this scheme is the fact that the coalesced word has to be created and broken up into chunks outside the communication processing, consuming some part of the time that otherwise could be used for processing. It should be noticed that, for sequential data accesses, performance of both solutions, that is, memory replication and memories storing coalesced words with increase bit-width, can be approximated as being equal. Nevertheless, the latter allows significant reduction in number

of memory instances if the total size of the memory after consolidation is smaller than the minimal (vendor and technology dependent) size of a BRAM instance.

Data coalescing plays an important role in provision of sufficient memory throughput where it was most required. An example of this is the implementation of the entity responsible of carrying out the interpolation process (holes filling), presented in Section 6.3.3.2. This process required up to 24 pel values to be loaded for each encountered *hole*. In order to meet the performance threshold pels were concatenated into bundles of four 8 bit values (32 bit data word) and stored in four memories (four-times replication). This organization allowed loading of all the necessary pels in time of four memory accesses, leaving more time for processing and facilitating meeting of the performance requirements.

The introduction of an indirect communication scheme, using an inter-module shared memory as shown in Fig. 6.8(d), solved the problem of long synchronization periods encountered in direct communication. As a result, the time spent on data unpacking became independent of the number of accesses required for (coalesced) data word creation, as long as these words were readily available from the shared memory. Relaxed performance requirements let the initiator module to carry out data word creation at its own pace and, only after all data have been processed, synchronize with the target. A side effect of this is a seamless coalescing scheme in which (sub)modules dedicated to data coalescing could be introduced into the system, in order to carry out the communication and data coalescing in parallel with computations. In cases when the memory throughput offered by this solution is not sufficient, the effective memory throughput could be further increased using memory replication —so that multiple concurrent accesses to the shared memory can be carried out in parallel.

6.3.3 Data dependency in *holes filling*

The amount of the additional information possible to be extracted and fused with the to-be-super-resolved LR pels is determined by the acquisition setup, deployed image registration (motion estimation) performance and the super-resolution kernel parameters values. Some of the crucial parameters are the search area size and number of frames in the sliding frames window (SFW). With the increase of these parameter values, the possibility of encountering additional information increases as more candidate-pels are being used in motion estimation. However, even in case of a favorable setup, the usable information tends to be scarce and its distribution is non-uniform. A common scenario is the one in which for most of the coordinates of the HR representation of the macroblock there's no additional information.

In other words, the post-fusion pel value associated with the HR coordinate is zero. For the sake of satisfactory output quality, these coordinates cannot be left with this value. Thus, for all coordinates with zero value (*holes*) new values need to be estimated. The estimation of these values, in the case of the considered (SR_{iuma}) implementation of the NUGP algorithm is carried out by means of the non-uniform mean-nearest neighbor interpolation. The new value to substitute the one held by a 'hole' pel is estimated based on its neighbor pels' values.

6.3.3.1 Problem statement and implications

The interpolation process requires that a square-shaped neighborhood of the coordinate which value is being computed, be readily available for access. The boundaries of this neighborhood are determined by the value assigned to the int_{window} parameter and frame borders (which effectively crop the neighborhood). The int_{window} parameter specifies the maximal distance (computed as the absolute difference of HR coordinates) from the to-be-filled coordinate that a pel coordinate can have in order for its value to be allowed to contribute to the process of new value creation. A graphical interpretation of the interpolation window and the way it is used to specify the neighborhood spread is presented in Fig. 6.9. In order to determine the set of pels needed for a MB interpolation it is necessary to consider the interpolation neighborhood of all pels that constitute a MB. Thus, the set that contains all pels that can contribute to interpolation of all pels of a MB is defined as a superposition of pels contained in neighborhoods of all pels contained within the MB. As shown in Fig. 6.9, this set comprises all pels of the MB currently being processed and a set of pels belonging to adjacent MBs that are within the distance of int_{window} from the MB borders.

When super-resolution is carried out at frame-level the necessary neighborhood pels are always readily available from the memory as the HR grids represent a complete frame. The switch to the macroblock-level processing leads to elimination of the frame-level memories, introduces data partitioning into MBs and significant changes in the execution flow. All of the aforementioned changes effect in a situation in which only a subset of pels making up the frame is available at a given moment. Thus, for some coordinates not all of the pels required in the interpolation process are available. Namely, a subset of pels from the MB borders that fall within the so called interpolation window requires pels belonging to the adjacent borders of adjacent MBs. For most MBs, the information necessary to carry out the processing for these pels may not be available at the time of reception of this macroblock. As a consequence of not all frame pels being readily available from the beginning

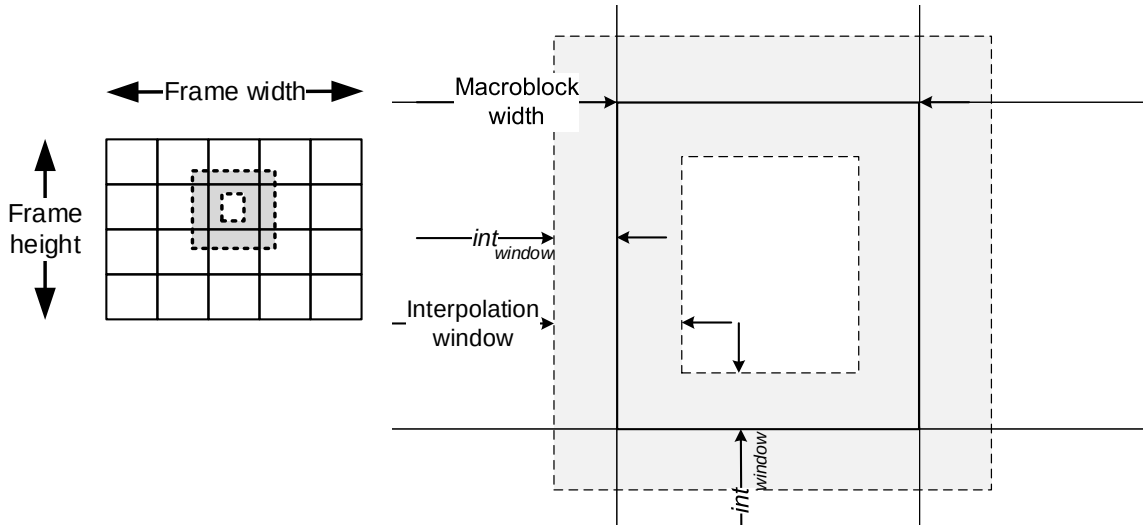
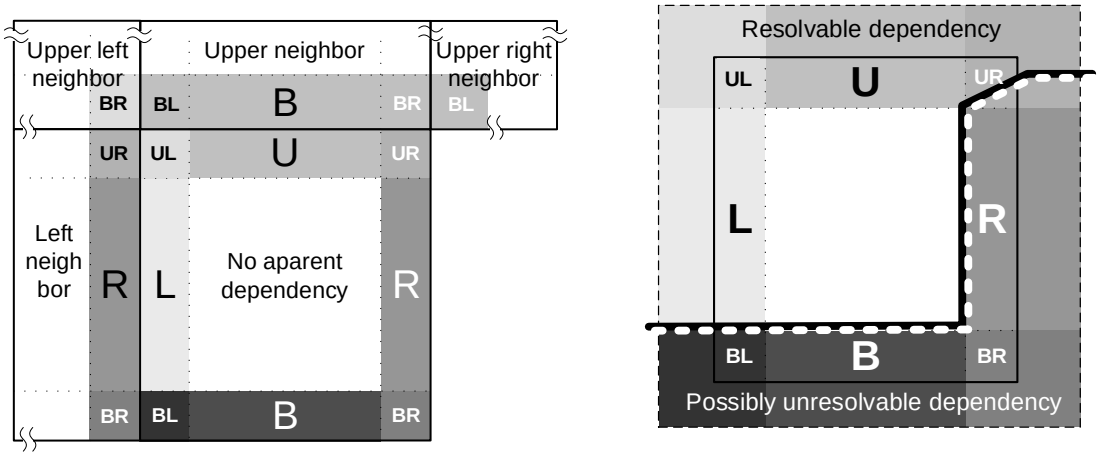


Fig. 6.9 Interpolation window placement on a MB grid with MB boundaries shown.

of the process, data dependencies arise and cause complications in the implementation of the interpolation process. In particular, the rightside columns and the downside rows of the macroblock being processed, that fall inside the interpolation window, cannot be processed as not all of the pels that make up their neighborhoods are readily available.

When processing is carried out at MB-level, the value of the int_{window} and MB_{size} parameters, apart from determining the overall amount of data for which the interpolation process has to be carried out, also determine the current workload, defined as the data for which the processing can be started at the moment of MB reception. When looked at from a different perspective, these parameters specify the pels whose processing does require pels from different macroblocks, and which effectively, present data dependencies and cannot be processed at reception time.

For constant (non-variable) size MBs, a MB can have up to 8 immediate neighbors. The actual number depends on MB's location within the frame. The data that presents MB dependencies can be further divided internally into 8 regions with different data dependencies. A simplified representation of this division is presented in Fig. 6.10(a), where the dependencies are labeled as: U, L, R and B, meaning that they dependent on upper, left, bottom or right neighbor's pels, respectively. Regions marked with labels consisting of two letters present dependencies on pels of two neighboring MBs. The dependencies are mutual, and none of the coordinates that present data dependency can be processed until both pels values are readily available. Considering that MBs are processed in raster scan order, those 8 dependencies (no dependencies being the 9^{th} type of dependency) can be generalized as *resolvable* or *non-resolvable* at the time of reception. The former type comprises data de-



(a) Data dependencies for available pels.

(b) Division into defined regions of dependencies.

Fig. 6.10 Data dependencies regions and types within the interpolation window at the time of a macroblock reception.

pendencies that depend only on previously processed MBs received and can be processed if the data were stored and are available. The latter type comprises dependencies that cannot be resolved at the time of MB reception as they reference data that will be provided in the future and cannot be made available beforehand. One should note that the dependencies are not equally present for all MBs of a frame. Some of the dependencies do not arise for MBs that belong to the frame borders as some of their immediate neighbors do not exist. Assuming raster scan order of execution and that the left and upper neighbors pels can be made readily available, the dependencies of the U, L and UL type are resolvable at the time of MB reception. On the other hand, regions with the B and R dependencies are non-resolvable at that time. When new data are introduced into the system and processed, the non-resolvable, at the time of reception, dependencies of the previously processed MBs may become resolvable. The R dependencies become resolvable as soon as the next MB (right neighbor) is to be interpolated (with the exception of the MBs that do not have neighbor to the right). The B type dependencies hold until a complete row of MBs is processed (with the exception of the MBs that do not have neighbors below, for which the B type dependencies are not present). A generalized view on the division of the interpolation neighborhood into regions based on dependency resolution at the moment of MB reception is presented in Fig. 6.10(b).

An additional side-effect of data dependencies is the necessity of supporting processing workload which cardinality varies along the frame coordinates. The workload increases along with the horizontal and vertical coordinates as the macroblocks from second row/col-

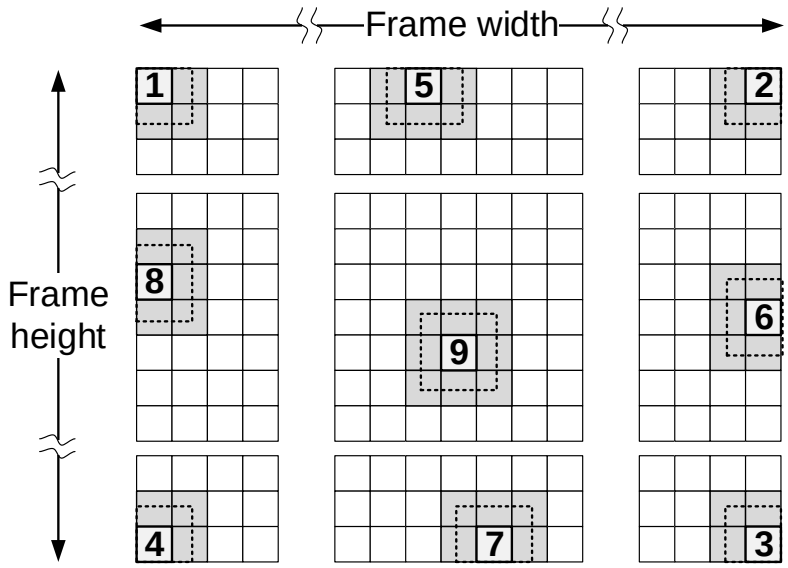
umn process some data of the upper/left neighbors. Macroblocks in the last row/column, due to the lack of the bottom neighbors are responsible of processing more of their data. The least and the most workload has been observed for the first and the last macroblock of the frame, respectively.

6.3.3.2 Implemented solution

During the development the following issues had to be tackled: (1) the value of the int_{window} parameter had to be determined, (2) the actual workload for each MB had to be determined, (3) future processing of the temporarily unresolvable dependencies had to be allowed, and (4) sufficient memory throughput had to be assured.

6.3.3.2.1 Interpolation window size specification. When processing is carried out at MB-level, the value of the int_{window} parameter, apart from determining the amount of the workload of the interpolation process, also determines the set of pels that present data dependencies and the size of interpolation buffers needed for data dependencies management. For our implementation, we have considered the use of three values of the interpolation window, namely the window radius of 1, 2 or 3 pels was contemplated. Based on the observed experimental results for these configurations the interpolation window value has been fixed to 2. Use of this value has an advantage of preventing the interpolation windows of the non-immediate neighbors to overlap due to its value being smaller than the half of the spread of the minimal MB width/height.

6.3.3.2.2 Workload determination. As aforementioned, the amount of data with resolvable dependencies is directly related with MBs placement within the frame. Macroblocks belonging to the frame borders present different data dependencies that macroblocks from further within the frame, and effectively required different amount of workload to be processed. Going into details, MB placement determines the existence (or non-existence) of its immediate neighbors, which specifies which data dependencies hold. In our implementation, the actual workload is being determined based on the distance (in HR coordinates) of the MB borders from the frame borders as this determines the existence of MB's neighbors and the limits of the neighborhood (that cannot extend beyond frame borders). Considering this criterion, 9 possible MB positions within the frame can be distinguished. The distinguished 9 MB positions and the corresponding codifications of them as MB placement flags are presented in Fig. 6.11. Gray squares without a number, centered on the squares that contain a number, represent the immediate neighborhood of the latter. The



MB position	bottom_border_flag	right_border_flag	left_border_flag	top_border_flag
1	*0*	*0*	**1**	**1**
2	*0*	**1**	**1**	*0*
3	**1**	**1**	*0*	*0*
4	**1**	*0*	*0*	**1**
5	*0*	*0*	**1**	*0*
6	*0*	**1**	*0*	*0*
7	**1**	*0*	*0*	*0*
8	*0*	*0*	*0*	**1**
9	*0*	*0*	*0*	*0*

Fig. 6.11 Placements of MBs within a frame that change the number of pels within interpolation window.

MBs located in each of frame corners (marked as 1–4 in Fig. 6.11), belonging to a frame border but not located in the corners (5–8), and not belonging to frame borders (9) are assigned a different type. The distinguished MB positions are codified as a set of four bits, referred to as *MB placement flags*, namely: *bottom_border_flag* (or $flags^{MB}_{bottom}$), *left_border_flag* ($flags^{MB}_{left}$), *right_border_flag* ($flags^{MB}_{right}$) and *top_border_flag* ($flags^{MB}_{top}$). Each of the flags was set if the current MB's border also belonged, respectively, to the bottom, left, right and top border of the frame. The $flags^{MB}_{top}$ and $flags^{MB}_{bottom}$ flags, are set if MB's upper or bottom border corresponded to frame upper or bottom border (the MB is located in the first, or last row), respectively. In a similar fashion, the $flags^{MB}_{left}$ and $flags^{MB}_{right}$ flags are set if the MB was located, respectively, in the first, or last column of the frame. Based on these flags the actual workload was determined and enforced by means of limiting the range of addresses accessed during the execution of interpolation. In our implementation the MB placement is expected to be determined and codified outside of the SRK module and then passed to the super-resolution kernel as a part of the input. The MB placement within the frame was set by the testbench and then propagated along the data path being passed from one module on to next one along with the (corresponding) MB pels.

6.3.3.2.3 Data buffering. Data dependencies resulted in the necessity of buffering the data that cannot be processed at the moment of MB reception until the data dependencies can be resolved. The data that have to be buffered are the ones belonging to the regions with unresolvable dependencies. As presented in Fig. 6.10(b), with the regard to the received MBs, these regions included the int_{window} right-most columns and down-most rows. The right-most columns' pels could be processed with the following macroblock and, thus, are stored in local registers. The down-most rows' pels have to be stored until the pels of the macroblocks below them are available. This means that int_{window} rows of a whole row (plus 1, due to dependency on bottom-right neighbor) of macroblocks have to be buffered. This set proved to be too numerous to be stored in registers and its storage was implemented using standalone module encapsulating RAM instance referenced in this work as the int_{buffer}.

For the used interpolation method, the pels that make up the execution context are the ones presented in Fig. 6.10(a). In our implementation, at MB-level the actual execution of *holes filling* is carried out over a set that comprises all of pels belonging to the execution context that are readily available. This set corresponds to a subset of the set presented in Fig. 6.10(a) which comprises the MB pels plus the pels of the left and upper neighbor that are stored in the buffers. In our implementation this set is stored in the so called *pel map*. A high-level view of the *pel map* organization is presented in Fig. 6.12. Once available, the MB pels are stored in a dedicated pel map area (defined address range). The regions with unresolvable dependencies (labeled as B/BL/BR and R/UR/BR) are stored in registers and RAMs until the dependencies are resolved. At that time, these pels correspond to the upper and left neighbor of the currently processed MB. Thus, when loaded from the buffer they are stored in the pel map in regions mapped to the corresponding regions of these neighbors. The current content of the pel map and the buffers loads/stores are managed based on the MB placement. Actual workload is specified based on the range (or number) of addresses for which holes filling is to be executed. The used address range is limited based on the received MB placement flags. A thing to notice is that the actual pel map implementation uses 1D linear addressing in order to meet the requirements of RAM instantiation.

6.3.3.2.4 Memory throughput assurance. For implemented interpolation window parameter value of 2 ($int_{window} = 2$), up to 24 values can contribute to the process of one *hole* value estimation. For high number of *holes* the sheer number of accesses becomes a likely system bottleneck.

In order to tackle high performance requirements that the *holes filling* process poses and to meet the target performance this process has been decomposed into two stages, one that

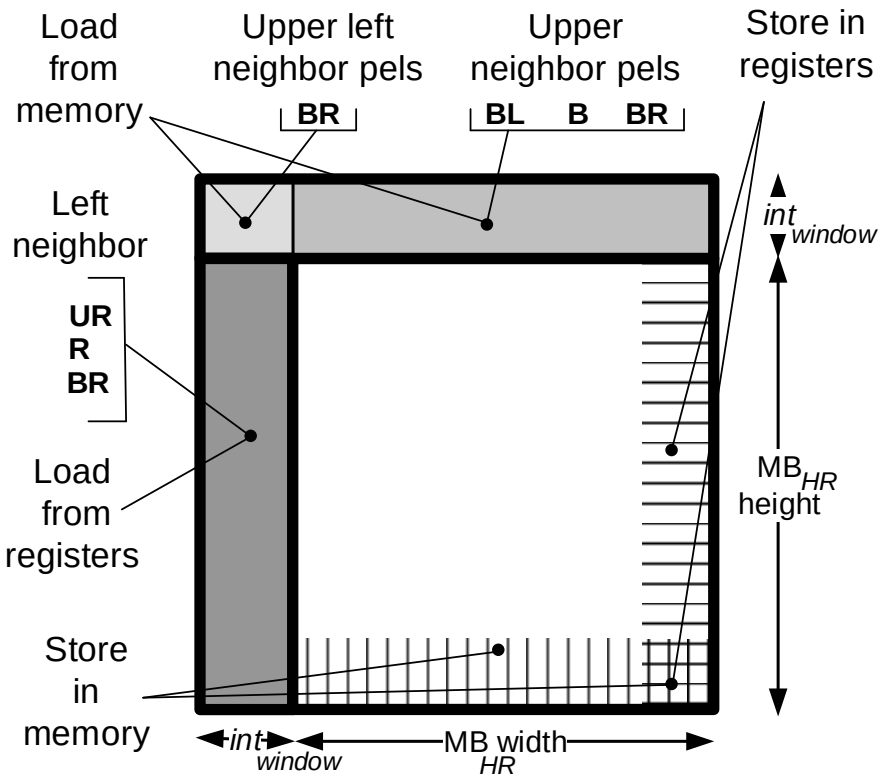


Fig. 6.12 Pel map organization with labels of regions loaded/stored in the buffers.

carries out memory management and the other that implements the interpolation. Each of these stages was encapsulated as a separate SystemC module, the *HolesFillingPrep* and the *MeanNearest* module, respectively. In accord with the scheme presented in Section 6.3.2.2 data passing between these modules was carried out using a shared memory with doubled memory address space. This memory represented the pel map structure and acted as the output/input of the respective modules. The *HolesFillingPrep* module was responsible for loading the appropriate MB context pels, management of the interpolation buffers and data dependencies.

There were many reasons justifying the carried out decomposition into submodules.

(i) Decomposition into two submodules effectively (almost) doubles the cycle budget for the holes filling transformation, allowing to use more cycles for each of the tasks.

(ii) Smaller modules result in better performance of the tools in terms of optimization of the critical path latency (at the cost of overall area growth)

(iii) Decoupling of data preparation from computations lowers the cost (in terms of area) of the computation units replication as replication of data preparation logic is limited.

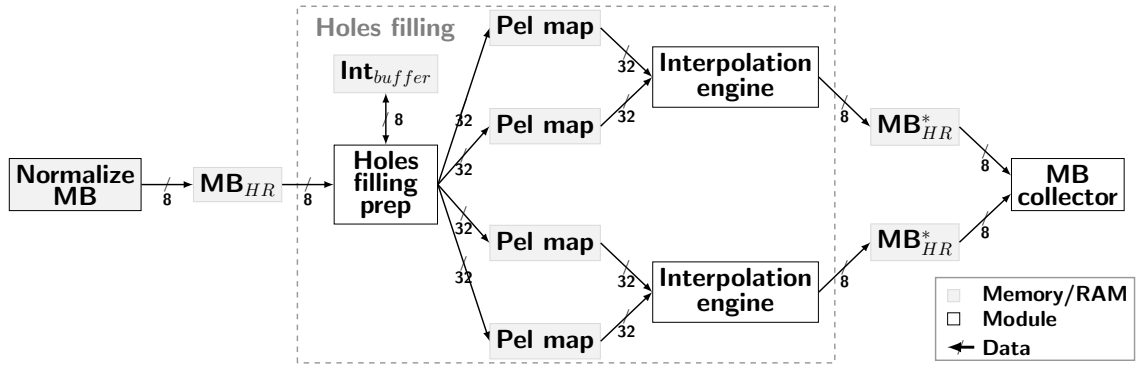


Fig. 6.13 Holes filling implementation after decomposition into submodules, performance and memory throughput optimization.

(iv) Finally, the remaining idle cycles leave some room for customization, in terms of broader ranges of algorithm parameters and/or data, without the need of module's redesign.

Even the decomposition the *holes filling* transformation did not assure that the worst case scenario can be processed within the (now doubled) cycle budget. Additional steps, facilitated by the decomposition were introduced. The *MeanNearest* module was replicated, allowing for the workload to be split between two computational units. In order to increase the memory throughput, additional cycles were used on implementation of data packing. Before being stored in pel map, the received context pels (8 bit values) are packed into one 32 bit word. The pel map memory used for data passing was also replicated, resulting in total of 4 pel maps being instantiated (2 for each *MeanNearest* module). Memory duplication requires additional resources but effectively doubles the overall memory throughput allowing for the *MeanNearest* module to issue up to two accesses simultaneously loading quadrupled elements, effectively accessing up to 8 pels in one cycle. All of the above modifications and tweaks reduced the number of required memory accesses while also reducing the amount of processing that needs to be carried out by a singular *MeanNearest* module instance. The data flow of the *holes filling* implementation after the described modifications is presented in Fig. 6.13.

6.3.4 Variable size of search area

In frame-level processing flow, all pels belonging to a search area (SA) are readily available from the frame buffer of the super-resolution kernel. After switching to MB-level processing, the frame buffer was removed. This resulted in the necessity of search area pels

extraction from a different source and their propagation to the super-resolution kernel.

6.3.4.1 Problem statement

In our implementation, the search area pels are extracted outside of the super-resolution kernel and fed to it as input. Only the SA pels which coordinates are valid frame coordinates are used in the SR processing. The SA pels which coordinates point outside of frame borders are not available and cannot be loaded. The pels that are not loaded can be, either, (i) assigned a value equal to zero (hereinafter *zero padded*) and sent to the kernel, or (ii) not sent at all. The former scheme is easier to be managed as the number of pels that are sent to the kernel, search area grid (storage structure), and the MB upper left corner coordinates within the search area grid are constant throughout the execution, for a given SRK parameters configuration. On the other hand, *zero padding* can be considered a waste of processing power and cycles. This waste could be avoided by sending a set comprising only the available pels, which cardinality is not constant. A side effect of not sending the padding pels is the decoupling of the geometrical relations of pels being send from the actual search area. This leads to the requirement of the actual search area to be reconstructed from the pels bundle being received prior to its use. In both cases, the way in which the frame borders clip the search area and effectively limit the regions of valid data has to be somehow signaled to the core.

6.3.4.2 Implemented solution

In order to avoid the necessity of padding and only transfer the relevant (non-padded) pels, our implementation allows receiving SAs that contain variable number of pels. In order to do so, two issues had to be solved: (i) the structure of the received pels bundle and the limits of valid data had to be signaled to the kernel, and (ii) the placement of the MV reference point, the upper left corner of the MB, within the received search area to be determined in order to allow correct MV processing and data extraction further in the data path. Both of the above are determined by the MB and search area location within the frame and the search area radius.

The number of pels in search area depends the SAR and MB_{width} SRK parameters and the SA placement within the frame. This relation is similar in nature to the problem of determination of the number of neighbors that a MB has which solution has been presented in Section 6.3.3.2.2. The SA placement determination has been solved in the same manner, that is by providing a description of the SA placement codified using four values which are

fed to the SRK alongside the SA pels. An additional problem with the search area is that it could be clipped by frame borders while spanning over a variable number of macroblocks. The number of MBs included in each direction of the SA had to be known, as it is required in order to determine the location of the upper left corner of the MB being processed and the overall limits of valid data in the SA structure, which are defined in relation to that point. To handle this problem, if a SA border points out of the frame limits (and is clipped), the corresponding flag from a set of SA-flags is assigned the number of MBs that fall within the SA in that direction. If search area is contained within the frame (and not clipped) the corresponding flag is cleared (assigned the value of '0'). The MB placement flags are used to detect the cases when the MB border overlaps with the frame border. In this case, signaled by the MB placement flag being set, the corresponding SA placement flag is considered invalid and the span of search area in that direction is set equal to zero. The SA placement and the MB placement flags allowing determining the location of the MV reference point and the limits of valid data regions based on the value of SAR and MB_{width}. These data are crucial in order to correctly compute the linear address that is used further in the data path (in order to access data stored in RAM). The former value is determined by the *left* and *top* flags in accord with (6.2). The latter also depends on the remaining *right* and *bottom* flags, and is determined using (6.3).

$$\begin{aligned} ref_x^{MV} &= \begin{cases} 0 & if \quad flag_{left}^{MB} = 1, \\ SAR & if \quad flag_{left}^{MB} = 0 \; and \; flag_{left}^{SA} = 0, \\ flag_{left}^{SA} \times MB_{width} + 1 & otherwise \end{cases} \\ ref_y^{MV} &= \begin{cases} 0 & if \quad flag_{top}^{MB} = 1, \\ SAR & if \quad flag_{top}^{MB} = 0 \; and \; flag_{top}^{SA} = 0, \\ flag_{top}^{SA} \times MB_{width} + 1 & otherwise \end{cases} \end{aligned} \tag{6.2}$$

$$limit_x^{SA} = \begin{cases} ref_x^{MV} + MB_{width} & if \quad right^{MB} = 1, \\ ref_x^{MV} + MB_{width} + SAR & if \quad flag_{right}^{MB} = 0 \; and \; flag_{right}^{SA} = 0, \\ ref_x^{MV} + (1 + flag_{right}^{SA}) \times MB_{width} & otherwise \end{cases}$$

$$limit_y^{SA} = \begin{cases} ref_y^{MV} + MB_{width} & if \quad flag_{bottom}^{MB} = 1, \\ ref_y^{MV} + MB_{width} + SAR & if \quad flag_{bottom}^{MB} = 0 \; and \; flag_{bottom}^{SA} = 0, \\ ref_y^{MV} + (1 + flag_{bottom}^{SA}) \times MB_{width} & otherwise \end{cases} \tag{6.3}$$

The mechanism of SA-flags values computation for a clipped search area is presented in Fig. 6.14. In this figure, two out of four borders of the search area span over frame borders, and hence have to be clipped. The right and upper SA borders span over two, and one MB, respectively, before being clipped, thus, corresponding *right* ($flag_{right}^{SA}$) and *top* ($flag_{top}^{SA}$) flags of the SA flags set values would be 1 and 2. The bottom and left borders span over the frame without crossing the frame borders, therefore the corresponding SA flags, the $flag_{bottom}^{SA}$ and $flag_{left}^{SA}$ would be cleared (zeroed) as these borders are not clipped by frame boundaries. By assigning the search area start to be placed at (0,0) coordinates, the placement horizontal and vertical coordinates of the MV reference (marked with a black circle) can be reconstructed as $ref_x^{MV} = SAR$ and $ref_y^{MV} = flag_{top}^{SA} \times MB_{size}$, respectively. The span of the clipped search area can be described by the search area limit (white dot with black lining). In this example the placement horizontal and vertical coordinates of that point are be computed as $limit_x^{SA} = ref_x^{MV} + (flag_{right}^{SA} + 1) \times MB_{width}$ and $limit_y^{SA} = ref_y^{MV} + SAR + MB_{height}$. These values are used to compute the linear addresses used to extract the pels indicated by motion vectors.

This type of codification allowed to precisely determine SA grid dimensionality, MB's positioning within the SA grid and correct processing of SA with variable size, including only the required pels. If *zero padding* would be implemented, the grayed regions of the search area in Fig. 6.14 would have to be padded with zeros and sent to the kernel. The reduction in traffic is depends on the search area radius (SAR), input frame dimensionality and MB size (or width for square shaped MBs), and does not depend on the frame content. The expected reduction as function of used SR parameters for the case of QCIF reference frame is presented in Table 6.1. Search area size was computed using the equation for SA_{LR} from Table 4.1 presented on page 114. As shown, when implemented, this scheme can result in saving of up to around 13% of accesses to pels that need to be transmitted per reference

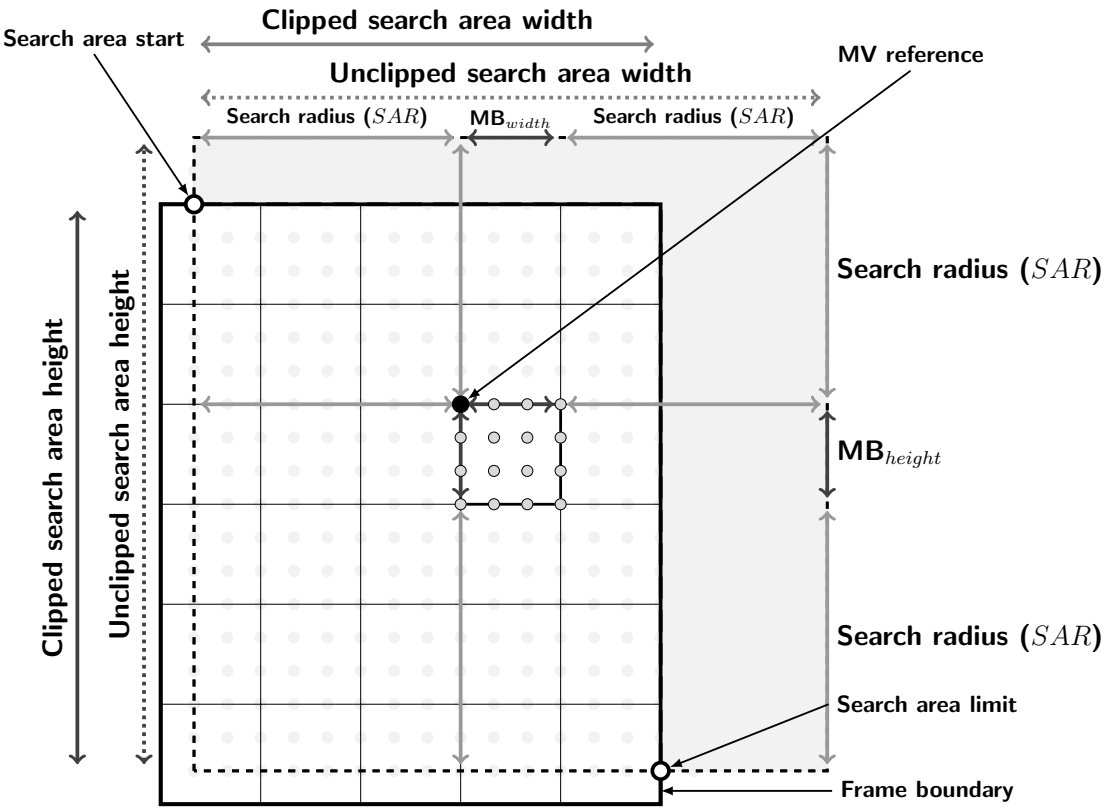


Fig. 6.14 Example of search area clipping by frame borders.

QCIF frame.

6.3.5 Grids construction and management

The reference software internally operates on grids which are used to represent images/frames at different stages of processing. These grids are instantiated as 2-dimensional arrays that, in the initial implementation, were $precision_{me}^2$ (HR grids) or $scale^2$ (SR grids) times larger than the input images (LR grids) from which they were created. This became a possible system bottleneck and required optimization.

6.3.5.1 Problem statement

A HR grid is constructed from a LR image by means of upsampling with the factor of increase in size equal to $precision_{me}$. During the upsampling process the array is zeroed and the loaded LR pels are placed at locations determined by the computed horizontal ($coordinate_{HR_x}$) and vertical ($coordinate_{HR_y}$) grid (HR) coordinates. A SR grid is con-

TABLE 6.1 *Expected savings gained by using the implemented scheme and allowing variable search area size vs padding with zeros for a QCIF (176x144 pels) reference frame. Total numbers of search area pels was computed using the equation for SA_{LR} from Table 4.1.*

MB_{width}	Search radius	Pels/Accesses		Saved accesses			MB_{nr}
		Per SA	Total	Total	[%]	Per MB	
4	2	64	101376	2544	**2.5**	1.6	1584
4	4	144	228096	7616	**3.3**	4.8	1584
4	8	400	633600	38784	**5.8**	24.5	1584
4	16	1296	2052864	224000	**10.9**	141.4	1584
8	2	144	57024	1904	**3.3**	4.8	396
8	4	256	101376	5056	**5**	12.8	396
8	8	576	228096	15104	**6.6**	38.1	396
8	16	1600	633600	74496	**11.8**	188.1	396
16	2	400	39600	1664	**3.8**	16.8	99
16	4	576	57024	3776	**6.6**	38.1	99
16	8	1024	101376	9984	**9.8**	100.8	99
16	16	2304	228096	29696	**13**	300	99

structed from HR grids by means of downsampling with factor of $\sqrt{precision_{me}^2/scale^2}$. During this process, the SR grid is zeroed and the pels from locations determined by the computed horizontal ($coordinate_{HR_x}$) and vertical ($coordinate_{HR_y}$) grid (HR) coordinates are placed at locations determined by the computed horizontal ($coordinate_{SR_x}$) and vertical ($coordinate_{SR_y}$) grid (SR) coordinates. The downsampling operation retains $scale^2$ pels from each $precision_{me}^2$ pels bundle.

The proposed system defined two types of the HR grids, namely, the MB HR and SA HR grids. As aforementioned, growth in dimensions of the HR representations depends on the internal precision of processing specified by setting the value of the $precision_{me}$ parameter. Our implementation considers only the scenario of the quarter-pixel precision ($precision_{me} = 4$). For this scenario, the presented approach results in grids being more than 16 times larger than the LR representation, significantly increased memory storage occupancy and the number of cycles required for memory zeroing. The number of elements in the MB HR grid — 2^8, 2^{10}, 2^{12} for 4x4, 8x8, 16x16 macroblock sizes, respectively — is significantly lower than the targeted cycle budget for all possible combination of algorithms values planned to be supported. Thus, storage of HR grids representing MB was considered an unlikely system's bottleneck. And in case it would become one, the actual number of memory access could be reduced using coalesced memory accesses/data words. The cardinality of the SA LR and HR set is given in (6.4) and (6.5), where *SAR* represents the search

area radius. Assuming that the execution time corresponds to the maximal number of accesses that can be carried out during this period, and using it as the maximal cardinality of the SA set in (6.5) the maximal value of SAR can be estimated. Using the targeted budget of 2867 cycles (for MB4x4) and quarter-pixel precision, we get $2867 > 16 \times (4 + 2 \times SAR)^2$, which leads to $SAR \leq 4$. This value represents an overestimate as it does not take into account the cycles required for synchronization and internal processing. Even so, this architecture gives no hope **for/of** meeting the targeted performance.

$$card(SA_{LR}) = (MB_{width_{LR}} + 2 \times SAR) \times (MB_{height_{LR}} + 2 \times SAR) \quad (6.4)$$

$$card(SA_{HR}) = card(SA)_{LR} \times precision_{me}^2 \quad (6.5)$$

A thing to notice is that the SA HR grids are used to carry out the *zeroes2ones*, the *up-holes* and the part of *shift and add* responsible of extracting the to-be-fused pels from search area. The fusion process accesses only the subset of the search area HR grid which cardinality is limited the cardinality of the MB HR grid, and thus, its memory occupancy and execution time does not increase with SA growth. The remaining two transformations that access the SA HR grids, *zeroes to ones* and *up holes*, were encapsulated by the *UpHoles* module.

6.3.5.2 Implemented solution

Theoretical analysis showed that the straight forward management of the grid construction implemented in the reference software leads to a system bottleneck.

In the proposed implementation, the problem with SA HR grids was solved by an intermediate memory addressing scheme that presented with (HR) grid coordinates was able to provide correct data operating on the LR representation. This scheme, already presented in Section 4.6.1, is based on the fact that the SA grids structure is regular with relevant data being only the LR data stored at addresses with coordinates for which the relation (6.6) and (6.7), where % represents the arithmetic modulo operator, are true.

$$coordinate_{HR_X} \,\%\, precision_{me} = 0 \quad (6.6)$$

$$coordinate_{HR_Y} \,\%\, precision_{me} = 0 \quad (6.7)$$

If the HR coordinate points at a full-pixel location, its LR coordinate counterpart address is computed as in (6.8) and the LR value identified by the LR coordinate is loaded. If the HR coordinate addresses a sub-pixel location no LR coordinate address is generated, instead,

the macroblock pel value located at the HR coordinate is set to zero.

$$coordinate_{LR}\left(\frac{coordinate_{HR_X}}{precision_{me}}, \frac{coordinate_{HR_Y}}{precision_{me}}\right) \tag{6.8}$$

Use of the above presented approach allows reducing the size of search area grids to the size of the input (LR) content. By substituting SA_{LR} from (6.4) with the targeted budget of 2867 cycles (for MB4x4) we get $2867 > (4+2\times SAR)^2$, which leads to theoretical limitation on search radius of $SAR \leq 24$. Even though in reality this margin/number was expected to be smaller, due to some cycles being spent on other tasks, it allows supporting the targeted range of SAR values without the need of using data word coalescing. The reduction in the observed execution time comes at the expense of, in the worst case scenario, one modulo, one comparison, and two division operations. The implementation of division and modulo is much simplified when $precision_{me}$ is assigned a value being a power of two. In that case, division is performed by right shifting the dividend, and modulo operation is carried out by extraction of a number of least significant bits of the operand. Implementation of division for other $precision_{me}$ values can be troublesome and introduce unacceptable latency.

The implementation of the above-mentioned modifications results in reduced memory occupancy at the cost of increased logical occupancy (LUTs). We consider this 'trade' to be a favorable one for two main reasons. First, the memory requirements for the reference implementation scale exponentially with the increase of search area size while the traded logical occupancy can be considered insignificant. Second, the memory is more likely to become the main system's bottleneck.

Further improvement could be obtained through modification of the ME process so that it would (i) provide the samples of the best candidate macroblock along the MV and SADs, and (ii) use coalesced data word in transmissions. This effectively would remove the necessity of the *UpHoles* block implementation, simplify the *ShadStep* module execution flow and allow to obtain additional speed up by preforming coalesced data accesses.

The modified addressing was implemented for the MB HR grid creation and accesses carried out to it from the *ReapSteps* modules. All other MB HR grids in the data path required access to all HR coordinates of the MB HR grid.

Further execution speed up was achieved by using coalesced memory access to the SR HR grids. By accessing x chunks of data at once we could speed up the storing/loading by x. Execution could be further sped up by using a custom vendor specific RTL RAMs with optimized reset, as this would help to significantly reduce the number of cycles required for zeroing of the memories. Most importantly, both of these techniques do not results in

reduction in memory occupancy. Moreover, implementation of the vendor-specific RAM IPs would have to be enforced by manual modifications of the resulting RTL description using vendor specific libraries. This would significantly limit the portability of the design. Thus, a different solution was sought for.

6.3.6 Frame window size determination

The concept of sliding frame window was used in order to provide support for dynamic super-resolution. The exact number of frames that are included in the SFW varies throughout the execution. This behavior requires some mechanism to be handled properly.

6.3.6.1 Problem statement

As stated in Section 2.2.4.1, the SFW can be seen as composed of two sets of frames: the frame being processed and a neighborhood comprising its preceding and succeeding frames. For the first frame of the sequence there are no previous reference frames available, thus, the SFW only contains the frames succeeding the one being processed. Having processed each frame the SFW has to be updated to include the processed frame in the preceding frames set. Analogously, when reaching the end of the sequence the number of SFW frames decreases, as there are no further frames (past the sequence limits) to be added to succeeding frames subset of the SFW.

6.3.6.2 Implemented solution

In our implementation, a variable number of frames in the SFW is managed by means of internal counters and external flags signaling state transitions. At any given moment, after being initialized, the system can be in one of three states, corresponding to the scenarios in which: (i) the preceding frames number has to be incremented, (ii) the SFW is fully occupied, and (iii) the succeeding frames number has to be decremented. The distinction between the preceding and succeeding frames set is not reflected in the implementation, as processing of these sets is the same, and the system needs to know only the number of reference structures to access.

For the first frame there are no previous reference frames and more than the maximal allowable succeeding frames available. Thus, initialization is carried out by setting the number of reference frames to the number of maximal value of the following reference frames (set at synthesis time). This value is incremented with each following SFW update, until the maximal allowed number is reached. This corresponds to the SFW being fully

occupied. Once the last frame of the sequence is contained within the limits of the SFW, in order not to reference frames beyond the limits of the sequence, the following update has to result in decrementing the number of reference frames. For the same reason, the number of reference frames is decremented with each following SFW update, reaching the maximal number of preceding reference frames at the time of processing of the last frame of the sequence. The frames number update conditions are summarized in (6.9), where, n is the number of frames in the sequence, n_{RF}^{i} represents the count of the reference frames in the SFW for ith frame of the sequence (from 1 to n), max_{PFS} and max_{SFS}, refer to the maximal cardinality of the preceding and succeeding frames set.

$$n_{RF}^{i} = \begin{cases} max_{SFS} & if \quad i = 1, \\ n_{RF}^{i-1} + 1 & if \quad 1 < i \leq max_{SFS} + 1 \\ n_{RF}^{i-1} & if \quad max_{PFS} + 1 < i \leq n - max_{SFS} \\ n_{RF}^{i-1} - 1 & if \quad n - max_{SFS} < i \leq n \\ max_{PFS} & if \quad i = n. \end{cases} \tag{6.9}$$

In the presented implementation, the presence of the first frame of a sequence and the last frame within the SFW limits is signaled by setting or clearing a corresponding flag. These flags are referenced as the *new sequence* (NS) and *window limit reached* (WLR) flag, respectively. These signals generation logic does not form a part of the SR Kernel, and their values are considered one of the input parameters of the core. In our case these signals are set by the testbench module.

After being received from the input, these flags travel along the data path with location flags. These flags and an internally stored frame counter are used in the process of determination of the SFW cardinality carried out by the *ShadPrep* module. The frame count, WLR and NS flags are checked for the first MB of each frame. If the WLR signal is set, the end of the sequence is within the forward frame window limits and the number of frames comprising SFW is to be decreased. Otherwise, the frame count is checked. If the locally stored frame number is lower than the number of frames to be included in the preceding frames set, the number of frames is increased. The count of frames is incremented with first MB of each frame and zeroed when the new NS flag is set. The number of reference frames currently in the SFW is propagated to *ReapSteps* with first MB of each frame. This value is checked in order to determine the *ShadStep*s with which the *ReapSteps* should synchronize and from which it will request data. The implemented process of SFW cardinality determination based on flags values is the one from (6.10) and (6.11), where i represents the locally

TABLE 6.2 *An example of the SFW cardinality computations, WLR flag and $u(i)$ values for a sequence comprising 300 frames ($n = 300$) and the maximal number of frames allowed in the proceeding and succeeding frames sets equal to 2 ($max_{SFS} = 2$ and $max_{PFS} = 2$).*

Frame		Flag		Update
i^{th}	n^i_{RF}	WLR	NS	$u(i)$
1	2	0	1	N/A
2	3	0	0	+1
3	4	0	0	+1
$4 \cdots 297$	4	0	0	0
298	4	0	0	0
299	3	1	0	-1
300	2	1	0	-1

stored frame number and $u(i)$ represents the current iteration update. An example of the SFW cardinality computations, NS/WLR flags and $u(i)$ values for a sequence comprising 300 frames ($n = 300$) and the maximal allowed number of the preceding and succeeding frames set equal to 2 ($max_{SFS} = 2$ and $max_{PFS} = 2$) are presented in Table 6.2.

$$n^i_{RF} = \begin{cases} max_{SFS} & if \quad i = 1, \\ n^{i-1}_{RF} + u(i) & if \quad 1 < i \leq n \end{cases} \tag{6.10}$$

$$u(i) = \begin{cases} -1 & if \quad NS_i = 0 \; and \; WLR_i = 1, \\ 0 & if \quad n^{i-1}_{RF} < max_{SFS} + max_{PFS} \\ & \quad and \; WLR_i = 0, \\ 1 & otherwise. \end{cases} \tag{6.11}$$

6.3.7 Division operation

Division operation is not a construct synthesizable by the used HL-synthesis tools. Thus, this operation had to be either emulated by other operations (i.e. bit shifts (+ addition) or multiplication) or implemented using a custom hardware module.

6.3.7.1 Problem statement

The typical software implementations use an algorithm based on counting the number of subtractions carried out until the reminder becomes negative (e.g. [HN10]). Algorithm 1

Algorithm 1 Division by successive substractions algorithm.

```
1:  procedure DIVISION(dividend, divisor)          ▷ Division of dividend by divisor
2:      reminder ← dividend
3:      quotient ← 0
4:      if quotient ≥ divisor then
5:          while reminder >= 0 do                 ▷ While the reminder is positive
6:              reminder ← reminder − divisor
7:              quotient ← quotient + 1
8:          end while
9:      else
10:         ...                                    ▷ Handle division by zero exception.
11:     end if
12:     return quotient                            ▷ Return the result
13: end procedure
```

presents pseudocode of such an approach. This algorithm is known in literature as the generalized division. The main drawback of this algorithm is its variable execution time. Additionally, the execution time is dependent on the run-time values being used in the process. When a $dividend >> divisor$ the number of iterations ($\sim$cycles) required for execution is increased significantly. These properties prohibit efficient hardware implementation.

In order to facilitate meeting the targeted cycle constraints, division was decided to be implemented as a separate module. This solution was expected to allow data load and store operations to be executed in parallel with the division, and facilitate performance customization based on division module replication or pipelining. In order to alleviate the aforementioned issues we considered two options: (i) use of a vendor specific divider IP or (ii) provision of a custom implementation. In most cases the former is the better choice, as it offers the best performance and its introduction requires the least effort. Nevertheless, in our implementation we have decided to tackle the problem of division operation implementation by provision of a custom divider module. There were two main reasons for that choice, namely customization possibilities and portability provision. The range of operations supported by the vendor-specific IP were considered to reach beyond our needs, leading to waste of resources allocated for execution of operations never used in our implementation. Moreover, use of vendor-specific solution leads to limited portability of the design and increases the effort required for migration to other FPGA devices. Apart from facilitating the possible migration, a custom implementation allows to optimize for the particular use scenarios encountered in the system.

6.3.7.2 Implemented solution

During the development the following issues had to be tackled: (1) division algorithm suitable for hardware implementation in FPGAs had to be chosen, (2) the chosen algorithm had to be customized and prepared for implementation in SystemC, and (3) implementation had to assure meeting the targeted cycle/latency budget.

6.3.7.2.1 Suitable algorithm determination. In the process of implementation several division algorithms were taken into consideration, among others, the *general*, *non-restoring*, *converge*, *pre-inverted divisor* and *reciprocal division* algorithms. All of these algorithms are presented and evaluated, also in terms of possible hardware implementation, in [DBS06]. Generalized division algorithm was not implemented due to variable number of cycles that it required, that became prohibitively high for dividends much greater than divisors. Convergence algorithms were not used due to the large precision of the intermediate representations, required by these algorithms to converge, that required a number of bits surpassing the one supported by the hardwired multiplication blocks. The pre-inverted divisor methods were discarded due to the wide range of possible divisor values that would require large look up tables.

6.3.7.2.2 Base 2 non-restoring division of integers. The implementation presented in this work uses the base 2 non-restoring division algorithm. The used algorithm is a variation of the one presented in [DBS06]. Having X represent the dividend, D the divisor, Q the quotient and R the remainder such that

$$X = Q \cdot D + R, |R| < |D|, \quad X, Q, D, R \in N \tag{6.12}$$

the base 2 non-restoring division algorithm can be executed as follows:

(1) Scale the divisor, so that it is greater than the dividend.

(2) Check the result of $X \cdot D$ and determine first reminder (r_0) and quotient digit value (q_0).

(3) Compute the subsequent p reminders and quotient digits value, where p specifies the required precision.

(4) Correct the final reminder and quotient value if needed. Invert the scaling of the quotient.

Scaling of the divisor allows simplifying hardware implementation of the division. Correct scaling should result in the divisor having additional 'leading' digit over the dividend. The only side effect of (over)shifting too much to the right could be the generation of 'redundant' sign extension digit. Let m define the scaling implemented as shift by m positions to the left. The result of $X \cdot 2^m \cdot D$ can be determined by looking at sign bits of the arguments. If $X \cdot 2^m \cdot D < 0$, then the first reminder (r_0) is incorrect, thus, the corresponding digit of the quotient (bit in case of base 2 system) (q_0) is cleared. Otherwise, the reminder is correct and the corresponding digit is set as in (6.13).

$$r_0 = \begin{cases} X + 2^m \cdot D & if \quad X \cdot D < 0, \\ X - 2^m \cdot D & if \quad X \cdot D \geq 0 \end{cases} \qquad q_0 = \begin{cases} 0 & if \quad r_0 \cdot D < 0, \\ 1 & if \quad r_0 \cdot D > 0. \end{cases} \tag{6.13}$$

The following reminder is computed based on the correctness of the current reminder. Updating the remainder can result in: (i) correct remainder r_i, when $r_i \cdot D > 0$ — corresponding to quotient digit $q_i = 1$, or (ii) incorrect remainder r_i, when $r_i \cdot D < 0$ — corresponding to quotient digit $q_i = 0$. In case of the reminder r_i being incorrect, then:

(i) the correct subsequent remainder is the $r_i = 2 \cdot r_{i-1}$,

(ii) the base for next quotient digit determination is $r_{i+1} = 2 \cdot (2 \cdot r_{i-1}) - D$, and

(iii) the value of base for next quotient digit determination could be obtained as a result of delayed correction by adding D as in $r_{i+1} = 2 \cdot (2 \cdot r_{i-1} - D) + D$.

Defining the current reminder as $r_i = 2 \cdot r_{i-1} - D$, the subsequent $p < m$ reminders (r_{i+1}) are derived from the preceding one (r_{i-1}) and the current quotient digit q_i, as follows:

$$q_i = \begin{cases} 0 & if \quad r_i < 0, \\ 1 & if \quad r_i \geq 0 \end{cases} \qquad r_{i+1} = \begin{cases} 2 \cdot (2 \cdot r_{i-1} - D) + D = 2 \cdot r_i + D & if \quad q_i = 0, \\ 2 \cdot (2 \cdot r_{i-1} - D) - D = 2 \cdot r_i - D & if \quad q_i = 1. \end{cases} \tag{6.14}$$

The remainder obtained in the last step ($i = p$) could also be incorrect. In the case of $r_p \cdot D < 0$, remainder restoration, carried out by addition of $D \cdot 2^m$, is required. Finally, the scaling applied to the divisor has to be inverted in order to obtain the final quotient value. The quotient scaling is carried out as in (6.15).

$$Q = q_0 \cdot 2^m + q_1 \cdot 2^{m-1} + q_2 \cdot 2^{m-2} + \cdots + q_{p-1} \cdot 2^{m-p+1} + q_p \cdot 2^{m-p}. \tag{6.15}$$

The above-described algorithm forms the base and reference used for SystemC code generation and validation. The control flow of the resulting implementation of the base 2

Algorithm 2 Base 2 non-restoring division algorithm with delayed correction.

procedure DIVISON($dividend, divisor, m, p$)
    ▷ Division of dividend by divisor with precision p
    $divisor_{2^m} \leftarrow divisor << m$ ▷ Prescale the divisor by the factor $m > p$
    $\{quotient_0, quotient_1, \ldots, quotient_{p-1}, quotient_p\} \leftarrow 0$
    if $dividend \cdot divisor \geq 0$ **then**
        $reminder_0 \leftarrow dividend - divisor$
    else
        $reminder_0 \leftarrow dividend + divisor$
    end if
    $i \leftarrow 0$
    while $i \leq p$ **do** ▷ While the target precision p is not reached
        if $reminder_i \cdot divisor \geq 0$ **then**
            $quotient_i \leftarrow 1$
            $reminder_{i+1} \leftarrow 2 \cdot reminder_i - divisor_{2^m}$
        else ▷ In case of $reminder_i \cdot divisor < 0$
            $quotient_i \leftarrow 0$
            $reminder_{i+1} \leftarrow 2 \cdot reminder_i + divisor_{2^m}$
        end if
        $i \leftarrow i + 1$
    end while
    quotient = $\{quotient_0, quotient_1, \ldots, quotient_{p-1}, quotient_p\}$
    if $reminder_p \cdot divisor < 0$ **then**
        $reminder_p \leftarrow reminder_p + divisor_{2^m}$
    end if
    return $(quotient, reminder_p >> m)$ ▷ Return the result
end procedure

non-restoring division is presented in Algorithm 2.

6.3.7.2.3 Division implementation. The execution time of the non-restoring division depends on the required precision p as this parameter determines the number of main loop iterations. In our implementation required precision is determined by the number of bits of the greater of the dividend and divisor. This relation between the execution time t_{div} of the division and the bit width of the operands can be approximated as in (6.16).

$$t_{div} \sim max(dividend^{bitwidth}, divisor^{bitwidth}) \approx log_2(max(divident, divisor)) \tag{6.16}$$

In our implementation, division is required during the fusion and interpolation stages. For the former, the required number of bits satisfactory for divisor representation depends on

the maximal number of frames in the reference frames in the SFW (n_{RF}^{max}) and maximal (allowed) value of the fusion weight ($weight_{fusion}^{max}$). In case of the latter stage, the maximal divisor bit width is derived from the cardinality of the pel neighborhood specified by the interpolation window and the maximal value of the contribution weight ($weight_{int}^{max}$). For both stages, the dividend adds additional dependency on the maximal pel value (pel_{val}^{max}) with respect to the divisor dependencies. Thus, in both cases the dividend bit width dominates the overall requirements. These relations are summarized in (6.17) and (6.18), and, (6.19) and (6.20), respectively, for the fusion and interpolation stages.

$$divisor_{fusion}^{bitwidth} \sim log_2\left(\lceil n_{RF}^{max}\rceil \times \lceil weight_{fusion}^{max}\rceil\right) \tag{6.17}$$

$$dividend_{fusion}^{bitwidth} \sim log_2\left(\lceil n_{RF}^{max}\rceil \times \lceil weight_{fusion}^{max}\rceil \times \lceil pel_{val}^{max}\rceil\right) \tag{6.18}$$

$$divisor_{interpolation}^{bitwidth} \sim log_2\left((int_{window}^2+1)^2 \times \lceil weight_{int}^{max}\rceil\right) \tag{6.19}$$

$$dividend_{interpolation}^{bitwidth} \sim log_2\left((int_{window}^2+1)^2 \times \lceil weight_{int}^{max}\rceil \times \lceil pel_{val}^{max}\rceil\right) \tag{6.20}$$

Substituting the parameters with their respective values we obtain the same requirement of 13 bits. Thus, implementation arguments bitwidth can be limited to 13 bits.

$$dividend_{fusion}^{bitwidth} \approx log_2(16\times 1\times 256) < log_2(2^{5+8}) < 13 \tag{6.21}$$

$$dividend_{interpolation}^{bitwidth} \approx log_2(25\times 1\times 256) < log_2(2^{5+8}) < 13 \tag{6.22}$$

The use of 13 bits to represent the value of dividend and 12 to represent the divisor resulted in the necessity of 12 rounds and a latency of 13 cycles for internal computations [DBS06]. Assuming that capturing the input and latching the output require 1 cycle each, the total number of cycles can be estimated as $N_{rounds}+3$. For the mentioned configuration this gives 18 cycles meaning that up to 160 divisions could be carried out during the targeted 2867 cycles limit. The *NormalizeMB* had to be capable of performing up to 240 (holes) divisions and 256 load/store operations in the time reserved for one 4x4 MB processing. Thus, the limit in the number of cycles for a division was estimated at around 9 cycles. This was more than the maximal number of cycles required for division to be carried out.

In order to meet the targeted performance, the division module could have been replicated with the workload being distributed to multiple instances. Nevertheless, this solution would further complicate the execution flow and require additional logic for available division input/output ports selection. Thus, in the final implementation the divider module was not replicated. Instead, the division was decomposed in to two stages that were later pipelined. This allowed the module to start and terminate a new division every 9 cycle (with

full pipeline).

Finally, a pipelined version of the divider module was implemented. In this version the division loop was split into two stages. The internal loops of the two stages are basically equivalent, with the exception of the starting values and ending condition. The second stage includes additional code for final reminder and quotient computation resulting in additional cycle of latency. This latency is matched by an additional 1 cycle stall in the first stage of the division resulting in $N_{rounds}+4$ being required for the division result to be readily available at the output. Nevertheless, by using a pipelined architecture the effective throughput (with loaded pipeline) is almost doubled, as the input/output data can be set every 9 cycles. This performance has been found sufficient for our implementation needs.

Once the division implementation met the performance target, the existing execution flow of the modules using division had to be modified in order to allow efficient outsourcing of computations and concurrent execution. These modules shared a bidirectional relationship with the newly created division modules. As aforementioned, this type of relationship requires tight synchronization scheme, in order to maximize periods of non-blocking execution. The processing carried out by the divider module has been pipelined allowing one division to be issued every 9 cycles. To maximize performance the pipeline should be kept full and two division operations should be issued before the first outcome is received. If the processing carried out in the internal loop of the outsourcing module is organized in a way that allows synchronizing with the divider every 9 cycles (the number of cycles required for a pipeline stage execution) then the maximal pipeline utilization can be achieved reducing the division operation processing time by a factor of (nearly) 2, while avoiding the consequences of replication of the division module.

6.4 Results

The results of the hardware implementation of the MB-level NUGPA core in and advanced FPGA technology are presented in Table 6.3. Implementation in the *xc5vf70t-1* [Xil09] device (labeled as $Tech_1$ in TABLE 6.3) for MB size of 4x4 pels, scale equal to 2, and SFW containing 3 frames (RF=2) occupied 10291 LUTs, 16 DSP blocks, and 66 Block RAMs. This corresponds to device resource occupancy of, respectively, 22%, 12% and 44.6%. Total memory usage was 2.376 MB (44%). No off-chip memory was required. The final clock frequency is of 109 MHz, resulting in the capability to apply 2 x super-resolutions on 25 QCIF frames per second.

TABLE 6.3 *Resulting hardware performance for two FPGA devices in comparison with [BB08] for the particular case of RF=2, scale=2, SA=2, and MB=4. N/A stands for not available.*

	SR_{iuma}		Reference [BB08]	
	Technology		Iteration stages [Xil07]	
	$Tech_1$ [Xil09]	$Tech_2$ [Xil07]	10	20
Occupancy [LUT]	10291	13031	35707	68317
BRAM	66	132	134	234
DSP blocks	16	2	N/A	N/A
Frequency [MHz]	109	68	58	58
Frame rate [fps]	25	16	61	61

6.4.1 Core-level implementation evaluation

Apart from presenting the observed implementation results, Table 6.3 contains data that allow a direct comparison with the state-of-the-art implementation presented in [BB08]. In terms of raw performance the MBL level implementation, offers lower frame rate and supports lower range of spatial resolutions. Nevertheless, the competitive implementation only features 10 iterative stages, that is, roughly a half of the number of 20 stages assigned by the authors as the threshold for producing satisfying outcome quality. Thus, in order to offer a better comparison of implementation efficiency, an approximation of the resources required to implement a solution featuring 20 iteration stages is also presented. The logic and memory requirements for this configuration were estimated by addition of the cost of implementation of additional 10 iteration stages to the cost of implementation of the system already sporting 10 iteration stages. On the other hand, the implementation presented in this work was also mapped and routed for the device [Xil07] targeted in [BB08]. The results for this implementation are labeled as $Tech_2$ in Table 6.3, and can be used for direct comparison.

The implementation presented in this work was optimized in order to offer base software quality while minimizing resources utilization. In order to measure the efficiency of the FPGA implementations, the LUTs per SR boost cost ($boostCost_{SR}$) figure of merit is proposed. This figure represents the number of look-up tables the implementation required in order to offer an increase of 1% in measured average (objective) quality of the super-resolved image over the image produced using bilinear interpolation. It is desired to keep this figure as low as possible. The $boostCost_{SR}$ is estimated as in (6.23), where the LUT_{nr} stands for the number of LUTs required for implementation and the $PSNR_{SR_i}$ and the $PSNR_{int_i}$ values represent, respectively, the PSNR value of the super-resolved, and

TABLE 6.4 *PSNR observed at the output (average over 90 initial frames; CIF; luma component).*

Implementation [Xil07]	Average PSNR [dB]		
SR_{iuma}[1]	**22.38**	**29.42**[Xil09]	**22.79**
IBP weighted (20 iterations) [BB08]	20.46	27.80	22.22
Nearest-neighbor interpolation	18.78	26.68	19.89
Bilinear interpolation	20.46	28.07	21.23
Bi-cubic interpolation	20.29	28.10	20.92
	mobile	foreman	paris

[1] MB_{width} = 4, RF=4 (2+2), SAR = 16, and $scale$=2

TABLE 6.5 *Objective quality boost ($boostCost_{SR}$; defined in* (6.23)*; lower is better) over interpolation cost in LUTs for the particular case of RF=2, scale=2, SA=2, and MB=4.)*

Implementation	$boostCost_{SR}\left[\frac{LUT}{\%}\right]$		
versus nearest-neighbor interpolation			
$SR_{iuma}Tech_2$	**1269**	**680**	**894**
weighted IBP [BB08]	16275	6546	5832
versus bi-cubic interpolation			
$SR_{iuma}Tech_2$	**2753**	**1266**	**1458**
weighted IBP [BB08]	No gain	30804	10994
Sequence	paris	mobile	foreman

interpolated i^{th} frame of the test sequence comprising a set of n frames.

$$boostCost_{SR} = \left(\frac{n \times LUT_{nr}}{100}\right) \Bigg/ \left(\sum_{i=1}^{n} \frac{PSNR_{SR_i} - PSNR_{int_i}}{PSNR_{int_i}}\right) \tag{6.23}$$

The $boostCost_{SR}$ for the proposed implementation was computed based on the implementation results presented in this section for $Tech_2$ and the average PSNR value noticed for 90 initial frames of the paris, foreman and mobile test sequences (luma component) presented in Table 6.4. The values for IBP were kindly provided by the authors of [BB08]. Other values were obtained with the set-up used for reference software evaluation (Section 3.4.1). The resulting values of the $boostCost_{SR}$ for the foreman, mobile and paris sequences are presented in Table 6.5.

The results of the study show that for the tested sequences the $boostCost_{SR}$ noted for the proposed implementation is at least 6.5 times lower than the one for the [BB08] implementation. The main reasons for such a big difference are: (i) the use of the block matching

TABLE 6.6 *Observed resource occupancy for some SRK parameters values combinations with* $scale = 2$, $MB_{width} = 4$, $RF = 2$, *and* $SAR = 2$. *(Synplify 2009.06 Premier, values synthesis in isolation, only FF-FF latency considered.)*

Module	Occupancy			Latency
	Instance	Share in total		
	[LUTs]	[LUTs]	[%]	[ns]
UpGrid	122	122	1.1	5.06
UpHoles	47	47 × 2	0.9	2.25
ShadPrep	393	393	3.4	4.92
ShadStep	250	250 × 2	4.3	4.50
ReapSteps	359	359	3.1	4.82
NormalizeMB	158	158	1.4	4.28
HolesFillingPrep	**3492**	3492	29.9	**7.55**
MeanNearest	2173	**2173 × 2**	37.2	5.99
Divisor	174	174 ×3	4.5	4.46
ReorderBufferReap	191	191	1.7	3.93
ReorderBufferSew	529	529	4.6	**7.85**
MBcollector	1007	1007	8.6	**9.22**
Sum of all submodules		11713	100	-
Kernel	10291	-	-	9.16

(BM) ME, and (ii) the non-iterative nature of the implemented algorithm. The former leads to higher PSNR, while the latter helps to reduce the overall occupancy. The impact of the ME algorithm manifests itself especially in results for sequences with abundance of local movement i.e. foreman and mobile. Those movements cannot be traced accurately using frame-level ME. For those sequences the PSNR obtained using frame-level estimation is significantly lower, leading to higher $boostCost_{SR}$.

6.4.2 Implementation results for each module

The resulting resources occupancy and critical path latency observed for each of the core's submodules are presented in Table 6.6. Resource figures are taken from the utilization report issued at the end of implementation in Synplify 2009.06 Premier. Occupancy and latency is measured using the so called 'Out-of-Context' flow to synthesize and implement the submodule instance in isolation. This ensures that the design is not distorted in order to route to device pins. LUT figures do not include LUTs used as pack-thrus, but do include LUTs used as memory. Default Synplify 2009.06 Premier settings were used. Other configurations may yield different results. Because surrounding circuitry will affect placement and

timing, no guarantee can be given that these figures will be repeatable in a larger design.

Most of the logical resources have been allocated for the implementation of the interpolation process. This task accounted for around 47% of total occupancy (with dedicated divisors). This was expected, as this task is by far the most computationally complex part of the processing. The very high occupancy observed for the *HolesFillingPrep* was not expected as this module tasks are mostly of management nature. The occupancy requirements of this module are made high by: (i) the logic required to manage the inter-MB data dependencies, and (ii) complex address generation due to memory replication, data packing and workload distribution. Both of the above increase the complexity of the module control flow and the number of variables used for its description using logical equations. Moreover, the latter, paired with delays caused by high fan-out due to data reuse (replication), is the main reason for the elevated critical path latency. These two modules are firm candidates for further decomposition and refinement.

In terms of the critical path latency, the core's performance is limited by the efficiency of implementation of the adapters. In particular, the *MBcollector* with latency of 9.22ns is the system bottleneck, followed by *ReorderBufferSew* with latency of 7.85ns. Both of these modules access large memories and require somewhat complex address generation. In the particular case of the *MBcollector* address generation is not only computationally expensive but also implements complex control flow in order to handle different MB placements within the frame. Moreover, adapters do not form an intrinsic part of the SR system and their (at least partial) elimination, either by introduction of modifications to the ME process or outsourcing of processing to other parts of the implementation platform, is planned. Thus, the effort put into their optimization has been significantly lower than for the SRK module. To recap, address generation of the *MBcollector* and *HolesFillingPrep* modules forms the system's performance bottleneck. A very likely solution to this problem would be to create a set of look-up-tables/ROM memories with pre-computed values. This approach is widely used in recent compression systems (e.g. *Scalable Video Coding* (SVC), HEVC, etc.). In the case of this system's modules, a good starting point would be to provide tables with pre-compiled values of row addresses and strides, organized in a way that would allow simple addressing of their content e.g. addressing based on placement flags and values obtained for the preceding accesses and concatenation. It is expected that, when optimized, this approach would significantly reduce the computational complexity and the number of cycles needed for execution at the cost of increased memory occupancy.

The least complex modules are the *UpHoles* and *UpGrid* modules. Both of these modules perform almost identical tasks. The difference, apart from different data set and work-

load cardinality, is that the former module's control flow is simplified due to HR holes grids elimination allowed by new addressing scheme in *ReapSteps*. By leaving the *UpGrid* module unchanged, we were able to quantify the impact of the aforementioned algorithmic change on the implementation efficiency. By applying the new addressing scheme on the *UpGrid* output, we expect its occupancy and latency to reach the values observed for the *UpHoles*, which corresponds to reduction in occupancy to ∼30% and latency to ∼50%.

Based on the data presented in the Table 6.6 we estimate the cost of supporting an additional reference frame to be in the range of ∼410 and 450 LUTs. This value includes the cost of:

- Additional instance of *UpHoles* and *ShadStep* modules (∼320 LUTs).
- Additional logic required to manage additional communication channels and workload in *ReapSteps* (∼30 LUTs).
- Additional logic (little) required to manage additional communication channels in *ShadPrep* (∼10 LUTs).
- Higher bitwidth of normalization arguments (∼15 LUTs).
- LUTs used for instantiation of memories in the data path of the newly instantiated modules (∼40 LUTs).

The increase in the latency of the *ShadPrep* caused by additional RFs introduction is expected to be insignificant. The occupancy and latency of the *ReapSteps* module increase to ∼636 LUTs and ∼6 ns and does not become a bottleneck. Extension of the division argument's bitwidth to 18bits results in occupancy and latency of the *Divisor* module of ∼223 LUTs and ∼7.61ns. The latter surpasses the latency observed for *HolesFillingPrep* becoming the bottleneck of the SR kernel implementation (still, *MBcollector* remains the system's bottleneck with 9.26ns). This problem might be mitigated by more aggressive unrolling of the division loop and introduction of additional latches separating the unrolled iterations.

In the presented implementation, we use a custom divider IP. We opted for that solution instead of using the default option for the targeted technology, the *LogiCORE IP Divider*, in order to preserve design portability to other technologies.

6.5 Conclusions

The focus of this chapter was the hardware implementation of the proposed MB-level flow of the NUGPA. This chapter started with a brief presentation of the methodology established

for this implementation. Next, the evolution of the system's architecture was presented using a series of diagrams presenting the architecture model at different levels of abstraction. Each of the ESL-level abstractions used in the implementation was briefly described in a way that highlighted its intended usage and limitations.

The final architecture, organization and the challenges tackled in order to meet the targeted performance of 24 fps with operating frequency of 109 MHz using the *xc5vf70t-1* device (Xilinx Virtex5 technology), were presented. The faced challenges description followed a common outline of stating the faced problem and describing the implemented solution. The reasoning behind the undertaken decisions and design choices was clearly stated. This structure facilitates transfer of knowledge and of the obtained 'know-how', allowing future designers to easily determine if the solution implemented in this work could be of relevance to their design. A particular case of design considerations was the implementation of a custom division accelerator, which was not motivated by the need for system's performance optimization but rather by the determination to preserve system portability.

Finally, the obtained implementation (post-HDL optimization) results have been presented, discussed and compared with the state-of-the-art solution [BB08]. The observed implementation results prove that the MB-level processing contributes towards real-time implementations of the NUGPA by significantly reducing the implementations memory occupancy. The comparison with the state-of-the-art has yielded satisfactory results as the observed occupancy for the proposed system was up to 5 times lower than the one reported for the reference state-of-the-art implementation, when mapped using the same target technology. In order to measure the efficiency of the implementation, a suitable figure of merit that considers output quality gain per resource occupancy has been formulated. Using this figure, we have proved that the presented implementation, whose optimality is not claimed in this work, does contribute to the state-of-the-art by implementing the MB-level processing flow, and advances the state-of-the-art of the SR implementations in terms of implementations efficiency.

Chapter 7

Conclusions and future work

7.1 Introduction

The main objective of this doctoral research has been to demonstrate real-time execution capabilities for the NUGP algorithm by means of delivering a hardware implementation that uses only on-device memory resources. In order to reach this objective we have introduced algorithmic-level modifications to the reference algorithm execution flow. These changes reduce its high memory occupancy requirements precluding high-performance hardware implementation. Developed theoretical models of the proposed and the reference flows have been used to identify their bottlenecks and quantitatively evaluate the gains and expenses associated with the change to the proposed finer-grain flow. For hardware implementation we have established a methodology that takes into account the ever-evolving nature of the algorithm's software implementation and uses a tiered hierarchy of abstractions from which the level of pin-accurate HDL description is obtained in an automated manner. Using the established methodology the proposed architecture has been successfully mapped onto Xilinx Virtex FPGA technology.

This chapter presents a recapitulation of the contributions provided by this doctoral thesis and restates their relevance to the super-resolution field. In order to do so, the conclusions will be drawn from the carried out research, and further research lines which might complement and enhance some of the aspects developed in this work will be presented.

7.2 Conclusions

1. The review of the SR algorithms taxonomy presented in this work has identified the non-iterative *direct* and *iterative back projection* as the algorithms of the multi-frame SR family best suited for hardware implementation. A closer look at the state-of-the-art of hardware implementations in FPGA has highlighted:

   (i) the trend of using execution flows that apply SR on fine-grain elements in order to increase exploitable parallelism and reduce memory utilization,

   (ii) the fact that the iterative approaches have to settle for sub-optimal output quality due to being limited by the amount of available resources, and

   (iii) the predominant impact of on-device memory occupancy and off-device memory access latency on the system's capabilities.

   This review has demonstrated that there are still many challenges within the field in terms of hardware implementations efficiency and quality, as well as facilitation of hardware implementation by considering the context of limited resources typical of FPGA targets.

2. This doctoral thesis has tackled key contemporary challenges faced at the time of developing hardware implementations of the SR techniques for video sequences, namely: the delivery of real-time execution capability, provision of high implementation efficiency, and preservation of software-level quality of the super-resolved image observed at the output. The ultimate result of this thesis has been to provide a hardware implementation that is characterized by the above-mentioned properties.

3. We have created a complete taxonomy of the different approaches and algorithms developed in the literature to carry out super-resolution.

4. We have introduced mathematical models for the image registration stage and for the reconstruction stage of the non-iterative restoration-interpolation process. In an attempt of complementing the existing algorithmic descriptions with a consistent closed form analytical model that captures the layered computation scheme of the SR process. The different transformations operating over different image spaces are clarified and underlined.

5. The starting point of this research work was a software implementation of the non-iterative version of the non-uniform grid projection algorithm coded in ANSI C.

The review of the state-of-the-art has contextualized the NUGP algorithm as belonging to the interpolation-based family of the direct classical multi-frame super-resolution algorithms, which is considered suitable for efficient hardware implementations. The presented quantitative results of the study on the objective quality of the super-resolved images produced by the reference software implementation prove the capability of the algorithm to provide quality better than other algorithms used by state-of-the-art hardware implementations. Considering the single-pass nature of the non-iterative version of the NUGP algorithm, we have concluded that this version of the algorithm meets the prerequisites for delivering a hardware implementation that meets real-time performance while preserving high quality of the super-resolved images.

6. The base software implementation did not consider hardware implementation challenges and was characterized by high use of memory resources precluding its direct implementation in FPGA technology using only on-device memory. With the goal of overcoming these weaknesses, we have proposed a modified execution flow that applies SR on a finer-grain element When implemented, this flow has removed the algorithm's dependency on frame-level buffers requiring only single MB execution context to be readily available from on-device memory. The theoretical models, developed based on the software implementation, of the proposed and reference flows have been used to identify these flows' bottlenecks, and quantitatively evaluate the gains and expenses associated with the change to the proposed finer-grain flow.

7. The results of the proposed modifications lead to significant memory occupancy reduction at the expense of increased memory traffic. The computed minimal and maximal value of the expected factor of reduction in memory occupancy associated with the switch to the MB-level flow was within the range of 3.5 to 16, depending on the SRK parameter values. The computed minimal and maximal value of the expected factor of increase in memory traffic associated with the change was within the range of 1.1 to 16.9. The exact value of these factors depends on the parameters value combination used to configure the SR kernel. For 87 out of the 96 investigated configurations (more than 90%) the factor of memory occupancy reduction has been greater than the factor of increase of memory traffic, proving the effectiveness of the proposed approach.

8. For the needs of the planned hardware implementation we have developed a high-level implementation methodology. The established methodology defines a hierarchy

of levels of abstractions, in which each of the models is obtained from the model being one level higher in the hierarchy. This renders the models tightly-coupled, facilitating inter-model propagation of modifications. By using SystemC to code the intermediate models we have been able to reuse most of the ANSI C code. This facilitates rapid propagation of modifications made at the functional level through the intermediate models down to the HDL description. The use of intermediate representation increases design portability by allowing high-level synthesis to target a range of HDL languages (VHDL, Verilog, etc.) from the same higher-level description. Furthermore, by using C-based languages throughout the abstraction hierarchy we leave open the possibility of rapid migration of the models to other C-based languages, most notably the OpenCL standard. The level of details of the provided methodology description facilitates the reuse of the established flow in implementations of other similar algorithms.

9. We have provided a hardware implementation that is characterized by real-time performance and efficiency, while preserving portability and software-level quality of the output super-resolved images. The final architecture has met the targeted performance of 24 fps with operating frequency of 109 MHz using the *xc5vf70t-1* device (Xilinx Virtex5 technology). The carried out comparison with the state-of-the-art has yielded satisfactory results as the observed logical resources occupancy for the proposed system was up to 5 times lower than the one reported for the state-of-the-art mapped using the same target FPGA technology [BB08]. The hardware implementation synthesis results: (i) have demonstrated the capability of reaching real-time performance in FPGA technology, while preserving the quality of the super-resolved images at the level offered by software implementation, and (ii) have proved the correctness of the presented algorithmic level changes and the established implementation methodology.

10. The portability of the design has been preserved by using several tiers of abstraction code in generic SystemC which allow the high-level description to be synthesized for a range of different languages (VHDL, Verilog, etc.) and targets (Altera, Actel, Xilinx, etc.) without (or with) vendor specific optimization enabled. The portability of the design is further reinforced by not using (explicitly) any vendor-specific accelerators (MACs, adders, dividers, etc.).

To recap, this thesis: (i) has successfully carried out knowledge transfer from the domain of compression algorithms to the domain of SR algorithms. (ii) has quantitatively evaluated

the impact of algorithm modifications, (iii) has provided a high-level methodology of implementation that facilitates rapid propagation of modifications, and (iv) has delivered an efficient hardware implementation in FPGAs that reaches real-time capabilities while preserving the quality of the output super-resolved images. The observed synthesis results justify the claim that the proposed implementation contributes to the advancement of the state-of-the-art by implementing the MB-level processing flow (typical flow of image/video compression algorithms) and by delivering higher implementation efficiency.

7.3 Future work

This work accomplishments leave room for new research lines focused on pushing further the state-of-the-art of SR implementations in hardware:

- The proposed implementation could be explored for further improvement, in particular:

  (i) Implementation of the algorithmic level improvements proposed by Quevedo in [Gut15]. These enhancements would improve the robustness of the fusion kernel by detecting and eliminating outliers, leading to a further increase in the observable output quality.

  (ii) Extension of the range of supported SRK parameters values. Additionally, explore the possibilities of supporting variable macroblock size used in recent compression algorithms.

  (iii) Mapping onto heterogeneous platforms and encapsulation as an IP. This task would require an exploration of changes that could lead to a better hardware/-software partitioning, motion estimation methods, buffering scheme, elimination of adapters, and appropriate interconnections interface selection (e.g. AHB, AXI, etc.).

- The work of Singla et al. [STdA13] presents an approach that uses multiple kernels of NUGPA applying SR in parallel on exclusive chunks of the input frame in order to speed-up the processing time. The presented approach is limited by memory accesses latency. As our approach uses only on-device memory its extension to such a configuration could allow alleviating these issues and is expected to result in significant execution speed-up. To provide performance scalability by means of replication, the architecture proposed in this work would require some modifications. In this line, it

would be interesting to explore: (i) further possibilities of reduction of memory occupancy, (ii) the possibility of elimination of the frame-level adapters from the SR kernel (in order to prevent their replication), and (iii) methods for efficient workload distribution. Apart from performances increase, the replication of kernels opens the possibility of supporting the multi-camera approach proposed in [Gut15], in which each camera's input is processed by a dedicated kernel instantiation.

- Multi-core and many-core platforms are undoubtedly the emerging trend in high performance computing (HPC). The appearance of hybrid solutions that use these platforms in tandem with FPGAs [PCC+14, WHK+14, MJK12], suggests new ways of conducting design space exploration for SR mappings onto these platforms. It would be of great interest to implement a SR algorithm onto several of these platforms, compare results of the mapping, analyze the arising implementation challenges, and the algorithmic- and architecture-level modifications their efficient mapping would require.

- In this line, we could take advantage of the ANSI C and SystemC implementations of NUGP algorithm developed for this thesis, as this code provides straight forward route for implementation using C-based languages like OpenCL or CUDA. OpenCL would be the preferred choice as it is an open and well defined standard supported on many platforms (CPU, GPUs, MP-SoCs, etc.) including FPGAs [Alt15]. OpenCL is expected to allow re-using most of the code in mappings on various platforms (NVIDIA Tegra 1000, Kalray, Intel Xeon Phi, Movidius Myriad 2, Amd APP, Altera Cyclone V, Xilinx EX-850 etc). The methodology established in this work is expected to facilitate this process as ESL flows using C-based languages (e.g. SystemC) have been reported to be the preferred starting point for OpenCL implementations in FPGAs [Eco12].

- The established methodology has been successfully used for mapping of SVC deblocking filter onto FPGA [CEN+13a, CEN+13b]. These studies extend the established methodology by describing steps that lead to a successful HDL migration to ASIC technology. It would be of interest to provide a manual (HDL) mapping and compare the results of the ESL-centered and manual HDLs mapping, both onto FPGAs and ASICs. This would allow quantitative evaluation of the difference in quality of the resulting HDL descriptions produced by these flows.

- Finally, it would be interesting to use the established methodology for implemen-

tation of other high complexity image processing algorithms and/or using different toolchains. This would (i) further reinforce the credibility of the methodology's effectiveness, robustness and utility in other applications, and (ii) allow revising the methodology's strengths and limitations. Additionally, it would be of use to compare the SystemC-based flow with flows centered on using other high-level of abstraction languages.

Appendix A

Publications

International conferences

[C1] **Tomasz Szydzik**, G.M. Callico, and Antonio Nunez, Quantitative Modelling of Image Processing Algorithms for Hardware Implementation, *in Design of Circuits and Integrated Circuits (DCIS), Conference on*, November 2015, *Accepted for presentation*.

[C2] **Tomasz Szydzik**, G.M. Callico, Antonio Nunez, F. Tobajas, R. Sarmiento, and E. Quevedo, Optimization of non-uniform grid projection image super-resolution algorithms by reduced granularity and modified addressing, *in Design of Circuits and Integrated Circuits (DCIS), Conference on*, vol., no., pp.1–6, 26–28 Nov. 2014, doi: 10.1109/DCIS.2014.7035527.

[C3] Pedro P. Carballo, Omar Espino, Romen Neris, Pedro Hernandez-Fernandez, **Tomasz M. Szydzik**, and Antonio Nunez, Scalable Video Coding Deblocking Filter FPGA and ASIC Implementation Using High-Level Synthesis Methodology, *in Digital System Design (DSD), Euromicro Conference on*, vol., no., pp.415–422, 4–6 Sept. 2013, doi: 10.1109/DSD.2013.52.

[C4] Pedro P. Carballo, Omar Espino, Romen Neris, Pedro Hernandez-Fernandez, **Tomasz M. Szydzik**, and Antonio Nunez, Implementation of scalable video coding deblocking filter from high-level SystemC description, *VLSI Circuits and Systems VI, SPIE Conference on*, vol. 8764, pp. 876408, 28 May 2013, doi: 10.1117/12.2016885.

[C5] **Tomasz Szydzik**, G.M. Callico, and Antonio Nunez, Real-time Implementation in FPGA of a Super-Resolution Algorithm using Macro-Block Execution Flow, *in De-*

sign of Circuits and Integrated Circuits (DCIS), Conference on, Poster abstract, Avignon France, Nov. 2012.

[C6] **T. Szydzik**, G.M. Callico, and A. Nunez, RealTime Video Restoration using FPGA Devices, in ACACES 2012 Poster Abstracts, Belgium: Academia press, Ghent, pp. 181, Fiuggi, Italy, 8–14 July 2012, `http://hdl.handle.net/1854/LU-3234245`.

[C7] **Tomasz Szydzik**, G.M. Callico, and Antonio Nunez, A Low Memory Requirements Execution Flow for the Non-Uniform Grid Projection Super-Resolution Algorithm, *in Multimedia (ISM), 2011 IEEE International Symposium on* , vol., no., pp.85–90, 5–7 Dec. 2011, doi: 10.1109/ISM.2011.22.

[C8] **Tomasz Szydzik**, G.M. Callico, and Antonio Nunez, Closing the gap between software and hardware super-resolution image reconstruction: provision of high-quality output, *VLSI Circuits and Systems V, SPIE Conference on*, vol. 8067, pp. 80670M, 3 May 2011, doi: 10.1117/12.888253.

[C9] M. Thadani, **T. Szydzik**, P. P. Carballo, P. Hernandez, Gustavo M. Callico, and A. Nunez, ESL quantitative assessment of the optimization steps in an ESL flow for a hardware implementation of a H.264/AVC decoder, *Digital System Design, Architectures, Methods and Tools, 12th Euromicro Conference*, 2009.

[C10] M. Thadani, P. P. Carballo, P. Hernandez, Gustavo M. Callico, **T. Szydzik***, and A. Nunez, ESL flow for a hardware H.264/AVC decoder using TLM-2.0 and high level synthesis: a quantitative study, *VLSI Circuits and Systems IV, SPIE Conference on*, vol. 7363, pp. 73630K, 28 May 2009, doi: 10.1117/12.821647.

Journals

[J1] **T. Szydzik**, G. M. Callico, and A. Nunez, Efficient FPGA implementation of a high-quality super-resolution algorithm with real-time performance, *in Consumer Electronics, IEEE Transactions on*, vol.57, no.2, pp.664–672, May 2011, JCR=1.045, Q2, doi: 10.1109/TCE.2011.5955206.

Other publications

[O1] **Tomasz Szydzik**, and D Moloney, Precision Refinement for Media-Processor SoCs: fp32 -> fp64 on Myriad, *Hot Chips: A Symposium on High Performance Chips*, IEEE Technical Committee on Microprocessors and Microcomputers, in cooperation with ACM SIGARCH, poster, Flint Center, Cupertino, CA, August 10–12, 2014, poster, acceptance rate < 4.9%.

[O2] **Tomasz Szydzik**, Marius Farcas, Valeriu Ohan, and David Moloney, Level-3 BLAS on Myriad Multi-Core Media-Processor, *Hot Chips: A Symposium on High Performance Chips*, IEEE Technical Committee on Microprocessors and Microcomputers, in cooperation with ACM SIGARCH, poster, Flint Center, Cupertino, CA, August 10–12, 2014, acceptance rate < 4.9%.

[O3] **T. Szydzik**, A. Nunez, L. De Paepe, and L. Albani, Contributions to visualization algorithm enabling GPU-accelerated image displaying for dual panel high dynamic range LCD display, *in Image and Signal Processing and Analysis (ISPA), 8th International Symposium on*, vol., no., pp.483–488, 4–6 Sept. 2013, doi: 10.1109/ISPA.2013.-6703789.

[O4] **T. Szydzik**, G.M. Callico, and A. Nunez, Real-time Implementation in FPGA of a Super-Resolution Algorithm using Macro-Block Execution Flow, *in Design of Circuits and Integrated Circuits (DCIS), Conference on*, poster, Avignon France, Nov. 2012.

[O5] **T. Szydzik**, G.M. Callico, and A. Nunez, RealTime Video Restoration using FPGA Devices, ACACES 2012, poster, Fiuggi, Italy, 8–14 July 2012.

[O6] M. Marrero-Martin, **T. Szydzik**, J. Garcia, B. Gonzalez, and A. Hernandez, Effect of separation and depth of N+ diffusions in the quality factor and tuning range of PN varactors, *VLSI Circuits and Systems V, SPIE Conference on*, vol. 8067, pp. 80670T, 3 May 2011, doi: 10.1117/12.888663.

Appendix B

Resumen en castellano

Se presenta en este capítulo una visión general del trabajo de investigación realizado en esta Tesis Doctoral, poniendo especial énfasis en las contribuciones en el campo de las implementaciones sobre FPGAs del *algoritmo de proyección de cuadrícula no-uniforme* (Non-Uniform Grid Projection Algorithm or NUGP algorithm) para super-resolución de imágenes y secuencias vídeo en tiempo real, los logros alcanzados y las líneas de investigación futuras.

B.1 Introducción

Vivimos en una realidad dominada por el contenido visual, más específicamente, por la alta resolución (HR) del contenido visual. Las imágenes de alta resolución son de crucial importancia en varias áreas centradas en torno a dos aplicaciones principales: la mejora de la información gráfica para la interpretación humana y para la visión artificial robusta y automática. Con el fin de lograr los nuevos niveles de calidad visual demandados, la calidad de los contenidos de baja resolución (LR) debe ser aumentada mediante un proceso denominado mejora de la resolución. La resolución de la imagen define los detalles contenidos en una imagen: cuanto mayor sea la resolución de una imagen, mayor será la cantidad de detalles que contiene. La resolución de una imagen digital se puede referir a: resolución de los píxeles, resolución espacial, resolución espectral, resolución temporal, y/o resolución radiométrica. En este contexto, esta tesis doctoral se centra en contribuir en la mejora de la resolución espacial.

La mejora de la resolución espacial se enfrenta a una serie de retos, de entre los cuales caben resaltar los siguientes:

(A) Las soluciones de base tecnológica no se consideran eficientes en cuanto a sus costes.

Con el aumento de una potencia de cálculo de fácil acceso, la mejora de la resolución espacial utilizando un nivel de post-procesamiento algorítmico se convierte en una alternativa prometedora, sobre todo si los recursos computacionales pueden ser compartidos con otros procesos.

(B) En el contexto de una resolución espacial limitada, la súper-resolución abarca métodos que permiten incrementar la resolución espacial más allá de la resolución nativa del sensor y de las mejoras ofrecidas por los métodos de interpolación. Sin embargo, la complejidad computacional introducida por el procesamiento adicional requerido para el proceso de SR para proporcionar resultados es muy alta. No todos los algoritmos pueden ser mapeados de forma eficiente en hardware, lo que lleva a un uso ineficiente de los recursos que supone un rendimiento limitado. Por lo tanto, la capacidad de proporcionar implementaciones hardware de la mayoría de los algoritmos SR capaces de producir una calidad de salida satisfactoria con prestaciones de tiempo real sigue siendo un desafío.

(C) Es habitual es que el nivel usado en la descripción a nivel de registros/pin-accurate se cree a partir de cero. El diseñador codifica manualmente el sistema directamente desde la especificación funcional proporcionada y de la descripción algorítmica. Debido a la relación no propagativa entre las implementaciones software y hardware, la propagación de los cambios introducidos en el software de base, así como la evaluación de su impacto en el rendimiento del sistema, es complicado y requiere una gran cantidad de tiempo y esfuerzo. Estos aspectos podrían ser aliviados mediante el uso de un enfoque multinivel asistido por ordenador en lugar de la conversión manual.

Como ya se ha señalado, las implementaciones basadas en software no son capaces de proporcionar un rendimiento de alta-resolución en tiempo real cuando se mapean sobre las tecnologías disponibles. Las actuales implementaciones hardware disponibles con prestaciones en tiempo real proporcionan una calidad de salida menor que sus correspondiente implementaciones software. Basándonos en nuestra experiencia con algoritmos de compresión, encontramos muchas similitudes entre la naturaleza de los procesos llevados a cabo por SRIR y los algoritmos de compresión, así como entre los problemas a los que se enfrentan sus implementaciones. Las implementaciones hardware de estos algoritmos pueden proporcionar soluciones sobre la forma de implementar algoritmos que procesan imágenes de tal forma que sus implementaciones hardware puedan lograr prestaciones en tiempo real.

Esta Tesis Doctoral se centra en proporcionar un ejemplo de algoritmo de procesamiento de imágenes y de las modificaciones necesarias para mitigar los problemas antes mencionа-

dos, lo que podría resultar en implementaciones hardware de algoritmos de mejora de la resolución de las imágenes basados en súper-resolución, siendo lo suficientemente eficientes como para permitir su uso en las aplicaciones más relevantes de electrónica de consumo. Este algoritmo no sólo debe proporcionar un rendimiento en tiempo real y una la calidad observada de salida súper-resuelta que coincida con el nivel proporcionado por el software encontrado en el estado del arte, sino que también debe facilitar una implementación hardware eficiente. Sólo cuando se logren todos estos requerimientos, se podrá encontrar la manera de llevar las mejoras de resolución de alta fidelidad desde las instalaciones de investigación hasta las casas de los usuarios.

B.2 Imágenes de súper-resolución y secuencias de vídeo

La súper-resolución comprende el conjunto de algoritmos que trata de reconstruir imágenes de alta-resolución corruptas debido a las limitaciones del sistema de imagen. Típicamente estas limitaciones surgen en el sistema de sensores y/o en la óptica causando *aliasing* como ocurre en el proceso de muestreo que no cumple los requerimientos especificados por el teorema de muestreo de Nyquist.

El *aliasing* en imágenes digitales suele considerarse como una degradación y los filtros (tanto ópticos como digitales) se diseñan para evitar el *aliasing* en las cámaras digitales. Sin embargo, el *aliasing* contiene además información extra de alta frecuencia con detalles adicionales sobre la escena. Los algoritmos de súper-resolución extraen la información presente en el *aliasing* para reconstruir una imagen de mayor resolución [Mil10].

B.2.1 Aplicaciones de la súper resolución

En este trabajo definimos el objetivo de súper resolución en referencia a las limitaciones del sistema de imagen que las técnicas de SR diseñan para superarlas. Las limitaciones abordadas por los métodos de SR se pueden clasificar en las siguientes tres categorías: (i) *óptica*, (ii) *geométricas*, y (iii) *temporales*. Las limitaciones ópticas se pueden superar mediante la manipulación de las condiciones de la imagen entre las subsecuentes capturas para generar y transferir subconjuntos complementarios de información a diferentes observaciones que se pueden fusionar, usando un método de SR, para formar la imagen final de una mayor resolución, permitiendo efectivamente romper la barrera de difracción que limita los sistemas ópticos de las imágenes. Las limitaciones geométricas del sensor usado se pueden superar efectivamente mediante el incremento de la resolución espacial más allá de la resolución

nativa del sensor usando un procesado algorítmico para mejorar la resolución espacial. En el contexto temporal, los métodos de encapsulado de súper-resolución mejoran la resolución en términos de reducción del *motion blur* y del *motion aliasing* usando información de múltiples observaciones. En la práctica, la SR espacial se lleva a cabo conjuntamente con la SR temporal. La mejora en la unión de estas técnicas se lleva a cabo mediante el uso de múltiples secuencias que han sido capturadas en diferentes tiempos de adquisición, con diferentes relaciones de trama y/o resoluciones espaciales. Los algoritmos que mejoran ambas resoluciones se conocen como SR *espacio-temporales*.

B.2.2 Clasificación de las técnicas de súper-resolución

Debido a su compleja naturaleza y vastas aplicaciones, la súper-resolución ha sido un área muy activa desde su introducción en 1974. Durante los últimos 40 años varios métodos se han propuesto bajo el nombre de súper-resolución. La más reciente y completa taxonomía de métodos de súper-resolución, presentado en [NM14], los clasifica en cuatro familias principales: (i) métodos basados en frecuencia, (ii) métodos basados en interpolación, (iii) métodos basados en regularización, (iv) métodos basados en aprendizaje. Las tres primeras categorías reconstruyen (o súper resuelven) la imagen a partir de un conjunto de imágenes de entrada de menor resolución (contexto multi-imagen), mientras la última categoría, basada en aprendizaje, es capaz de alcanzar el mismo objetivo explotando la información proveniente de una imagen y una base de datos con un conocimiento alcanzado *a priori*.

Los métodos de SR se pueden además clasificar basándose en el contexto de su ejecución, que es el número de imágenes que son requeridas para llevar a cabo el proceso de mejora. Basándonos en este criterio, dividimos los algoritmos de SR en dos grupos: (i) métodos SR multi-imagen (o clásicos) y (ii) métodos de imagen única (o basados en aprendizaje). En la SR basada en aprendizaje, la pérdida de la información de alta resolución se asume que puede estar disponible en forma de un conocimiento adquirido *a priori* y que ha sido aprendido de los pares de ejemplo de baja resolución/alta resolución.

B.2.3 Contextualización del algoritmo NUGP

El algoritmo de proyección de cuadrícula no uniforme pertenecen a la familia de técnicas *directas* que aplican SR en el dominio espacial trabajando en la ejecución de contexto multi-imagen.

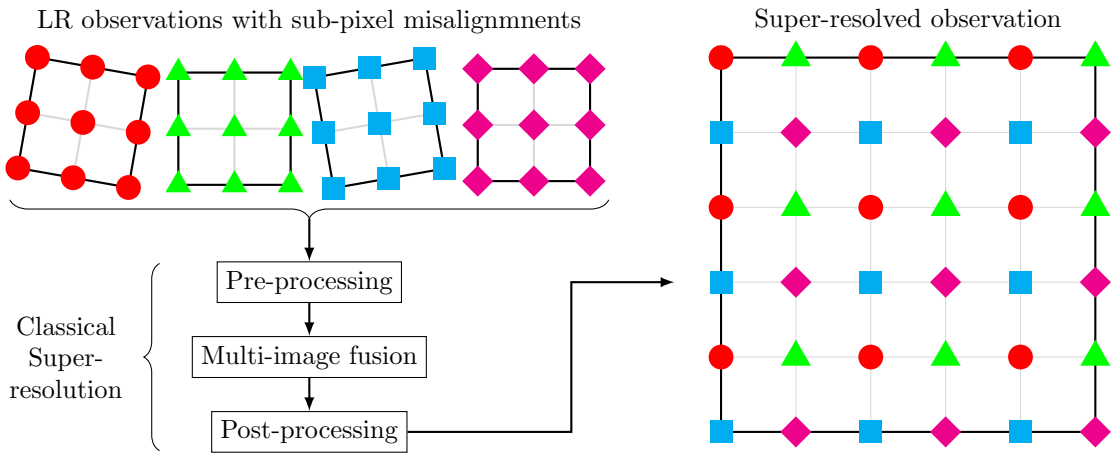


Fig. B.1 Flujo de ejecución de la reconstrucción por súper-resolución multi-imagen basado en desplazamientos sub-pixel.

B.2.3.1 Súper resolución multi-imagen

Los algoritmos de SR clásicos operan basándose en la suposición de que la información de alta frecuencia esta accesible a través de múltiples imágenes de baja resolución, que aparece implícitamente en la forma de *aliasing*. Cada observación de baja resolución se analiza en búsqueda de dicha información complementaria. Si tal información se encuentra, la observación se considera como una restricción lineal sobre los valores desconocidos. Si es suficiente con una observación única de baja resolución, entonces el conjunto de ecuaciones se puede llegar a determinar y puede ser resuelto, llevando a la recuperación de la imagen de alta resolución. Solo las observaciones que no se pueden obtener de las otras se consideran que tienen la propiedad de unicidad. En el caso de usar una cámara, la unicidad de la observación es más probable cuando las observaciones se capturan con desplazamientos sub-pixel. El origen del desplazamiento sub-pixel puede ser causado ya sea por movimientos controlado (e.g. satélites orbitando, jitter de cámara, etc.) o no controlado (e.g. vibración del sistema de imagen al sostener con la mano o el movimiento local en la escena). En la práctica, para que el problema inverso de escalado con factor de escala SR esté bien definido, hacen falta al menos un número $escala^2$ de observaciones que cumplan los requisitos de unicidad. Dado un conjunto de observaciones suficientemente grande, el incremento en la resolución puede ser un valor arbitrario.

El flujo clásico de SR multi-imagen comprende 3 pasos principales: el *pre-procesado*, la *reconstrucción* (fusión multi-imagen) y el *post-procesado*. Dado que el conjunto de imágenes usadas contiene observaciones que son únicas, la reconstrucción clásica de imágenes

por SR se puede llevar a cabo tal y como se muestra en la Fig. B.1. A menos que el movimiento sea conocido *a priori*, antes de combinar las imágenes éstas tienen que ser registradas para determinar las variaciones entre ellas. Desde el momento en el que las imágenes de entrada representen la misma escena y puedan ser registradas de forma satisfactoria, el flujo de SR permite que cualquier método de adquisición sea usado. El pos-procesado depende del núcleo de SR usado y del proceso de aplicación de la SR. La mayoría de los algoritmos del estado del arte llevan a cabo una regularización post-fusión y restauración/reconstrucción.

B.2.3.2 Técnicas directas

La base teórica de las técnicas directas es la teoría de muestreo no-uniforme, la cual permite la reconstrucción de funciones de muestras tomadas de una posición distribuida no uniformemente. Esos métodos siguen la estrategia SR clásica presentada en la Fig. B.1. Dado un conjunto de observaciones de LR, una de las imágenes de LR es elegida como imagen objetivo y las otras son registradas frente a ésta. A continuación, la imagen objetivo es escalada usando un factor de escala específico, y las otras imágenes de LR son mapeadas (es decir: escaladas y desplazadas) dentro de la cuadrícula objetivo, usando la información obtenida en el registro. Los datos perdidos son obtenidos por medio de una interpolación no uniforme. Finalmente, se podría aplicar un núcleo opcional de enfocado/refinamiento a la imagen resultante. El proceso de fusión es normalmente implementado como una suma de valores multiplicados por pesos, y es por lo que esta estrategia normalmente se referencia como *desplazamiento y suma* (shift and add) [FREM03, FREM04b, FREM04a, FM06, AIAOM13].

Muchos de los métodos directos recientes llevan a cabo el registro y la fusión descomponiendo la imagen en zonas menores o bloques. Eso significa que las imágenes de LR son primero divididas en bloques. Luego cada bloque de las imágenes objetivo es comparado con el conjunto de bloques, incluyendo el correspondiente bloque y algunos bloques de su alrededor, en las imágenes LR de referencia. Basándose en la similitud de esos bloques y en sus similitudes con respecto al bloque actual, se asociado un peso a cada bloque, indicando la contribución de ese bloque en el proceso de producir un bloque de salida. Esta estrategia ayuda a manejar mejor una oclusión, local y movimientos complejos (por ejemplo, cambios faciales de la expresión). La familia de métodos que hace uso de este tipo de procesamiento es normalmente referenciada como SR directa *no-paramétrica* [PE09, PETM09, TMPE09, BBV10, CCL10, HS12, ZLRG12].

B.2.3.3 Súper-resolución de secuencias de vídeo

Los métodos existentes para la SR de secuencias de vídeo pueden ser clasificados en las siguientes 4 categorías: (i) métodos basado ventana deslizante de fotogramas (*Sliding Frame Window*, SFW) [SKR07, NHBS07, NSLZ07, PJ07], (ii) método secuencial [EF99c, EF99b, FEM06, CB07], (iii) método simultáneo [BS99, ZM07, AMK03], and (iv) método basado en el aprendizaje [BBM03, DKA04, KHX$^+$06]. Las 3 primeras estrategias requieren un contexto multi-imágenes/fotograma, mientras que la SR basada en aprendizaje es capaz de ejecutarse en el contexto de un solo imagen/fotograma. Considerando asegurada la disponibilidad de las secuencias de vídeo, la técnica de SR multi-imagen constituye la estrategia preferente, especialmente en el contexto de una implem entación hardware.

Las estrategias multi-imagen requieren de un cierto número de fotogramas para ejecutarse. Este conjunto de fotogramas, llamado *ventana de fotogramas* (o *ventana de trabajo*), forma el contexto de la ejecución. La ventana de fotograma comprende dos tipos de imágenes: la imagen objetivo que está siendo súper-resuelta y un número de *imágenes de referencia* (o *fotogramas de referencia* (RF)).

Una consecuencia de usar un contexto multi-imagen para la ejecución del algoritmo, es necesario registrar los fotogramas pertenecientes al contexto actual. En el contexto de SR de secuencias de vídeo hay dos tipos principales de estrategia que están siendo utilizadas para el registro de imagen y actualización del contexto: el método de *fijación* y el método *progresivo*. En el método de fijación, uno de los fotogramas del contexto es elegido como el fotograma de referencia y los otros fotogramas desalineados se registran en relación a este fotograma. En el caso del registro progresivo, el fotograma actual es registrado en relación a sus vecinos temporales inmediatos a partir de los fotogramas anteriores.

El algoritmo NUGP implementa la estrategia de SR basada en la ventana deslizante ilustrada en la Fig. B.2. Esta estrategia utiliza un registro fijo sobre un conjunto de fotogramas de baja resolución consecutivos, los cuales son combinados más tarde produciendo un fotograma de alta resolución. El mayor inconveniente de esta estrategia es que la correlación temporal entre las sucesivas imágenes de alta resolución reconstruidas queda prácticamente sin explotar. Además, la ocupación de memoria asociada con esta estrategia es alta, debido a que todos los datos requeridos del contexto tienen que estar disponibles para su lectura de la memoria.

Normalmente se requiere que el procesamiento SR de secuencias de vídeo sea *dinámico*, es decir, que produzca una secuencia de salida que (al menos) mantenga el número y la tasa de fotogramas de la secuencia de entrada. Debe notarse que la *SR dinámica* de secuencias de vídeo tiene una demanda significante mayor en términos de rendimiento, velocidad de

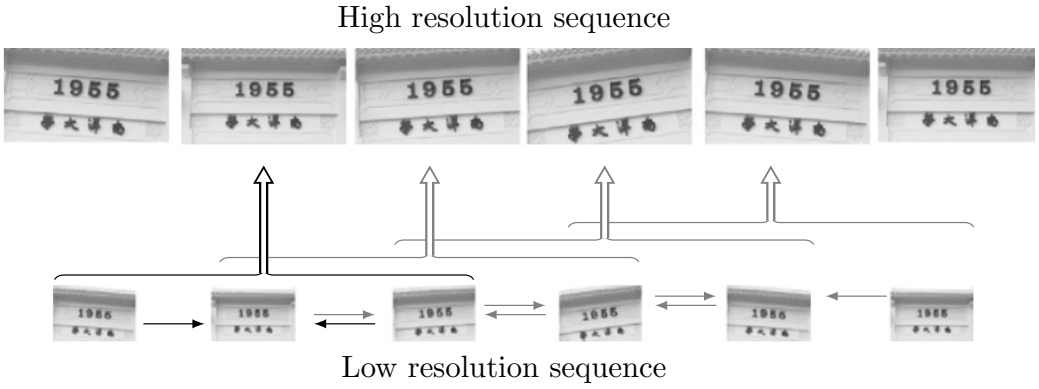


Fig. B.2 Estrategia de ventana deslizante de fotogramas para la SR de secuencias de vídeo ejecutándose en un contexto multi-fotograma. Basado en [TM11].

ejecución y memoria ocupada comparada con la SR de imágenes estáticas [Goh13]. En el contexto de la SR dinámica de secuencias de vídeo, a causa de su enorme carga computacional, se usan algoritmos de menor complejidad para poder mejorar la velocidad de ejecución y por tanto facilitar la posibilidad de poder llevar a cabo el procesamiento en el tiempo destinado para ello. Las implementaciones de los algoritmos basados en interpolación, los métodos *directos* y los métodos *IBP* (*Iterative Back Projection*), resultan ser más satisfactorios con respecto a las limitaciones de ejecución en tiempo real.

B.3 El algoritmo de proyección sobre la cuadrícula no-uniforme

En este trabajo se aborda el desafío de proporcionar la implementación de un algoritmo de SR capaz de procesar vídeo y trabajando en tiempo real, usando un algoritmo de proyección sobre una cuadrícula no-uniforme , tal y como el que ha sido propuesto en [MC03].

B.3.1 Flujo de ejecución del algoritmo NUGP

NUGPA es un algoritmo de fusión que utiliza los datos no redundantes encontrados en un conjunto de imágenes afectadas por deformación asociada con el movimiento y por el *aliasing* causado por la limitación en banda de los sensores. Usando la clasificación mencionada, NUGPA encaja con la fusión basada en la interpolación de la familia de métodos directos que aplican la transformación en el dominio espacial, usando para ello múltiples

imágenes. La principal ventaja de esta clase de algoritmos de SR (la baja carga computacional y que hacen posible crear aplicaciones en tiempo real) tiene como contrapartida un modelo de degradación limitado. NUGPA lleva a cabo la reconstrucción de la imagen de súper-resolución en tres etapas: (1) la estimación de movimiento, (2) la reconstrucción basada en la fusión y (3) la interpolación no-uniforme.

B.3.1.1 Estimación de movimiento

En la primera etapa se estima la función de deformación y las métricas que la caracterizan. Con el objetivo de registrar las imágenes y encontrar las regiones con mayor probabilidad de contener información adicional, el algoritmo de súper-resolución NUGP utiliza una versión de la estimación de movimiento basada en coincidencias de bloques. Durante el registro de la imagen un conjunto de píxeles que forma un bloque (en adelante, un *macro-bloque* (MB)) del fotograma que está siendo procesado se compara con los bloques de otros fotogramas de la SFW. Para cada MB sólo los píxeles que pertenecen a un conjunto limitado (llamado *área de búsqueda*, SA) y delimitado a cierta proximidad espacial (llamado *radio del área de búsqueda*, SAR; expresado en número de píxeles) participa en el proceso de creación de un conjunto de candidatos que serán usados en el proceso de registro de la imagen. El proceso de estimación de movimiento calcula los llamados *híper-datos* que, en nuestro caso constan del *vector de movimiento* (MV) y de los índices de similitud asociados basado en la *suma de diferencias absolutas* SAD (Sum of Absolute Differences). Estos vectores de movimiento son vectores que identifican al MB cuyos índices de similitud tienen el valor más próximo a los que se buscan. En la B.3 se muestra un ejemplo de coincidencia de bloques para un SFW que contiene dos fotogramas de referencia, donde el radio del área de búsqueda es igual al ancho del MB (MB_{width}), y contiene $2 \times MB_{width} + 1)^2$ candidatos (para mayor claridad sólo se muestran nueve candidatos).

B.3.1.2 Reconstrucción basada en la fusión

Los *híper-datos* generados durante la etapa de estimación de movimiento se introducen en el núcleo de SR. EL proceso de reconstrucción se inicia con la creación de la cuadrícula HR. Las dimensiones de la cuadrícula se determinan en base a la precisión usada en el registro de imágenes (en lo sucesivo $precision_{me}$). En primer lugar, la cuadrícula de HR se rellena con los píxeles del fotograma de LR que está siendo procesado. Las nuevas coordenadas espaciales de HR se calculan multiplicando las coordenadas LR por la precisión de la estimación de movimiento.

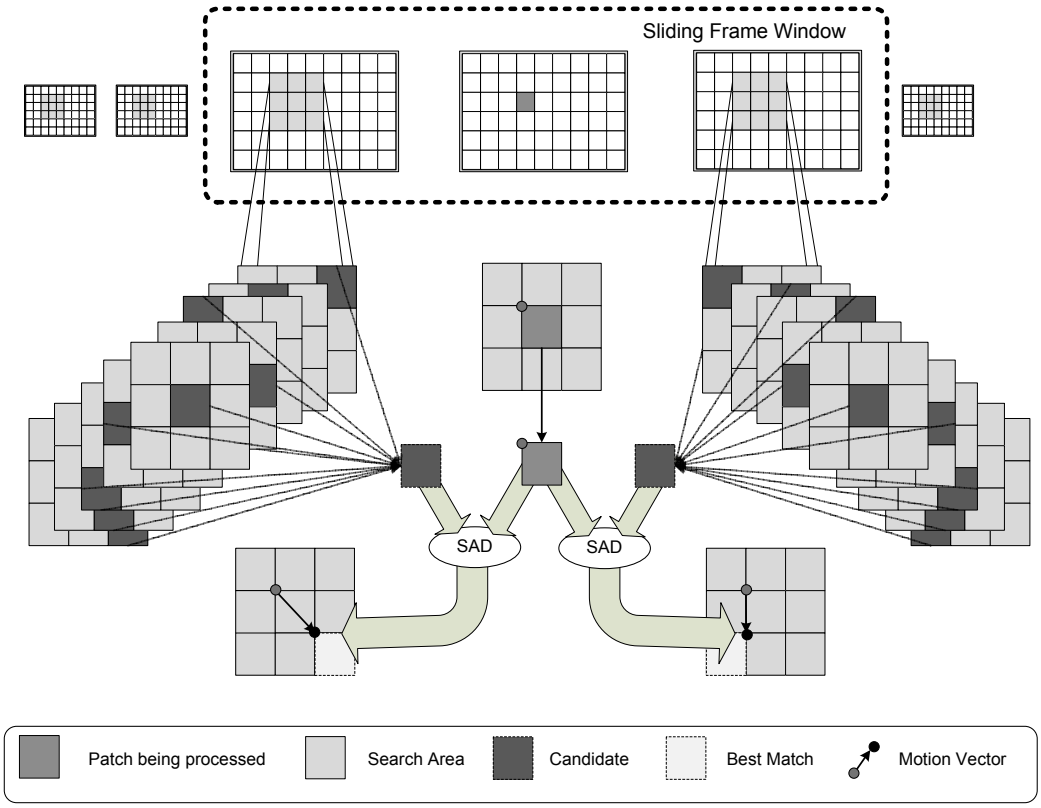


Fig. B.3 Estimación de movimiento basada en la coincidencia de bloques utilizado por NUGPA usando el criterio de similitud basado en la suma de diferencias absolutas SAD (Sum of Absolute Differences).

Tras colocar todos los píxeles LR en la cuadrícula de HR, se consideran los píxeles que provienen de un conjunto de los fotogramas contenidos en el SFW. En este punto, la mayoría de las coordenadas de la cuadrícula de HR no contienen datos válidos. Esas coordenadas se conocen como *huecos* (*holes*). Los valores de los huecos están creados en base de la fusión de los datos extraídos de los fotogramas de la ventana de trabajo. Durante la fusión, los híper-datos recibidos (es decir, MVs y SADs) determinan qué datos contribuirán a la estimación del valor de los huecos y con qué peso. Se permite más de un valor para tomar parte en el proceso de la formación de un nuevo valor súper-resuelto del hueco.

B.3.1.3 Interpolación no-uniforme

En la mayoría de los casos, no todos los huecos de la cuadrícula de HR se rellenan durante la fusión. A los huecos que no han obtenido un valor durante el proceso de fusión se les

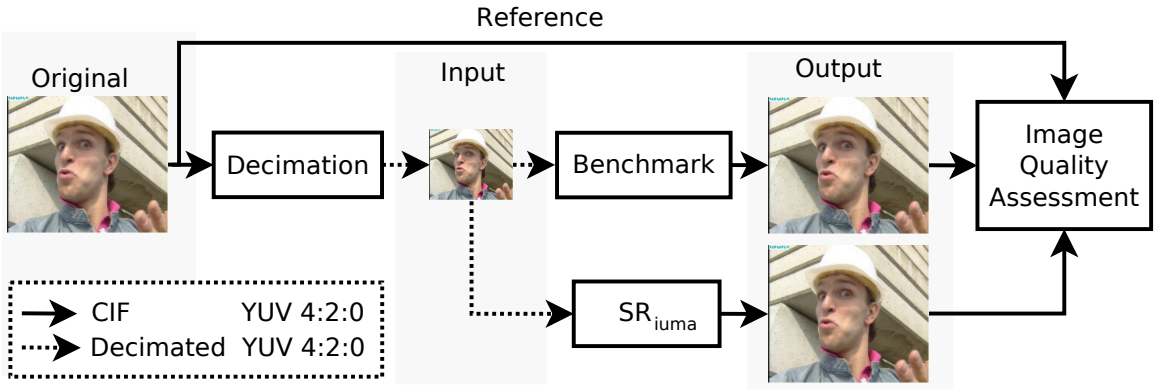


Fig. B.4 Flujo de datos del experimento utilizado para evaluar la calidad de la salida de SR.

asigna un valor estimado mediante interpolación. Por último, la cuadrícula de HR post-interpolación se ajusta a las dimensiones esperadas especificadas por el factor de escala (en lo sucesivo *scale*). La escala determina la relación entre la dimensión de la imagen de salida súper-resuelta y la dimensión de la imagen de entrada de baja resolución. Para valores de escala más pequeños que el valor de la precisión (ME) debe reducirse el muestreo de la cuadrícula de HR. Cuando los valores de los parámetros anteriormente mencionados son iguales, el ajuste no es necesario y la cuadrícula post-interpolación se convierte directamente en la imagen súper-resuelta producida por NUGPA.

B.3.2 Evaluación cuantitativa de la calidad de la imagen súper-resuelta del algoritmo NUGP

Para evaluar el rendimiento en términos de la calidad de la salida percibida, se aplica el NUGPA a los componentes de luma (componente Y del formato YUV) y se utiliza el flujo que se muestra en la Fig. B.4.

La secuencia de referencia (y por lo tanto también la de salida) será una secuencia en formato YUV 4:2:0p de 8 bits y tamaño CIF (288x352 píxeles). El uso de este formato supone la entrada de LR correspondiente al formato QCIF (144x176 píxeles) o un formato 72x88 píxeles para los valores del factor de escala de la súper-resolución (scale) de 2 y 4, respectivamente. El caso común es utilizar imágenes de todavía menor resolución (32x32, 64x64, 128x128) [ESHEK12]. Las secuencias utilizadas en los experimentos se describen brevemente en la Tabla B.1. Estas secuencias forman parte de la *'Xiph.org video Test Media - derf's collection'* disponible on-line [Xip15].

En la Tabla B.2 se muestra la media de valores de PSNR de 90 fotogramas iniciales

TABLE B.1 *Secuencias de test utilizadas en los experimentos.*

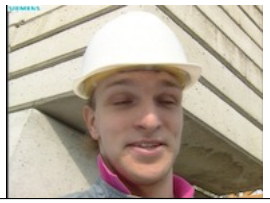
Sequence	Visualization	Description
Foreman		Presenta un obrero realizando rápidos movimientos de la cabeza al azar mientras habla a la cámara. La secuencia cambia a presentar el sitio de trabajo. Esta secuencia presenta movimiento local y global rápido y un cambio de contexto acentuado.
Mobile		Presenta un tren eléctrico en movimiento en la mesa, un sistema planetario en forma de móvil y un calendario colgado en la pared. El tren se mueve horizontalmente, mientras que el calendario se mueve de arriba abajo presentando un movimiento vertical. Una bola que gira presenta una mezcla de ambos tipos de movimiento. Existe un movimiento global ya que la cámara se mueve hacia la izquierda. Esta secuencia contiene gran cantidad de textura con muchas detalles presentes.
Paris		Presenta una pareja hablando sentada junto a una mesa. Mientras hablan, la mujer está tirando una pequeña pelota al aire, y el hombre está haciendo mover un lápiz entre sus manos. La secuencia presenta movimientos locales rápidos. El movimiento global está ausente.

TABLE B.2 *PSNR observado de la secuencia de salida (media sobre los 90 fotogramas iniciales; CIF; componente luma; MBwidth = 4, RF=4 (2+2), SAR = 16, y scale=2).*

Implementation	Media de PSNR [dB]		
SR_{iuma}	**22.38**	**29.42**	**22.79**
IBP con pesos (20 iteraciones) [BB08]	20.46	27.80	22.22
Nearest-neighbor interpolation	18.78	26.68	19.89
Bilinear interpolation	20.46	28.07	21.23
Bi-cubic interpolation	20.29	28.10	20.92
	mobile	foreman	paris

de las secuencias paris, foreman y mobile (componentes de luma) mejoradas usando varios métodos de SR y de interpolación. La Tabla B.2 proporciona una comparación con el estado del arte de las implementaciones hardware [BB08], demostrando que NUGPA es capaz de proporcionar imágenes de muy alta calidad. En nuestros experimentos solo se ha comparado la calidad del componente luma (Y) el cual se ha llevado a cabo el proceso de súper-resolución. Los resultados presentados en [WBS05] indican que la incorporación de los componentes de color no altera el rendimiento del modelo de evaluación de una forma significativa.

B.3.3 Flujo de ejecución con granularidad fina

Los principales factores que limitan la implementación hardware son la naturaleza iterativa de los procesos y el consumo de recursos. La selección del algoritmo NUGP soluciona la primera limitación, permitiendo una implementación hardware eficiente. Desafortunadamente, la ocupación de memoria de la versión no iterativa del algoritmo mostrado en el capítulo anterior hace prohibitiva su implementación en una FPGA que no emplee memoria externa.

La referencia software usada originalmente (software *SRiuma*) trabajaba a nivel de fotograma y por este motivo sus requerimientos del almacenamiento en memoria son muy elevados, resultando en que la implementación hardware usando solo bloques de memoria RAM (*Block RAM*, BRAM) sea inviable. Una rápida evaluación de la referencia software y del flujo de datos del algoritmo identifica los buffers de fotograma completo usados por el núcleo de fusión como los principales consumidores de memoria. El flujo de ejecución implementado requiere que al menos dos de los fotogramas que están siendo procesados estén disponibles para ser leídos en la memoria durante el tiempo en que se lleva a cabo el proceso de fusión. Por lo tanto, para eliminar los buffers de fotogramas completos el flujo de ejecución necesita ser modificado. En concreto, la granularidad del flujo de datos debe ser 'refinada' para reducir los requerimientos de almacenamiento en memoria del algoritmo.

Para afrontar este problema hemos propuesto un flujo que internamente aplique un proceso completo de SR a un pequeño conjunto de píxeles (llamado macro-bloque, MB) y que por lo tanto represente un punto medio entre los flujos de proceso con granularidad "gruesa" y "fina". En comparación con el procesamiento a nivel de fotograma, el cual requiere que la representación de HR de todos los fotogramas de la SFW estén disponibles en la memoria de salida durante todo el proceso de súper-resolución, la estrategia a nivel de macro-bloque requiere que solo las áreas de búsqueda (SA), en lugar de todo el fotograma,

estén disponible en la memoria. Como resultado, se reducen los requerimientos de almacenamiento de memoria con el inconveniente de incrementar el número de accesos a memoria. El procesado de un bloque de píxeles en lugar de un solo píxel permite optimizar la lógica de síntesis, consiguiendo así que ocupe un menor espacio y permitiendo la implementación del algoritmo completo en el dispositivo FPGA seleccionado.

El flujo de ejecución propuesto para el NUGPA es similar al flujo a nivel de fotograma. La principal diferencia entre los flujos son: (i) los tamaños de los elementos básicos guardados en las memorias locales, (ii) la sustitución de los fotogramas de referencia por áreas de búsqueda, (iii) el hecho de que la transición de un paso del algoritmo al siguiente se produce cuando un macro-bloque, y no un fotograma, es procesado, y, (iv) la necesidad de tener en cuenta dependencias entre macro-bloques, lo cual no ocurre cuando el flujo de ejecución se lleva a cabo a nivel de fotograma, y que complica significativamente el proceso de la localización y rellenado de los huecos (*holes filling*). La elección del flujo de ejecución ha supuesto un impacto importante en las características de implementación, como la ocupación de la memoria y la lógica, el rendimiento y la escalabilidad del sistema, entre las principales. Estos cambios resultan en diferentes estrategias de almacenamiento en memorias intermedia, incrementan la complejidad del manejo de la memoria y mitigan el incremento en el tráfico de la memoria.

B.3.4 Evaluación del flujo de ejecución a nivel de macro bloque

Los factores de reducción en ocupación de la memoria y del incremento en el tráfico de las comunicaciones asociado con el cambio de flujo realizado desde el nivel de fotograma al nivel de macro-bloque, han sido cuantitativamente evaluados. También han sido cuantitativamente evaluados el factor de reducción de ocupación de la memoria (*Total Memory storage Requirements*, TMR) y del incremento del tráfico de las comunicaciones (*Total Memory Access Count*, TMAC), asociados con el cambio del flujo de nivel de fotograma a nivel de macro bloque.

B.3.4.1 Estudio de la metodología empleada

El estudio de la reducción en la ocupación de memoria debido al cambio de flujo desde el nivel de fotograma al flujo a nivel de macro-bloque se ha llevado a cabo tal y como se explica a continuación. En primer lugar, se ha identificado la memoria empleada en cada etapa del algoritmo, y se ha clasificado dentro de cada uno de los tipos de memoria definidos. A continuación, para cada uno de los tipos de memoria definidos, se ha evaluado el tamaño de

la memoria y las dependencias con los parámetros del algoritmo.

Para evaluar el impacto del cambio de flujo desde nivel de fotograma al flujo a nivel de macro-bloque, en el tráfico de la memoria se han introducido, en primer lugar, figuras para modelar los accesos condicionales a la memoria. A continuación, se han identificado los patrones de acceso a memoria de cada etapa del algoritmo, así como sus dependencias con respecto a los parámetros del algoritmo. Esto ha llevado a la creación de ecuaciones generales para modelar el número de accesos a memoria realizado en cada etapa del algoritmo.

Basándonos en las ecuaciones definidas, la ocupación de memoria y el número de accesos a memoria de cada etapa del algoritmo han sido cuantificados, para las dos estrategias de flujo de ejecución (flujo a nivel de fotograma y flujo a nivel de macro-bloque) para las 96 combinaciones de los parámetros del algoritmo NUGP. El conjunto de las 96 combinaciones empleadas para llevar a cabo las pruebas ha sido construido combinando los distintos valores de tamaño de macro-bloque (MB_{width} = 4, 8 y 16), radio del área de búsqueda (SAR = 2, 4, 8 y 16), número de fotogramas de referencia soportados (RF = 2, 4, 8 y 16), y el aumento del factor de escalado espacial (scale = 2 y 4).

B.3.4.2 Evaluación cuantitativa de la reducción de la memoria y del aumento del tráfico

Los resultados de este estudio han demostrado que, para el formato de fotograma QCIF, el cambio desde el nivel de fotograma al esquema de ejecución a nivel de MB, puede conducir a una reducción de la memoria en un factor de entre 6, 8 y 40, y un incremento en los accesos totales a memoria en un factor de entre 1, 22 y 117, dependiendo de los valores de los parámetros del algoritmo NUGP. Los requisitos de memoria totales mínimos y máximos calculados para el flujo de MBL son de 122 KB y de 1051 KB.

Dado que los conjuntos de parámetros del algoritmo que se han analizado y que se han utilizado en ambos estudios han sido los mismos, es posible confrontar los valores de ambos factores y analizar la relación de la reducción de la memoria frente al aumento del tráfico (un factor adecuado de mérito, en adelante TMR/TMACratio) como una función de los valores de los parámetros del algoritmo. El estudio realizado ha demostrado que para 20 de las 96 combinaciones probadas, el factor observado de aumento en el tráfico de memoria ha sido mayor que el factor de reducción observado en la ocupación de la memoria. Esto era esperado, dado que el rango de valores calculados anteriormente (de 1,22 a 117) es mayor que el rango de valores finalmente calculados (de 6,8 a 40). Un análisis de estas 20 combinaciones ha demostrado que en trece de ellas se ha observado un tamaño de MB 4, lo que significa que las ventajas aportadas por la granularidad más fina del flujo del SRIR

a nivel de MB están limitadas para los casos de tamaño de MB más pequeño y valores de SAR superiores a 8. El crecimiento exponencial del tamaño de las memorias utilizadas para la gestión del reordenamiento de fotograma a MB para las métricas del ME ha sido identificado como la principal causa de esta limitación.

Por lo tanto, las optimizaciones en la reutilización de datos y las políticas de buffering deben centrarse en reducir el tráfico de la memoria sobre todo para las combinaciones de valores de los parámetros SRIR antes mencionadas.

B.3.4.3 Optimización a nivel algorítmica

Un análisis de los patrones de acceso a memoria ha mostrado que la gran mayoría de los accesos a la memoria durante la construcción de las rejillas de alta resolución y el proceso de extracción de los píxeles (de RFs) se lleva a cabo para coordenadas de alta resolución que no llevan valores de píxeles de baja resolución sino un valor constante de '0'. El mapeo de las coordenadas de LR a HR (y su inverso) para los píxeles de LR depende sólo de los valores de las coordenadas de LR (conjunto ordenado de valores fijos y limitado) y de la resolución de la estimación de movimiento (valor fijo) y se determina tan pronto como se determinan estos valores. Por lo tanto, es posible distinguir (en tiempo de ejecución) entre las coordenadas que identifican los píxeles de LR y los que no apuntan a píxeles LR (y por lo tanto, conteniendo el valor constante de '0') dadas sólo las coordenadas HR. Además, el valor adecuado del píxel de LR se puede cargar directamente de las representaciones de LR de las estructuras de referencia y del MB que está siendo procesado.

La identificación de los píxeles que no son de LR ha sido implementada mediante la introducción de un nuevo esquema de direccionamiento que es capaz de determinar si la coordenada HR a extraer direcciona o no un píxel de LR. Si es así, el píxel se carga desde la memoria de LR. De lo contrario, al píxel se le asigna un valor constante sin realizar el acceso real a memoria. La desventaja es un (ligero) incremento de la complejidad computacional del proceso de fusión. La introducción del esquema de direccionamiento descrito ha posibilitado una reducción significativa en el tráfico de la memoria para los dos flujos de ejecución. En comparación con las versiones de referencia (es decir, sin optimizaciones) el mínimo, máximo y el recuento medio observado para los accesos a memoria se han reducido en un 13,8%, 13,3%, 12% y 25,6%, 87,4%, 81%, respectivamente, para los flujos FL y MBL respectivamente. La mayor tasa de reducción en el recuento de accesos observado para el flujo de grano fino ha rebajado los factores del incremento del tráfico de memoria mínimo y máximo asociada con el cambio en el flujo de MBL a 1,1 y 16,9, respectivamente, de 1,22 y 117. Esta es una optimización que ofrece interesantes compromisos para la implementación

del diseño con memoria local a medida y manteniendo el tráfico de las comunicaciones.

La implementación del nuevo esquema de direccionamiento ha permitido eliminar la mayoría de los contenidos de alta resolución. También ha disminuido la ocupación total de memoria, tanto para el flujo de FL como de MBL. La ocupación de memoria máxima se ha reducido de 9158 KB y 1051 KB a 2475 KB y 718 KB, respectivamente, para las implementaciones a nivel de fotograma y de macro-bloque. La ocupación mínima desciende desde 3317 KB y 122 KB a 1832 KB y 113 KB, respectivamente. En resumen, la optimización introducida ha rebajado el mínimo, máximo, y la ocupación media de la memoria en un 73%, 44,8%, 62,5% y 31,7%, 6,9%, 22,7%, respectivamente para el flujo FL y MBL. En consecuencia, el nuevo esquema de direccionamiento también ha reducido los valores mínimos y máximos observados del factor de reducción en la ocupación de memoria para el cambio al flujo de MBL a 3,5, y 16, respectivamente (de 6,8 y 40). A pesar de la notable disminución, el cambio de flujo desde el nivel de fotograma al de MB es todavía lo suficientemente significativo, manteniendo el tráfico de memoria, como para mantener la aproximación MBL como la más adecuada.

Todo lo anterior ha reducido aún más el número de combinaciones de prueba para las que el incremento en el recuento de accesos a memoria es todavía mayor que la reducción lograda en la ocupación de memoria. Una superposición de los factores de reducción en la ocupación de memoria frente al factor de aumento en el tráfico de memoria calculado para las versiones de referencia y optimizada se presenta en la Fig. B.5. La figura muestra que para una escala de 4 (peor caso) el número de combinaciones para las que el incremento observado en el tráfico de memoria es mayor que la reducción de la ocupación se ha visto reducido de 12 a 5. Teniendo en cuenta todas las 96 combinaciones de prueba, el número de estas combinaciones se ha visto reducido de 20 a 9. Hay que tener en cuenta que para estas combinaciones el tamaño de la memoria todavía se reduce significativamente, sin embargo, se espera un aumento moderado en el tráfico que tiene que afrontarse o tolerarse.

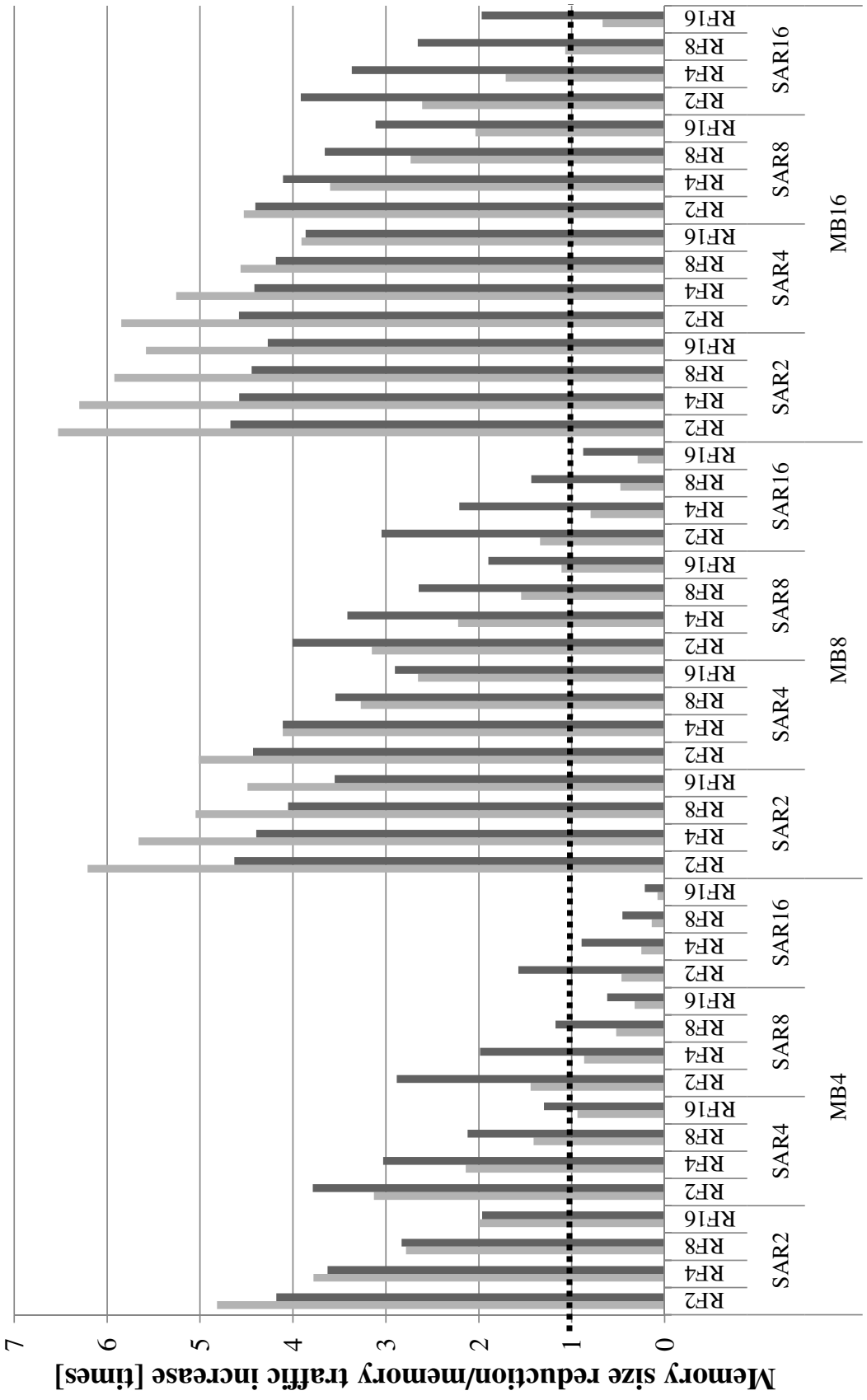


Fig. B.5 Factor de reducción de la ocupación de la memoria frente al factor de incremento del tráfico de memoria para las implementaciones inicial y optimizada a nivel de macro-bloque (para un factor de escala=4).

B.4 Metodología de diseño y verificación

El objetivo de cualquier implementación hardware es crear una descripción del sistema que, cuando se implemente en el dispositivo de destino, resulte en un sistema hardware que tenga el mismo comportamiento observable que la referencia proporcionada, al tiempo que se alcanzan los requisitos de rendimiento establecidos. En el flujo de diseño típico primero se analiza el algoritmo de referencia, arquitecturalmente limitado, y luego se usa para crear una descripción a nivel de registro y pin-accurate usando un lenguaje de diseño de hardware y siguiendo un método de implementación específico. Esta descripción puede ser luego utilizada para crear una descripción a nivel de puertas utilizada para configurar el hardware de destino. En la mayoría de las implementaciones hardware la funcionalidad completa del sistema se prototipa primeramente en software. Este paso intermedio se utiliza para probar la funcionalidad del algoritmo y proporciona una referencia de alto nivel de abstracción para ser utilizada en la validación y las pruebas.

B.4.1 Visión general de la metodología de diseño y verificación

En la implementación propuesta, se ha optado por seguir el segundo de los flujos antes mencionados, referenciado en la literatura como metodología a nivel de sistemas electrónicos (electronic system level, ESL). Para acelerar la implementación hardware del núcleo de súper-resolución se ha optado por la metodología de verificación del diseño presentada en la Fig. B.6. La metodología establecida se ha basado en la metodología de co-desarrollada por el autor y utilizada con éxito en los proyectos descritos en [SHFC$^+$08, TSC$^+$09, CEN$^+$13b, CEN$^+$13a]. La metodología sigue el flujo ESL típico. En primer lugar, el código de referencia proporcionado es usado para crear un modelo funcional de alto nivel de abstracción. Luego, este modelo se refina de forma iterativa, y, usando una metodología de síntesis de alto nivel, se transforma en una micro-arquitectura RTL. Este último paso se realiza para crear el archivo de configuración hardware necesario mediante síntesis lógica.

B.4.1.1 Flujo de diseño

En este caso de estudio, el código base proporcionado había sido escrito en ANSI C. Este código ha sido instrumentalizado y utilizado como base para el análisis a nivel funcional. Los resultados de la etapa de análisis permitieron determinar la carga de trabajo de cada estructura funcional y guiaron la especificación de los prototipos de los módulos hardware, así como de la partición hardware/software. A continuación, las funciones C se agruparon

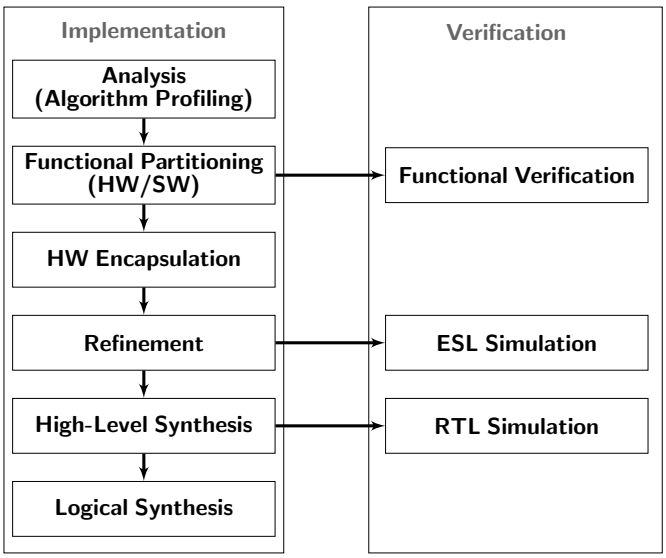


Fig. B.6 Visión general de la metodología de implementación.

por funcionalidad y se encapsularon como módulos SystemC. Mediante el uso de lenguaje SystemC, basado en C, se permite el flujo de ejecución y la mayoría del código de software permanece inalterado durante la transformación a la representación intermedia. En el caso del núcleo de súper-resolución, se decidió que sólo el núcleo de súper-resolución sería implementado.

Se ha implementado el núcleo de SR en hardware, dejando fuera el bloque de estimación de movimiento, dado que es un problema bien definido y conocido para la que ya existen implementaciones independientes [HN10]. Durante la partición funcional se decidió que el tratamiento de los componentes de croma (U y V en el formato YUV) no se llevaría a cabo utilizando los recursos de la FPGA, sino que se llevaría a cabo en paralelo usando uno de los microprocesadores del SoC. Los valores de los componentes de la crominancia están simplemente interpolados.

La funcionalidad encapsulada por el núcleo de súper-resolución (luma) debe ser iterativamente refinada con el fin de permitir su síntesis desde alto nivel. Por el contrario, la parte funcional del bloque de estimación de movimiento y la funcionalidad de los módulos de procesamiento de croma no se refinan después del encapsulamiento inicial. Sólo se cambia el código que lleva a cabo la comunicación a fin de aplicar los cambios introducidos en el núcleo de súper-resolución. La mayor parte de las decisiones de diseño se han realizado durante las etapas de partición y refinamiento, incluyendo el análisis del impacto en la ocupación de los recursos y la latencia de la ruta crítica. Las iteraciones para el refinamiento

se han guiado mediante las mediciones de rendimiento (latencias, tiempos de ciclo y utilización de recursos) obtenidas a partir de la síntesis de alto nivel. Esto asegura la mejor calidad de los resultados del sistema final.

Mientras que los lenguajes HDL se utilizan a menudo para las descripciones de los Niveles de Transferencia de Registros (Register Transfer Level, RTL), SystemC se aplica generalmente al modelado a nivel de sistema, la exploración de la arquitectura, el desarrollo del software, la verificación funcional y la síntesis de alto nivel [Age08]. Con el fin de obtener la representación HDL del sistema, la descripción TLM tuvo que ser refinada en una descripción sintetizable. Una descripción SystemC se considera sintetizable si se utiliza un subconjunto sintetizable de lenguaje SystemC. El conjunto de construcciones consideradas sintetizables es específico para cada herramienta de síntesis del alto nivel. En nuestro caso, la preparación del modelo sintetizable para la síntesis de alto nivel implicaba la sustitución de las construcciones del lenguaje no soportadas por el compilador SystemC elegido. Entre las modificaciones necesarias, las más importantes fueron: la eliminación de la asignación de la memoria dinámica, la reducción dimensional de la memoria a una dimensión, la eliminación de los punteros aritméticos, la ruptura de los bucles combinacionales, la expansión de funciones en el momento de su referencia (*function inlining*), la asignación de matrices en memoria BRAM y la eliminación de datos en punto flotante. Una vez que el código fue sintetizado, se sintetizó una descripción RTL del núcleo y se utilizó en la síntesis lógica.

La verificación del diseño ha sido planeada para cada nivel de abstracción jerárquica, es decir, para los niveles: funcional/algorítmico, *nivel del sistema electrónico* ESL (*Electornic System Level*) y RTL.

Verification of the design has been planned for each tier of the abstraction hierarchy, namely, the functional/algorithmic, ESL and RTL tiers.

B.4.1.2 Validación del modelo algorítmico

En la etapa algorítmica, la descripción del sistema estuvo representada por las definiciones de funciones, las clases y el flujo de control secuencial codificado en lenguaje C. La verificación requiere los datos de simulación de las implementaciones de software a nivel de fotograma y a nivel de MB. Estos datos, una vez obtenidos, se compararon con el dato proveniente de la salida de referencia.

La introducción de los adaptadores de granularidad fotograma a macro-bloque permitieron sustituir el núcleo de súper-resolución que llevaba a cabo el procesamiento a nivel de fotograma con el que se procesaban los datos MB a MB sin necesidad de tener que hacer cambios en otras partes del sistema. Esto facilitó la reutilización de las funciones

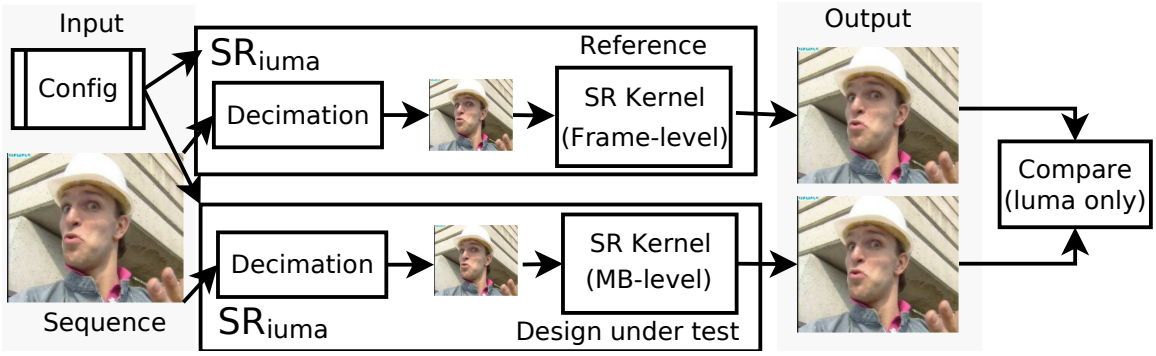


Fig. B.7 Flujo de datos del experimento utilizado para verificar el modelo funcional.

responsables de la carga/almacenamiento de las imágenes, el diezmado de la imagen y la estimación de movimiento. Este hecho condujo a la configuración utilizada en la etapa de validación que se presenta en la Fig. B.7. El formato de la imagen de referencia (y por lo tanto de salida) es una secuencia YUV 4:2:0p con codificación de 8 bits y tamaño CIF (288x352 píxeles). La resolución de entrada de LR depende del valor del factor de escala de súper-resolución. Una vez el procesamiento se llevó a cabo, las imágenes de salida producidas por la implementación se compararon con las imágenes del software de referencia. Para considerar que la verificación era correcta, se buscaba que los resultados observados fueran idénticos para las luminancias de todos los fotogramas. Sólo después de que las salidas generadas por todas las simulaciones hubieran sido idénticas, el modelo del sistema fue considerado como verificado.

B.4.1.3 Validación del modelo ESL

Una vez validado el modelo funcional del sistema con el núcleo SR propuesto frente a la referencia, se pasó a la creación de la descripción SystemC del nuevo sistema. Para validar la descripción SystemC se usaron los datos de referencia obtenidos durante la validación a nivel algorítmico. La validación se llevó a cabo siguiendo una configuración de validación genérica del HDL. De acuerdo a esta configuración, a nivel más alto de la organización, el sistema comprende dos módulos: *el módulo objeto de la verificación* DUT (*Desing Under Test*) y *el módulo del banco de pruebas TB* (*Test Bench*), que encapsula el resto del código utilizado en la validación. Es importante destacar que el banco de pruebas no tiene por qué ser sintetizable en ningún punto del desarrollo.

La depuración del diseño está basada en el almacenamiento y revisión de los resultados intermedios de procesamiento. Para ello, se han utilizado interfaces adicionales de salida, a través de las que se trasmiten al TB los datos recogidos en los llamados “puntos de va-

lidación". En la mayoría de los casos, los puntos de validación se han situado en lugares intermedios entre los distintos módulos, recolectando y traspasando datos de unos a otros. Estos puntos se han implementado a través de la replicación de canales de transferencia a las interfaces externas. En algunos casos, con el fin de seguir el progreso del procesamiento dentro de los módulos de computación y/o poder investigar los valores de variables y señales internas, los puntos de validación se instanciaron como una parte de estos módulos. El receptor de los datos recogidos en los puntos de verificación fue el hilo de monitorización implementado por el módulo del banco de pruebas. Una vez recibida, la información se comprobaba para determinar la correcta ejecución, proporcionando la información por consola y/o almacenando dicha información. La mayor parte del código del testbench ha sido trasladada desde el código C usado para la validación del modelo de algoritmo. Con el fin de disminuir los tiempos de simulación, el proceso de estimación de movimiento no se ha implementado como código, sino que se ha emulado mediante la carga de los híper-datos (métricas ME) a partir de los archivos obtenidos en las simulaciones del algoritmo a nivel funcional. La salida observada para el modelo de algoritmo de la implementación a nivel de MB se considera como la salida esperada y ha sido usada en la verificación de la implementación a nivel de MB de los modelos ESL y RTL. El resto de cambios han sido realizados con el fin de adaptarse al protocolo de comunicaciones a través de las interfaces a nivel de pin o TLM.

B.4.1.4 RTL

Una vez verificada, la descripción ESL se convirtió en sintetizable y se usó como entrada en la síntesis de alto nivel. La descripción del sistema generada es la descripción RTL en VHDL o EDIF. Con el fin de validar el HDL sintetizado, se llevaron a cabo distintas simulaciones. La configuración de la validación ESL presentada anteriormente fue reutilizada en las simulaciones RTL. Esto fue posible a través de una simulación de lenguaje mixto en la cual las unidades de diseño VHDL se instanciaron en el diseño en SystemC. La instanciación de los módulos HDL en el SystemC se llevó a cabo mediante unos "encapsulados" que comunican las interfaces RTL con las de SystemC. Una vez que la descripción HDL fue compilada y los encapsulados fueron introducidos en el sistema, se usó el entorno de verificación presentado en la Fig. B.8 para validar la exactitud de la descripción HDL incrustada en el código SystemC.

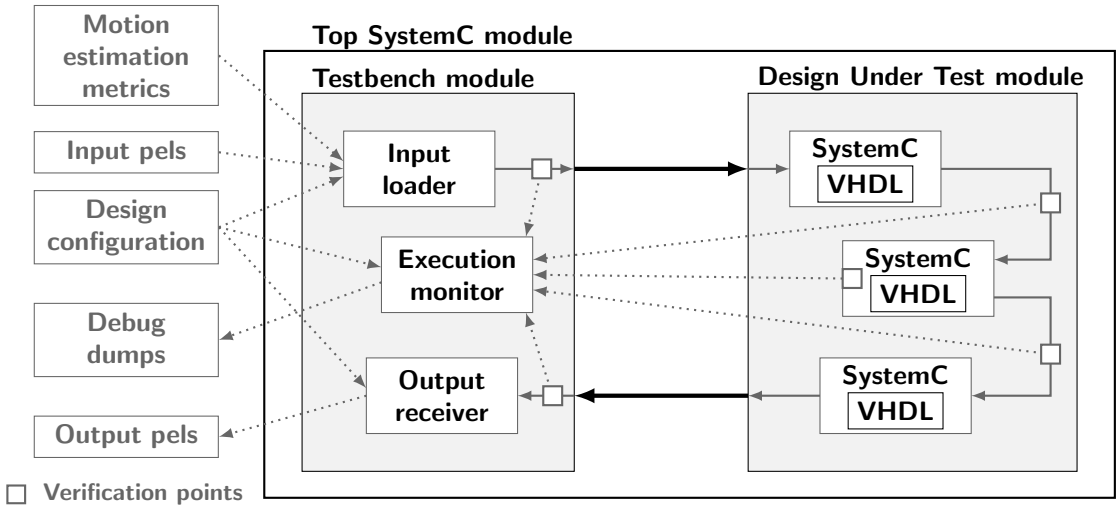


Fig. B.8 Entorno de verificación RTL basado en encapsulados SystemC de VHDL.

B.5 Implementación en FPGA

En este capítulo se explica en detalle la implementación hardware. Esta implementación se ha llevado a cabo a través de la metodología establecida en la sección anterior. De acuerdo a esta metodología, se han modelado distintos niveles de abstracción para alcanzar el nivel de descripción usado en la síntesis lógica. El resultado de la síntesis ha permitido determinar el cuello de botella del sistema y ha servido de guía para el proceso de refinamiento de la arquitectura y la organización de los flujos de procesamiento. El proceso para alcanzar prestaciones de tiempo real implica asumir distintos retos de implementación y múltiples iteraciones de refinamiento.

B.5.1 Visión general del sistema de súper-resolución

El sistema de súper-resolución basado en el algoritmo NUGP, en el nivel más alto de organización, comprende tres bloques: (i) la estimación de movimiento, (ii) el núcleo de súper-resolución SRK (*Super Resolution Kernel*), y (iii) el código que gestiona los interfaces de entra/salida. Como ya se ha mencionado, la implementación de la estimación de movimiento está fuera del objetivo de este trabajo. Para mejorar el rendimiento de la simulación, la estimación de movimiento se ha emulado mediante la introducción de código adicional que gestiona la carga de los resultados de estimación de movimiento a partir de archivos con los resultados de estimaciones de movimiento. Como ya se ha indicado, la implementación hardware del algoritmo de BM se ha desarrollado en paralelo [HN10]. El

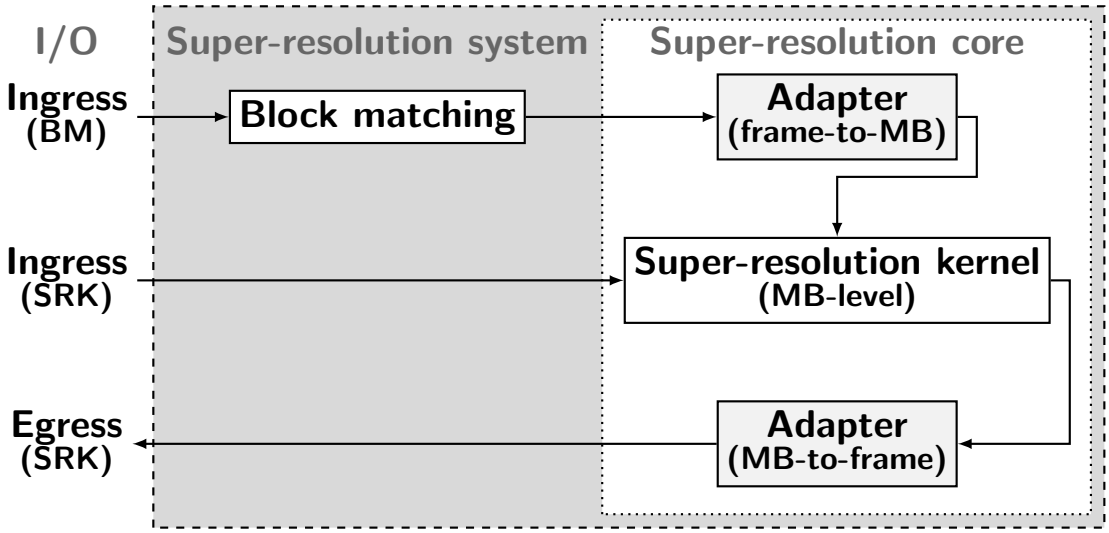


Fig. B.9 Visión general del sistema de súper-resolución con el núcleo de procesamiento propuesto.

módulo SRK incluye la lógica de gestión del NUGPA, esto es, dado el macro-bloque a procesar y los datos asociados (las métricas del ME, las áreas de búsqueda, etc.), genera la representación súper-resuelta del MB.

El código original del SRK se transformó con el fin de operar a nivel de MBs, dado que el algoritmo de estimación de movimiento original operaba a nivel de fotograma, es decir, computaba las métricas fotograma tras fotograma. Tras las modificaciones, el código del SRK se volvió incompatible con el resto del sistema. Para hacer frente a esta situación, se desarrollaron dos módulos de adaptación: un primer módulo para recibir y reordenar las salidas de la estimación de movimiento, poniéndolas en el orden adecuado para el procesamiento MB a MB, y un segundo módulo para reconstruir el fotograma súper-resuelto de los MBs generados por el SRK. La implementación de las modificaciones indicadas estructura el código de referencia de súper-resolución como se indica en la Fig. B.9. En referencia a la nomenclatura establecida, el núcleo de súper-resolución y el código de adaptación forman el denominado dispositivo bajo test, mientras que la gestión de las I/O con la emulación del código BM se ha encapsulado formando un prototipo del módulo de banco de pruebas.

In reference the established nomenclature, the super resolution kernel and the adapters code formed the design under test, and the I/O management code with BM emulation code was encapsulated forming a prototype of the testbench module.

B.5.2 Desafíos en la implementación

El objetivo de prestaciones para la ejecución ha sido establecido para que el sistema sea capaz de súper-resolver 24 fotogramas por segundo (fps), de tamaño QCIF y con formato YUV 4:2:0 progresivo, que se corresponde con el formato 24p [Ski05]. A la frecuencia objetivo ftargeted se le asignó el valor de 109 MHz. Este valor fue presentado en las publicaciones [TSC+09, TCH+09] para la implementación de un decodificador H.264 baseline, que siguió el flujo de diseño ESL establecido y que iba dirigido a la misma familia de dispositivos FPGA. El número máximo de ciclos $limit_{cycles}$ disponibles para un procesamiento de un MB 4x4 se estimó dividiendo la estimación de la frecuencia objetivo, por el número de MBs (MBsnr) que vayan a procesarse en un segundo, como se muestra en la ecuación (B.1). Basándose en las prestaciones requeridas de 1584 MBs 4x4 por segundo y una estimación de la frecuencia de operación de 109 MHz, el límite de ciclos se estimó en $limit_{cycles}$ ciclos. La idea de procesar un MB abarca los siguientes elementos: el procesamiento interno del módulo, la sincronización y la comunicación entre módulos.

$$limit_{cycles} = \left\lfloor \frac{f_{targeted}}{MBs_{nr} * fps} \right\rfloor = \left\lfloor \frac{109 * 10^6}{1584 * 24} \right\rfloor = 2867 \tag{B.1}$$

Con el fin de cumplir con los objetivos de ejecución establecidos, se han abordado una serie de desafíos que se presentan brevemente a continuación:

- **Implicaciones a nivel de sistema de los valores de los parámetros del SRK.** El valor de los parámetros del SRK tiene un impacto directo en el rendimiento y en la eficiencia de ejecución de la implementación del núcleo de súper-resolución, NUGPA. A fin de tener ese impacto en cuenta, el código del diseño se ha parametrizado usando las macros que representan valores determinables en tiempo de síntesis para la sincronización, la definición de la memoria, la definición de los iteradores de bucle y la replicación del módulo condicional.

- **Desafíos relacionados con la memoria.** El rendimiento alcanzado por una implementación en dispositivos FPGA de un algoritmo de procesamiento de imágenes por ordenador, es más probable que esté limitado por los recursos de memoria disponibles que por los recursos computacionales.

  - *Implementación de memoria usando recursos eficientes.* El uso de look-up tables y de memorias distribuidas para la implementación de las matrices es altamente ineficiente. Con el fin de forzar al sintetizador a que infiera correctamente

el uso de los dispositivos de bloques de RAM (BRAM) disponibles se han utilizado macros específicas del compilador para la replicación de memorias de acceso aleatorio.

  - *Comunicación entre módulos que permita la ejecución de módulos en paralelo.* Con el fin de aprovechar el paralelismo ofrecido por el hardware, el sistema tiene que descomponerse en módulos que puedan ejecutarse en paralelo, haciendo uso de un esquema de comunicación basado en memoria compartida con espacios de direcciones separados de escritura y lectura, intercambiados después de los eventos de sincronización.
  - *Garantía del rendimiento de memoria.* Con el fin de aumentar el rendimiento de la memoria, la implementación propuesta utiliza una mezcla de replicación de memoria y de accesos a memoria coalescentes. Se logran ganancias adicionales mediante el uso de memorias compartidas con ampliación de doble mapa de memoria e interfaces de comunicación y replicación de memoria.

- **Dependencia de datos en el rellenado de los huecos.** El proceso de interpolación requiere que las coordenadas de una vecindad de píxeles en forma de cuadrado, cuyo valor está siendo calculado, estén disponibles de forma inmediata para su acceso. En nuestra implementación, la carga de trabajo actual se determina en función de la distancia (en coordenadas de alta resolución) de los bordes de los MB a partir de los bordes del fotograma utilizando un conjunto de indicadores. Los píxels que no puedan ser procesados en el momento de la recepción debido a dependencias de datos se almacenan temporalmente hasta que las dependencias de datos se puedan resolver.

- **Tamaño variable de la zona de búsqueda.** Después de cambiar a procesamiento a nivel de MB, el buffer de fotograma se eliminó. Esto dio lugar a la necesidad de extracción de los píxeles del área de búsqueda a partir de fuentes diferentes. En nuestra implementación, los píxeles del área de búsqueda cuyas coordenadas son coordenadas válidas de fotograma se extraen fuera del núcleo súper-resolución y se realimentaron de nuevo como entradas. El SA se reconstruye internamente en base a un conjunto definido de indicadores que señalan la posición del SA dentro del fotograma.

- **Construcción y gestión de las cuadrículas.** Las cuadrículas de alta resolución del sistema se construyen a partir de una imagen de LR por medio de interpolación con el factor de escala igual a la precisión. En nuestra implementación, el problema con las cuadrículas de alta resolución del SA fue resuelto usando un esquema de direc-

cionamiento de memoria intermedia que, junto con las coordenadas de HR de la cuadrícula, fue capaz de proporcionar datos correctos operando en la representación de LR.

- **Determinación del tamaño de la ventana de fotogramas.** El concepto de ventana deslizable de fotogramas se utilizó con el fin de proporcionar capacidades de súper-resolución dinámica. El número exacto de fotogramas que se incluyen en la SFW varía a lo largo de la ejecución. En nuestra implementación, un número variable de fotogramas en la SFW se gestiona por medio de contadores internos y de señalizadores externos que señalan transiciones de estado de la SFW.

- **Operación de división.** La operación de división no es una construcción sintetizable por las herramientas de síntesis de alto nivel utilizadas. En nuestra implementación, esta operación fue emulada tanto por medio de otras operaciones (es decir, desplazamientos de bits (+ sumas) o multiplicaciones) como implementándola utilizando un módulo hardware personalizado.

B.5.3 Arquitectura del núcleo de súper-resolución: arquitectura implementada y organización

La arquitectura funcional del núcleo de súper-resolución que alcanza las prestaciones y la funcionalidad perseguidas se presenta en la Fig. B.10. Internamente, el núcleo mantiene la división en tres partes principales: (i) la gestión del buffer de reordenación, (ii) el núcleo de súper-resolución, y (iii) la reconstrucción del fotograma. Sin embargo, con el fin de cumplir con el número de ciclos de ejecución establecido, muchos de los bloques funcionales tuvieron que ser descompuestos en módulos más pequeños. La tarea de gestión de reordenación del búfer (*Reorder Buffer management*) tuvo que ser dividida en dos etapas, encapsuladas como dos módulos: el *ReorderBufferReap* y el *ReorderBufferSew*. El primero de los módulos opera a nivel de fotograma, almacenando el resultado del ME en la memoria siguiendo un orden adecuado para procesar a nivel de MB. Por lo tanto, el módulo *ReorderBufferReap* controla la recepción de las métricas de estimación de movimiento, el reordenamiento de datos y el acceso de almacenamiento al buffer de reordenamiento. Sólo una vez que se han recibido todos los datos de los fotogramas de referencia del SFW actual, el *ReorderBufferSew* puede procesarlos. El *ReorderBufferSew* gestiona las solicitudes de datos que surgen dentro del núcleo de súper-resolución, lleva a cabo la carga de datos del buffer de reordenamiento y proporciona datos leídos al *ShadPrep* del SRK.

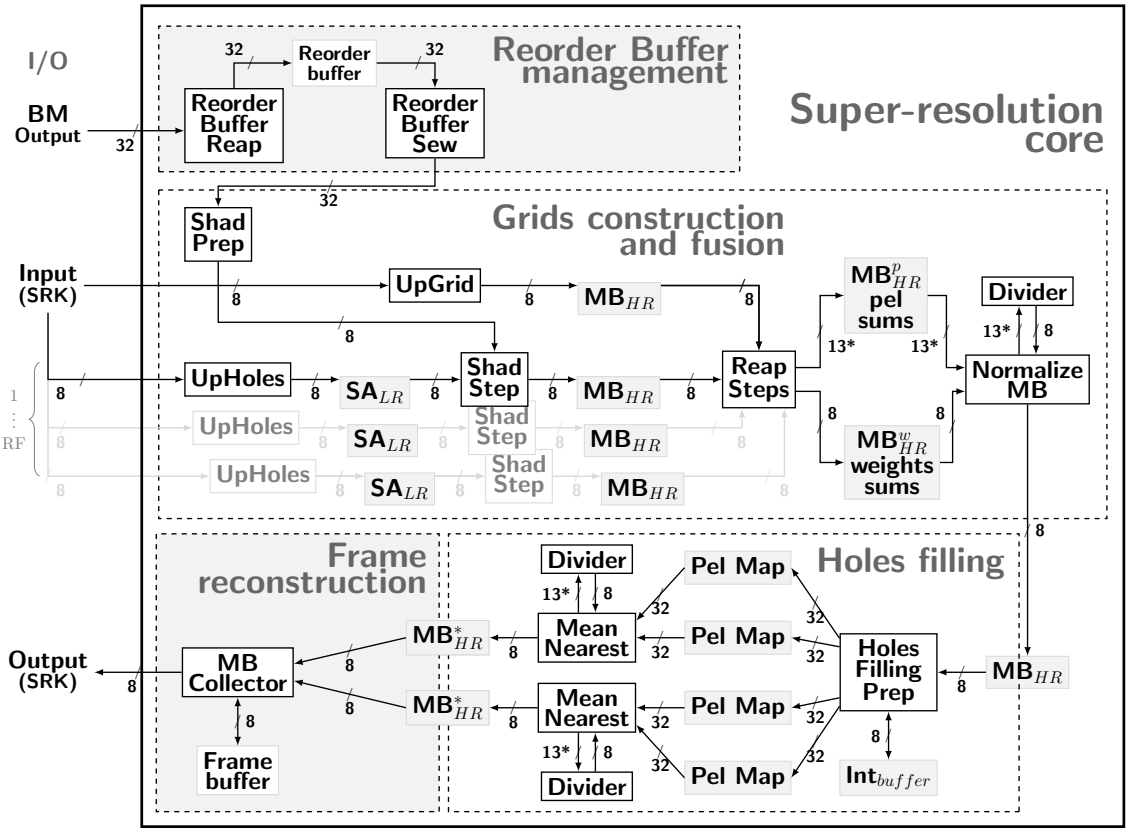


Fig. B.10 Visión general de la organización del modelo pin-accurate.

El núcleo de súper-resolución lleva a cabo dos tareas principales: (i) construcción de la cuadrícula y fusión, y (ii) interpolación. De este modo, los módulos que componen el SRK pueden verse como pertenecientes a uno de los dos grupos, dependiendo de sus tareas. Los módulos *UpGrid*, *ShadPrep*, *UpHoles* y *ShadStep*, llevan a cabo la construcción de las cuadrículas. La fusión de las cuadrículas es realizada por los módulos *ReapSteps* y *NormalizeMB*. El proceso de rellenado de huecos se lleva a cabo por los módulos *HolesFillingPrep* y *MeanNearest*.

La entrada al SRK comprende los píxeles LR del macro-bloque y de sus áreas de búsqueda (uno por cada fotograma en el SFW), y un conjunto de indicadores (*flags*). Estos indicadores describen el macro-bloque y el área de búsqueda dentro del fotograma e identifican las dependencias de datos entre macro-bloques. El módulo *ShadPrep* recibe los indicadores y solicita resultados de ME (MVs y SADs) del módulo *ReorderBufferSew*. Una vez que estos parámetros están disponibles, se calcula la localización actual del macro-bloque dentro de la zona de búsqueda y los pesos para cada área de búsqueda de los actuales SFW.

Las coordenadas de localización calculadas y los pesos, junto con los MVs y los indicadores, se transmiten a los módulos *ShadStep*.

Los píxeles del macro-bloque de LR se proporcionan desde la entrada principal de súper-resolución al módulo *UpGrid*. Cada píxel recibido se somete a la transformación *zeroes2ones* y se almacena en la memoria de entrada del módulo *ReapSteps*. Como resultado, la memoria de entrada de ReapSteps mantiene la representación de HR (interpolados 4x) de los MBs de LR. Después de haber terminado su tarea, el módulo *UpGrid* envía una señal de sincronización al módulo *ReapSteps*.

Los píxeles del área de búsqueda son recibidos por los módulos *UpHoles*. La tarea de cada uno de los módulos de *UpHoles* es almacenar los píxeles recibidos en la memoria de entrada del módulo *ShadStep* correspondiente. Los módulos ShadSteps examinan los indicadores, identifican (en base a los MVs y a las coordenadas recibidas del *ShadPrep*) y extraen los píxeles que se utilizan para la creación de la cuadrícula del SA. La cuadrícula del SA construida se almacena en la memoria de entrada del módulo *ReapSteps*. Después de haber terminado su tarea, cada *ShadStep* envía una señal de sincronización al módulo *ReapSteps*

Habiendo recibido las señales de sincronización de los *ShadSteps* y el módulo *UpGrid*, el *ReapSteps* inicia su funcionamiento. La tarea del módulo *ReapSteps* consiste en la carga de datos de las cuadrículas, la fusión de las mismas, y el almacenamiento del resultado en la memoria de entrada del módulo *NormalizeMB*. En primer lugar, recibe los indicadores y los pesos de uno de los módulos *ShadSteps*. Luego, para cada coordenada de HR se cargan píxeles de las memorias de entrada (la cuadrícula de HR de MB y la cuadrícula del SA) y las fusiona. Los resultados de la operación de fusión para cada coordenada de HR son los valores de la suma ponderada de los píxeles con valor no-nulo cargados desde las memorias de entrada, y la suma de los pesos de los píxeles no-nulos que contribuyeron a la operación de fusión. Dichos valores se almacenan en la memoria de entrada del módulo *NormalizeMB*. Para cada coordenada de HR, el módulo *NormalizeMB* carga los dos valores producidos por los módulos *ReapSteps*. Entonces, se divide la suma ponderada por la suma de los pesos y se almacena el resultado en la memoria de entrada del módulo *HolesFillingPrep*.

Los módulos *MeanNearest* y *HolesFillingPrep* llevan a cabo el proceso de rellenado de huecos. Estos módulos calculan los valores de píxel para las coordenadas de HR de la cuadrícula, cuyos valores no fueron estimados durante el proceso de fusión. El resultado producido por las partes de construcción de la cuadrícula y fusión del SRK se transmite al módulo *HolesFillingPrep*. Los datos se utilizan para construir una estructura llamada el mapa de píxeles. Esta estructura se compone de píxeles del macro-bloque que se está

procesando actualmente y de los MBs procesados previamente. Los elementos de la imagen que se utilizan para ver el mapa de píxeles de la construcción se determinan en función de la ubicación del macro-bloque dentro del fotograma. Las dependencias de datos entre macro-bloques se gestionan mediante la exclusión del procesamiento de algunas partes del mapa de píxeles. Los píxeles que se van a procesar por el módulo *MeanNearest* forman una región continua llamada la región de trabajo. La región de trabajo se identifica por sus coordenadas ubicadas en la esquina superior izquierda e inferior derecha. Sólo los píxeles pertenecientes a la región de trabajo son procesados por los módulos *MeanNearest*. Los píxeles con dependencias entre macro-bloque no resueltas son excluidos de la región de trabajo. Una copia de estos píxeles se almacena en las memorias locales hasta que se resuelven las dependencias de datos.

Una vez creado, el mapa de píxeles se almacena en la memoria de entrada del módulo *MeanNearest*. Sobre la base de las coordenadas recibidas, cada módulo *MeanNearest* determina su región de trabajo y la procesa píxel a píxel. Para cada píxel cargado se calculan las coordenadas en las que se almacenará en la memoria de entrada del siguiente módulo. Si el valor del píxel es diferente de cero, se almacena en la memoria de entrada del módulo *MBcollector*, y se carga el siguiente píxel. En caso contrario, antes de almacenar un nuevo píxel, el valor de píxel se calcula usando una interpolación del vecino más próximo (*mean nearest neighbors interpolation*).

El módulo *MBcollector* recibe los píxeles súper-resueltos y lleva a cabo la reconstrucción del fotograma. Debido a las dependencias entre macro-bloques, el número de píxeles súper-resueltos en las memorias de entrada del módulo *MBcollector* es variable. El número exacto de píxeles súper-resueltos que se tienen que extraer, su ubicación en la memoria de entrada y la memoria buffer del fotograma son determinados en base a los indicadores recibidos y del valor del parámetro del factor de escalado. Habiendo recibido y procesado todos los MBs que pertenecen a un fotograma, el módulo envía el contenido de las memorias de fotograma a través del puerto de píxeles súper-resueltos.

B.5.4 Resultados

La Tabla B.3 resume los resultados de la implementación hardware, y muestra una comparación con la implementación del estado de la técnica presentada en [BB08]. El sistema ha sido implementado en un dispositivo FPGA avanzado[Xil09]. La implementación resultante (etiquetada como $Tech_1$ en la Tabla B.3). para tamaño de MB de 4x4 píxeles, factor de escala igual a 2, y SFW con 3 fotogramas (RF = 2) ocupó 10291 LUTs, 16 bloques DSP,

TABLE B.3 *Resultados de las prestaciones hardware para dos dispositivos FPGA en comparación con [BB08] para el caso particular de RF=2, factor de escala=2, SA=2, y MB=4 MB. N/A significa no disponible.*

	SR_{iuma}		Referencia [BB08]	
	Dispositivo		Etapas iterativas [Xil07]	
	$Tech_1$ [Xil09]	$Tech_2$ [Xil07]	10	20
Lógica [LUT]	10291	13031	35707	68317
BRAM	66	132	134	234
Bloques DSP	16	2	N/A	N/A
Frecuencia [MHz]	109	68	58	58
Tasa de fotogramas [fps]	25	16	61	61

y 66 bloques RAM. Esto corresponde a una ocupación de los recursos del dispositivo del 22%, 12% y 44,6%, respectivamente. El uso total de memoria fue de 2376 MB (44%). No se requirió del uso de memoria externa al chip. La frecuencia final del reloj fue de 109 MHz, dando como resultado la posibilidad de aplicar súper-resolución con factor de escala 2 a 25 fotogramas QCIF por segundo.

Esto es significativamente menor que la aplicación dada en [BB08], que ofrece una velocidad de fotogramas de 61 fps y una mayor resolución de salida. Sin embargo, la implementación alternativa sólo cuenta con 10 etapas iterativas, es decir, la mitad de las 20 etapas asignadas por los autores como el umbral para producir resultados de calidad satisfactoria. Por lo tanto, con el fin de ofrecer una comparación más equitativa de los recursos usados por el estado del arte, también se presenta una aproximación de las prestaciones alcanzadas por una solución con 20 etapas iterativas. Los requisitos de lógica y memoria para esta configuración se estimaron mediante la suma de los costes de ejecución de 10 etapas de iteración adicionales a el coste implementación del sistema con 10 etapas más de iteración. Por otro lado, la aplicación que se presenta en este trabajo también fue mapeada y ruteada para el dispositivo [Xil07] usado en [BB08]. Los resultados de esta implementación se etiquetan como $Tech_2$ en la Tabla B.3, y pueden ser utilizados para una comparación directa.

La implementación que se presenta en este trabajo ha sido optimizada para conseguir la calidad del software de base y reducir al mínimo los requisitos hardware. Con el fin de medir la eficacia de las implementaciones FPGA, se propone utilizar como figura de mérito el uso de look-up tables (*LUT*) frente a las mejoras de SR ($boostCost_{SR}$). Esta figura de mérito representa el número de LUTs requeridas por la aplicación y necesarias para obtener un incremento del 1% en calidad promedio (objetiva) medida. El objetivo es mantener esta figura de mérito lo más baja posible. El boostCostSR se estima tal y como se muestra en

TABLE B.4 *Figura de mérito $boostCost_{SR}$ frente al coste de interpolación en términos de LUTs para el caso particular de RF=2, factor de escala=2, SA=2, y MB=4).*

Implementación	$boostCost_{SR}\left[\frac{LUT}{\%}\right]$		
versus nearest-neighbour interpolation			
$SR_{iuma}\ Tech_2$	**1269**	**680**	**894**
IBP con pesos [BB08]	16275	6546	5832
versus bi-cubic interpolation			
$SR_{iuma}\ Tech_2$	**2753**	**1266**	**1458**
IBP con pesos [BB08]	No gain	30804	10994
Secuencia	paris	mobile	foreman

la ecuación (B.2), donde LUT_{nr} representa el número de LUTs necesarios para la implementación y los valores $PSNRSR_i$ y $PSNRint_i$ representan, respectivamente, el valor PSNR del fotograma súper-resuelto e interpolado para el fotograma i de la secuencia de prueba, que comprende un conjunto de n fotogramas.

$$boostCost_{SR} = \left(\frac{n \times LUT_{nr}}{100}\right) \Bigg/ \left(\sum_{i=1}^{n} \frac{PSNR_{SR_i} - PSNR_{int_i}}{PSNR_{int_i}}\right) \tag{B.2}$$

El $boostCost_{SR}$ se calculó para la implementación propuesta, basándose en los resultados de la implementación presentados en esta sección para $Tech_2$ y en el valor medio PSNR para los 90 fotogramas iniciales de las secuencias de prueba paris, foreman y mobile (componente de luminancia) presentados en la Tabla B.2. Los valores de IBP fueron amablemente proporcionados por los autores de [BB08]. Otros valores se obtuvieron utilizando la configuración utilizada para la evaluación del software de referencia. Los valores resultantes de la figura boostCostSR para las secuencias paris, foreman y mobile se presentan en la Tabla B.4.

El estudio muestra que, para las secuencias de prueba utilizadas, la figura $boostCost_{SR}$ para la implementación propuesta es al menos 6,5 veces inferior a la de la implementación presentada en [BB08]. Las principales razones de una diferencia tan grande en la $boostCost_{SR}$ radican en el uso del block-matching (BM) ME, y en la naturaleza no iterativa del algoritmo implementado. La primera conlleva un mayor PSNR, mientras que la segunda permite reducir la ocupación total del dispositivo usado. El impacto que tiene el algoritmo de ME en los resultados se manifiesta de una manera clara en resultados para secuencias con abun-

dancia de movimientos locales como foreman y mobile. Estos movimientos no pueden ser trazados con exactitud usando un ME a nivel de fotograma. Para estas secuencias el PSNR obtenido usando una estimación a nivel de fotogramas es significativamente más bajo, apuntando a mayores valores de $boostCost_{SR}$.

B.6 Conclusiones

El principal objetivo de esta tesis doctoral ha sido demostrar la capacidad de ejecución en tiempo real del algoritmo NUGP por medio de una implementación hardware que usa solo recursos de memoria internas al dispositivo. Para alcanzar este objetivo, primero, se han introducido modificaciones a nivel de algoritmo en el flujo de ejecución del algoritmo de referencia para reducir sus elevados requisitos de ocupación de memoria, lo que descartaba una implementación hardware de alto rendimiento. Se han usado modelos teóricos que desarrollan el flujo propuesto y el flujo de referencia para identificar los cuellos de botella y evaluar cuantitativamente las ventajas y desventajas asociadas con el cambio en el flujo propuesto de granularidad más fina. Los resultados de las modificaciones propuestas muestran una reducción significativa de la ocupación de memoria a cambio de incrementar el tráfico de memoria. Los valores esperados máximos y mínimos que se han calculado del factor de reducción de la ocupación de la memoria asociada con el cambio al nuevo flujo propuesto estaban entre el rango de 3,5 y 16, dependiendo de los valores de los parámetros SRK. Los valores calculados máximos y mínimos del factor del incremento de tráfico en memoria asociados al cambio realizado estaban en el rango de 1,1 y 16,9 respectivamente. El valor exacto de estos factores depende de la combinación del valor de los parámetros usados para configurar el núcleo de SR. Para 87 de las 96 configuraciones investigadas (más del 90%) el factor de reducción de memoria ocupada ha sido mayor que el factor del incremento del tráfico de la memoria, demostrando la efectividad de la estrategia propuesta.

Para nuestra implementación hardware hemos establecido una metodología que tiene en cuenta la naturaleza de constante evolución del algoritmo y su implementación software, al tiempo que usa una jerarquía escalonada de abstracciones a partir de las cuales el nivel de descripción HDL *pin-accurate* es obtenido de una manera automatizada. Por medio del uso de SystemC como lenguaje de codificación de los modelos intermedios hemos sido capaces de reutilizar la mayoría del código C. Esto facilita la propagación rápida de las modificaciones hechas a nivel funcional, a través de modelos intermedios, hasta la descripción HDL final. El uso de representaciones intermedias incrementa la portabilidad del diseño, permitiendo una síntesis de alto nivel para conseguir un rango de lenguajes HDL (VHDL, Verilog,

etc.) a partir de la misma descripción de alto nivel. Además, usando lenguajes basado en C se deja abierta la posibilidad a través de toda la abstracción jerárquica de una rápida migración de los modelos a otros lenguajes basados en C, en particular al estándar OpenCL. El nivel de detalles de la descripción metodológica proporcionada facilita la reutilización del flujo establecido en las implementaciones de otros algoritmos similares.

Usando la metodología establecida, la arquitectura propuesta ha sido mapeada satisfactoriamente usando una tecnología FPGA Xilinx Virtex 5. La arquitectura final ha conseguido un rendimiento de 24 fotogramas por segundo con una frecuencia de operación de 109 MHz usando el dispositivo xc5vf70t-1 *xc5vf70t-1*. La comparación con el estado del arte llevada a cabo muestra resultados satisfactorios dado que la ocupación de recursos lógicos observados por el sistema propuesto fue hasta 5 veces menor que el reportado en el estado del arte, mapeado usando la misma tecnología FPGA [BB08]. Los resultados obtenidos de la síntesis para la implementación hardware: (i) han demostrado la capacidad de alcanzar un rendimiento en tiempo real en tecnología FPGA, mientras se mantiene la calidad de las imágenes súper-resueltas al mismo nivel que el ofrecido por las implementaciones software, y (ii) ha proporcionado la corrección de los cambios de nivel de los algoritmos presentados y la metodología propuesta. La portabilidad del diseño se ha preservada gracias al uso de gran cantidad de niveles del código de abstracción en el SystemC genérico, el cual permite una descripción de alto nivel para ser sintetizada por un amplio rango de lenguajes (VHDL, Verliog, etc.) y tecnologías diferentes (Altera, Actel, Lattice, Vantis, Lucent, etc.) sin (o con) una optimización específica del proveedor. La portabilidad del diseño es además reforzada por no usar (explícitamente) ningún acelerador específico (MAC, sumadores, divisores, etc.).

En resumen, en este trabajo ha llevado a cabo satisfactoriamente la transferencia de conocimiento del dominio de los algoritmos de compresión al dominio de los algoritmos de mejora de imagen usando SR, evaluado cuantitativamente el impacto de las modificaciones, proporcionado una metodología de alto nivel de las implementación que facilita una propagación rápida de las modificaciones y proporcionando una implementación hardware eficiente en FPGAs que permite el procesamiento en tiempo real mientras mantiene la calidad de las imágenes súper-resueltas producidas.

B.7 Líneas futuras de investigación

Los logros alcanzados en este trabajo abren el camino para nuevas líneas de investigación focalizadas en indagar más en el estado del arte de la implementación de SR en hardware:

- La implementación propuesta podría ser analizada y modificada para lograr mayores mejoras, en particular: (i) Implementación de las mejoras a nivel de algoritmo propuestas por E. Quevedo en [Gut15]. (ii) Extensión del rango de los valores de los parámetros SRK soportados y exploración de las posibilidades para soportar efectivamente tamaños de macro-bloque variables. (iii) Mapeado en plataformas heterogéneas y encapsulado en una IP.

- Implementación del sistema con escalado basado en replicación. El trabajo de G. Singla [STdA13] presenta una aproximación que usa múltiples núcleos del NUGPA, aplicando la SR en paralelo en bloques exclusivos del fotograma de entrada para aumentar la velocidad de procesado. La aproximación presentada está limitada por la latencia de los accesos a memoria. Como la aproximación presentada solo usa memoria interna, su extensión a este tipo de configuraciones podría aliviar los problemas asociados de memoria logrando un aumento significativo de la velocidad.

- Las plataformas multi-core y many-core suponen una tendencia emergente indudable en el ambiente de cómputo de alto rendimiento (High Performance Computing, HPC). La aparición de soluciones híbridas que usan estas plataformas en conjunto con FPGAs [PCC+14, WHK+14, MJK12], sugieren explorar el espacio de diseño para el mapeado de SR en esas plataformas.

- Finalmente, sería interesante usar la metodología propuesta para implementar otros algoritmos de procesado de imagen de alta complejidad, usando diferentes herramientas en la cadena de diseño.

References

[ABCC08] Maria E. Angelopoulou, Christos-Savvas Bouganis, Peter Y. K. Cheung, and George A. Constantinides. FPGA-based real-time super-resolution on an adaptive image sensor. In *Reconfigurable Computing: Architectures, Tools and Applications*, volume 4943 of *Lecture Notes in Computer Science*, pages 124–135. Springer, March 2008.

[ABCC09] Maria E. Angelopoulou, Christos-Savvas Bouganis, Peter Y. K. Cheung, and George A. Constantinides. Robust real-time super-resolution on FPGA and an application to video enhancement. *ACM Trans. Reconfigurable Technol. Syst.*, 2(4):1–29, 2009.

[Acc08] Accellera. SystemC source release. Online, 2008. http://systemc.org/downloads/standards.

[AEH+14] H. Ashikaga, H.L. Estner, D.A. Herzka, E.R. Mcveigh, and H.R. Halperin. Quantitative assessment of single-image super-resolution in myocardial scar imaging. *Translational Engineering in Health and Medicine, IEEE Journal of*, 2:1–12, 2014.

[Age08] European Space Agency. System-level modeling in SystemC. Online, 2008. http://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/System-Level_Modeling_in_SystemC.

[AIAOM13] K. Al Ismaeil, D. Aouada, B. Ottersten, and B. Mirbach. Multi-frame super-resolution by enhanced shift & add. In *Image and Signal Processing and Analysis (ISPA), 2013 8th International Symposium on*, pages 171–176, September 2013.

[Alt15] Altera. Implementing FPGA design with the opencl standard. Online, Nov 2015.

[AMK03] Luis D. Alvarez, Rafael Molina, and Aggelos K. Katsaggelos. Multi-channel reconstruction of video sequences from low-resolution and compressed observations. In *Progress in Pattern Recognition, Speech and Image Analysis*, volume 2905 of *Lecture Notes in Computer Science*, pages 46–53. Springer Berlin Heidelberg, 2003.

[App14] Apple. Patent application publication us2014/0125825: Super-resolution based on optical image stabilization. Online, 2014.

[ATB12] Le An, N. Thakoor, and B. Bhanu. Vehicle logo super-resolution by canonical correlation analysis. In *19th IEEE International Conference on Image Processing (ICIP)*, pages 2229–2232, September 2012.

[BAA05] D. Barreto, L. D. Alvarez, and J. Abad. Motion estimation techniques in super-resolution image reconstruction. A performance evaluation. In *Virtual Observatory: Plate Content Digitization, Archive Mining Image Sequence Processing*, pages 254–268. Heron Press, 2005.

[BAHH92] James R. Bergen, P. Anandan, Keith J. Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *The Second European Conference on Computer Vision, Proc. of*, European Conference on Computer Vision (ECCV) '92, pages 237–252, 1992.

[BB95] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Comput. Surv.*, 27(3):433–466, September 1995.

[BB08] O. Bowen and C.-S. Bouganis. Real-time image super resolution using an FPGA. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 89–94, September 2008.

[BB09] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing: Core Algorithms (Undergraduate Topics in Computer Science)*. Springer, 2009 edition, March 2009.

[BBM03] Christopher M. Bishop, Andrew Blake, and Bhaskara Marthi. Super-resolution enhancement of video. In *Artificial Intelligence and Statistics, In Proc. of*, 2003.

[BBV10] S.V. Basavaraja, A.S. Bopardikar, and S. Velusamy. Detail warping based video super-resolution using image guides. In *17th IEEE International Conference on Image Processing (ICIP)*, pages 2009–2012, September 2010.

[BEZN04] M. Ben-Ezra, A. Zomet, and S.K. Nayar. Jitter camera: high resolution video from a low resolution detector. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proc. of*, volume 2, pages II–135–II–142, June 2004.

[BEZN05] M. Ben-Ezra, A. Zomet, and S.K. Nayar. Video super-resolution using controlled subpixel detector shifts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):977–987, June 2005.

[BK00a] S. Baker and T. Kanade. Hallucinating faces. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 83–88, 2000.

[BK00b] S. Baker and T. Kanade. Limits on super-resolution and how to break them. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 372–379, 2000.

[BM10] Brian Bailey and Grant Martin. *ESL Models and their Application: Electronic System Level Design and Verification in Practice (Embedded Systems)*. Springer, December 2010.

[BM11] R. Sudheer Babu and K. E. Sreenivasa Murthy. A survey on the methods of super-resolution image reconstruction. *International Journal of Computer Applications*, 15(2):1–6, February 2011.

[Boc09] A. Bock. *Video Compression Systems: From First Principles to Concatenated Codecs (Iet Telecommuncations)*. The Institution of Engineering and Technology, July 2009.

[Bro92] Lisa Gottesfeld Brown. A survey of image registration techniques. *ACM Comput. Surv.*, 24(4):325–376, December 1992.

[BS72] Daniel I. Barnea and H.F. Silverman. A class of algorithms for fast digital image registration. *Computers, IEEE Transactions on*, C-21(2):179–186, February 1972.

[BS99] S. Borman and R.L. Stevenson. Simultaneous multi-frame MAP super-resolution video enhancement using spatio-temporal priors. In *IEEE International Conference on Image Processing (ICIP), Proceedings of*, volume 3, pages 469–473, 1999.

[BS00] S. Bhattacharjee and Malur K. Sundareshan. Modeling and extrapolation of prior scene information for set theoretic restoration and super-resolution of diffraction-limited images. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 2, pages 347–350, September 2000.

[Cad10] Cadence. *Cadence C-to-Silicon Compiler User Guide*. Cadence, September 2010.

[CB06] G.H. Costa and J.C.M. Bermudez. Statistical analysis of the lms algorithm applied to super-resolution video reconstruction. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 3, page III, May 2006.

[CB07] G.H. Costa and J.C.M. Bermudez. Statistical analysis of the lms algorithm applied to super-resolution image reconstruction. *Signal Processing, IEEE Transactions on*, 55(5):2084–2095, May 2007.

[CB08] G.H. Costa and J.C.M. Bermudez. Informed choice of the lms parameters in super-resolution video reconstruction applications. *Signal Processing, IEEE Transactions on*, 56(2):555–564, February 2008.

[CCL10] Ming-Hui Cheng, Hsuan-Ying Chen, and Jin-Jang Leou. Video super-resolution reconstruction using a mobile search strategy and adaptive patch size. In *Signal Processing and Multimedia Applications (SIGMAP), Proceedings of the 2010 International Conference on*, pages 108–111, July 2010.

[CD00] B. Cohen and I. Dinstein. Polyphase back-projection filtering for image resolution enhancement. *Vision, Image and Signal Processing, IEE Proceedings of*, 147(4):318–322, August 2000.

[Cel06] Celoxica. *Agility Compiler manual*, 2006.

[CEN+13a] Pedro P. Carballo, Omar Espino, Romén Neris, Pedro Hernández-Fernández, Tomasz M. Szydzik, and Antonio Núñez. Implementation of scalable video coding deblocking filter from high-level SystemC description. In *Proc. SPIE*, volume 8764, pages 876408–1–876408–10, 2013.

[CEN+13b] P.P. Carballo, O. Espino, R. Neris, P. Hernandez-Fernandez, T.M. Szydzik, and A. Nunez. Scalable video coding deblocking filter FPGA and asic implementation using high-level synthesis methodology. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 415–422, September 2013.

[CG03] Lukai Cai and Daniel Gajski. Transaction level modeling: an overview. In *The 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS), Proc. of*, pages 19–24, 2003.

[CJ06] Woong Il Choi and Byeungwoo Jeon. Hierarchical motion search for h.264 variable-block-size motion compensation. *Optical Engineering*, 45(1):017002–017002–9, 2006.

[CKK+96] Peter Cheeseman, Bob Kanefsky, Richard Kraft, John Stutz, and Robin Hanson. Super-resolved surface reconstruction from multiple images. In *Maximum Entropy and Bayesian Methods*, volume 62 of *Fundamental Theories of Physics*, pages 293–308. Springer Netherlands, 1996.

[CLL+06] Gustavo M. Callico, Rafael Peset Llopis, Sebastian Lopez, Jose Fco. Lopez, Antonio Nunez, Ramanathan Sethuraman, and Roberto Sarmiento. Low-cost super-resolution algorithms implementation over a hw/sw video compression platform, user's guide and reference manual. *EURASIP Journal on Applied Signal Processing*, 2006:1–29, 2006.

[CLS+08] G. Callico, S. Lopez, O. Sosa, J.F. Lopez, and R. Sarmiento. Analysis of fast block matching motion estimation algorithms for video super-resolution systems. *Consumer Electronics, IEEE Transactions on*, 54(3):1430–1438, August 2008.

[CLT+08] G.M. Callico, S. Lopez, K. Tarajano, J. Lopez, and R. Sarmiento. Impact of fast motion estimation algorithms on super-resolved video sequences. *Consumer Electronics, 2008. ICCE 2008. Digest of Technical Papers. International Conference on*, pages P3–19, January 2008.

[CMMC08] Jérôme Cornet, Florence Maraninchi, and Laurent Maillet-Contoz. A method for the efficient development of timed and untimed transaction-level models of systems-on-chip. In *The conference on Design, Automation and Test in Europe (DATE), Proc. of*, pages 9–14, 2008.

[CN07] Gustavo M. Callico and Antonio Nunez. Mobile receiver design increases the resolution of compressed video. In *SPIE newsroom Electronic Imaging & Signal Processing*. SPIE, 2007.

[CNYA12] Jin Chen, J. Nunez-Yanez, and A. Achim. Video super-resolution using generalized gaussian markov random fields. *Signal Processing Letters, IEEE*, 19(2):63–66, February 2012.

[CTH03] G. Caner, A.M. Tekalp, and W. Heinzelman. Super resolution recovery for multi-camera surveillance imaging. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, volume 1, pages I–109–12, July 2003.

[CY13] Yuzhang Chen and Kecheng Yang. MAP-regularized robust reconstruction for underwater imaging detection. *Optik - International Journal for Light and Electron Optics*, 124(20):4514–4518, 2013.

[CYX+11] Yuzhang Chen, Bofei Yang, Min Xia, Wei Li, Kecheng Yang, and Hiaohui Zhang. Model-based super-resolution reconstruction techniques for underwater imaging. In *Photonics and Optoelectronics Meetings (POEM) 2011: Optoelectronic Sensing and Imaging, Proc. of SPIE*, volume 8332, pages 83320G–83320G–10, November 2011.

[CZ01] D. Capel and A. Zisserman. Super-resolution from multiple views using learnt image models. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proc. of*, volume 2, pages II–627–II–634, 2001.

[DBS06] Jean-Pierre Deschamps, Gery Jean Antoine Bioul, and Gustavo D. Sutter. *Synthesis of arithmetic circuits : FPGA, ASIC and embedded systems*. John Wiley & Sons, Inc., Hoboken, New Jersey, March 2006.

[DH97] Frank H. Durgin and Alexander C. Huk. Texture density aftereffects in the perception of artificial and natural textures. *Vision Research*, 37(23):3273 – 3282, 1997.

[DHWG07] Shengyang Dai, Mei Han, Ying Wu, and Yihong Gong. Bilateral back-projection for single image super resolution. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1039–1042, July 2007.

[DKA04] G. Dedeoglu, T. Kanade, and J. August. High-zoom video hallucination by exploiting spatio-temporal regularities. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proc. ofn*, volume 2, pages II–151–II–158, June 2004.

[Dur96] FrankH. Durgin. Visual aftereffect of texture density contingent on color of frame. *Perception & Psychophysics*, 58(2):207–223, 1996.

[Eco12] G. Economakos. ESL as a gateway from OpenCL to FPGAs: Basic ideas and methodology evaluation. In *Informatics (PCI), 2012 16th Panhellenic Conference on*, pages 80–85, October 2012.

[EF95] A.M. Eskicioglu and P.S. Fisher. Image quality measures and their performance. *Communications, IEEE Transactions on*, 43(12):2959–2965, December 1995.

[EF97] M. Elad and A. Feuer. Restoration of a single superresolution image from several blurred, noisy, and undersampled measured images. *Image Processing, IEEE Transactions on*, 6(12):1646–1658, December 1997.

[EF99a] M. Elad and A. Feuer. Super-resolution reconstruction of continuous image sequences. In *IEEE International Conference on Image Processing (ICIP), Proceedings of*, volume 3, pages 459–463, 1999.

[EF99b] M. Elad and A. Feuer. Super-resolution reconstruction of image sequences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(9):817–834, September 1999.

[EF99c] M. Elad and A. Feuer. Superresolution restoration of an image sequence: adaptive filtering approach. *Image Processing, IEEE Transactions on*, 8(3):387–395, March 1999.

[EHO01] M. Elad and Y. Hel-Or. A fast super-resolution reconstruction algorithm for pure translational motion and common space-invariant blur. *Image Processing, IEEE Transactions on*, 10(8):1187–1193, August 2001.

[ESHEK12] Fathi E. Abd El-Samie, Mohiy M. Hadhoud, and Said E. El-Khamy. *Image Super-Resolution and Applications*. CRC Press, December 2012.

[Fat07] Raanan Fattal. Image upsampling via imposed edge statistics. *ACM Trans. Graph.*, 26(3), July 2007.

[FD87] DavidJ. Finlay and PeterC. Dodwell. Speed of apparent motion and the wagon-wheel effect. *Perception & Psychophysics*, 41(1):29–34, 1987.

[FDC84] David Finlay, Peter Dodwell, and Terry Caelli. The waggon-wheel effect. *Perception*, 13(3):237, 1984.

[FEM06] Sina Farsiu, Michael Elad, and Peyman Milanfar. Video-to-video dynamic super-resolution for grayscale and color sequences. *EURASIP J. Appl. Signal Process.*, 2006:232–232, January 2006.

[FM06] Michael Farsiu, S.and Elad and Peyman Milanfar. A practical approach to super-resolution. In *Visual Communications and Image Processing, Proc. of SPIE*, pages 24–38, 2006.

[Fos10] Luca Fossati. TLM 2.0 standard into action: Designing efficient processor simulators, 2010.

[Fos11] Luca Fossati. Electronic system level design at esa a hardware perspective. Online, September 2011.

[FREM03] S. Farsiu, D. Robinson, M. Elad, and P Milanfar. Robust shift and add approach to super-resolution. In *Conference on Applications of Digital Signal and Image Processing, Proceedings of SPIE*, pages 121–130, 2003.

[FREM04a] S. Farsiu, D. Robinson, M. Elad, and P Milanfar. Fast and robust multi-frame super-resolution. In *Proceedings SPIE Conference on Image Reconstruction from Incomplete Data*, number 13(10), pages 1327–1344, 2004.

[FREM04b] Sina Farsiu, Dirk M. Robinson, Michael Elad, and Peyman Milanfar. Dynamic demosaicing and color superresolution of video sequences. In *Conference on Image Reconstruction from Incomplete Data III, Proc. of SPIE*, volume 5562, pages 169–178, October 2004.

[Gau07] Matthew Gaubatz. Metrix mux visual quality assessment package. Online, 2007. http://foulard.ece.cornell.edu/gaubatz/metrix_mux.

[GBI09] D. Glasner, S. Bagon, and M. Irani. Super-resolution from a single image. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 349–356, September 2009.

[GDAG09] Giaime Ginesu, Tiziana Dessi, Luigi Atzori, and Daniele D. Giusto. Super-resolution reconstruction of video sequences based on back-projection and motion estimation. In *Proceedings of the 5th International ICST Mobile Multimedia Communications Conference*, Mobimedia '09, pages 25:1–25:7, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Ger74] R.W. Gerchberg. Super-resolution through error energy reduction. *J. Mod. Opt.*, (21(9)):709–720, 1974.

[Get11] Pascal Getreuer. Linear Methods for Image Interpolation. *Image Processing On Line*, 1, 2011. Online.

[GH97] J.J. Green and B.R. Hunt. Super-resolution in a synthetic aperture imaging system. In *International Conference on Image Processing, Proceedings of*, volume 1, pages 865–868, October 1997.

[Gha90] M. Ghanbari. The cross-search algorithm for motion estimation [image coding]. *Communications, IEEE Transactions on*, 38(7):950–953, July 1990.

[Ghe06] Frank Ghenassia. *Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Gib50] James J. Gibson. *The Perception of the Visual World.* Houghton Mifflin, 1950.

[GJZGSZ00] D. D. Gajski, Domer Jianwen Zhu, R. Gerstlauer, and A. Shuqing Zhao. *SPECC: Specification Language and Methodology*. Springer, first edition, 2000.

[Goh13] S. Gohshi. Limitation of super resolution image reconstruction for video. In *Computational Intelligence, Communication Systems and Networks (CICSyN), 2013 Fifth International Conference on*, pages 217–221, June 2013.

[Goh14] Seiichi Gohshi. 4k-to-8k tv up-converter with super resolution. In *Annual Technical Conference Exhibition, SMPTE 2014*, pages 1–6, October 2014.

[Gro02] Thorsten Grotker. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[Gro09] OSCI TLM Working Group. *OSCI TLM-2.0 language reference manual*. Open SystemC Initiative (OSCI), July 2009.

[GSH+11] T. Goto, S. Suzuki, S. Hirano, M. Sakurai, and T.Q. Nguyen. Fast and high quality learning-based super-resolution utilizing tv regularization method. In *18th IEEE International Conference on Image Processing (ICIP)*, pages 1185–1188, September 2011.

[GSP86] A. Goshtasby, George C. Stockman, and Carl V. Page. A region-based approach to digital image registration with subpixel accuracy. *Geoscience and Remote Sensing, IEEE Transactions on*, GE-24(3):390–399, May 1986.

[Gut15] Eduardo Quevedo Gutierrez. *Contribuciones al proceso de Super-Resolutcion mediante tecnicas de filtros selectivos, topologia de Macro-Bloques y sistemas Multi-Camara*. PhD thesis, University of Las Palmas de Gran Canaria, April 2015.

[GW08] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*, pages 65–68. Prentice Hall, Inc., 3rd edition, 2008.

[Has14a] Hasselblad. Hasselblad h5d 200c. Online, 2014. http://static.hasselblad.com/2014/11/H5D-200c-MS_Datasheet_EN_v3.pdf.

[Has14b] Hasselblad. Hasselblad multi-shot cameras. Online, 2014. http://www.hasselblad.com/medium-format/h5d-multi-shot.

[HBA97] R.C. Hardie, K.J. Barnard, and E.E. Armstrong. Joint MAP registration and high-resolution image estimation using a sequence of undersampled images. *Image Processing, IEEE Transactions on*, 6(12):1621–1633, December 1997.

[HFL10] Y. HaCohen, R. Fattal, and D. Lischinski. Image upsampling via texture hallucination. In *Computational Photography (ICCP), 2010 IEEE International Conference on*, pages 1–8, March 2010.

[HKK97] M.-C. Hong, Moon Gi Kang, and A.K. Katsaggelos. An iterative weighted regularized algorithm for improving the resolution of video sequences. In *Image Processing, 1997. Proceedings., International Conference on*, volume 2, pages 474–477, October 1997.

[HMM99] Takahiro Hamada, Satoshi Miyaji, and Shuichi Matsumoto. Picture quality assessment system by three-layered bottom-up noise weighting considering human visual perception. *SMPTE Journal*, 108(1):20–26, January 1999.

[HN10] Rodrigo Herrera Navarro. Modelado SystemC, simulacion y sintesis de un subsistema para mejora de video. Master's thesis, Universidad de Las Palmas de Gran Canaria. Escuela de Ingeniería de Telecomunicación y Electrónica, 2010.

[HS12] Kwok-Wai Hung and Wan-Chi Siu. Single image super-resolution using iterative wiener filter. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 1269–1272, March 2012.

[HYCC07] Yu He, Kim-Hui Yap, Li Chen, and L.-P. Chau. A nonlinear least square technique for simultaneous image registration and super-resolution. *Image Processing, IEEE Transactions on*, 16(11):2830–2841, November 2007.

[HZ07] Gang Hong and Yun Zhang. Combination of feature-based and area-based image registration technique for high resolution remote sensing image. In *Geoscience and Remote Sensing Symposium, 2007. IGARSS 2007. IEEE International*, pages 377–380, July 2007.

[IEE02] IEEE. *IEEE Standard VHDL Language Reference Manual*, 2002.

[IEE06] IEEE. *IEEE Standard for Verilog Hardware Description Language*, 2006.

[IEE09] IEEE. *IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*, December 2009.

[Ins06] The Institute of Electrical and Electronics Engineers, Inc., 3 Park Avenue, New York, NY 10016-5997, USA. *IEEE Standard SystemC Language Reference Manual*, March 2006.

[IP91] Michal Irani and Shmuel Peleg. Improving resolution by image registration. *CVGIP: Graph. Models Image Process.*, 53(3):231–239, 1991.

[IP92] M. Irani and S. Peleg. Image sequence enhancement using multiple motions analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proc. of*, pages 216–221, June 1992.

[ITU94a] ITU. *Specification and alignment procedures for setting of brightness and contrast of displays*, 1994.

[ITU94b] ITU. *Specification of a signal for measurement of the contrast ratio of displays*, 1994.

[ITU98a] ITU. *Subjective assessment methods for image quality in high-definition television*, 1998.

[ITU98b] ITU. *Subjective assessment of standard definition digital television (SDTV) systems*, 1998.

[ITU98c] ITU. *Worldwide unified colorimetry and related characteristics of future television and imaging systems*, 1998.

[ITU02] ITU. *Methodology for the subjective assessment of the quality of television pictures*, 2002.

[ITU08] ITU. *Subjective video quality assessment methods for multimedia applications*, 2008.

[JG05] K. Jia and S. Gong. Face super-resolution using multiple occluded images of different resolutions. In *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pages 614–619, September 2005.

[JJ81] J. Jain and A. Jain. Displacement measurement and its application in interframe image coding. *Communications, IEEE Transactions on*, 29(12):1799–1808, December 1981.

[JWB03] Zhongding Jiang, Tien-Tsin Wong, and Hujun Bao. Practical super-resolution from dynamic video sequences. In *Computer Vision and Pattern Recognition. Proceedings. IEEE Computer Society Conference on*, volume 2, pages II–549–II–554, June 2003.

[KCR09] Changhyun Kim, Kyuha Choi, and Jong Beom Ra. Improvement on learning-based super-resolution by adopting residual information and patch reliability. In *16th IEEE International Conference on Image Processing (ICIP)*, pages 1197–1200, November 2009.

[KDM+05] N. Kroupis, M. Dasygenis, K. Markou, D. Soudris, and A. Thanailakis. A modified spiral search motion estimation algorithm and its embedded system implementation. In *IEEE International Symposium on Circuits and Systems ISCAS*, volume 4, pages 3347–3350, May 2005.

[Key81] R. Keys. Cubic convolution interpolation for digital image processing. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 29(6):1153–1160, December 1981.

[KHX+06] Dan Kong, Mei Han, Wei Xu, Hai Tao, and Yihong Gong. Video super-resolution with scene-specific priors. In *British Machine Vision Conference (BMVC), Proceedings of*, pages 549–558, 2006.

[KI12] T. Katsuki and M. Inoue. Posterior mean super-resolution with a compound gaussian markov random field prior. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 841–844, March 2012.

[KIH+81] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro. Motion compensated interframe coding for video conferencing. In *National Telecommunications Conference (NTC), Proc. of*, pages G.5.3.1–G.5.3.4, November 1981.

[KKC+10] Minjae Kim, Bonhwa Ku, Daesung Chung, Hyunhak Shin, Bonghyup Kang, D.K. Han, and Hanseok Ko. Robust dynamic super resolution under inaccurate motion estimation. In *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, pages 323–328, August 2010.

[Koc13] Dirk Koch. *Partial Reconfiguration on FPGAs*, volume 153 of *Lecture Notes in Electrical Engineering*, chapter 2.7.1.1 Fan-out Impact on the Signal Latency. Springer-Verlag New York, 1 edition, 2013.

[KSS+12] Y. Kawamoto, S. Suzuki, Y. Sakuta, T. Goto, and M. Sakurai. A study on fast learning-based super-resolution utilizing tv regularization for hdtv. In *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, pages 725–726, January 2012.

[LCA07] Audiovisual Communications Laboratory LCAV. Super-resolution lcav. Online, 2007.

[LG00] F. Lopes and M. Ghanbari. Hierarchical motion estimation with spatial transforms. In *International Conference on Image Processing, Proc. of*, volume 2, pages 558–561, September 2000.

[LSZ97] A. Lorette, H. Shekarforoush, and J. Zerubia. Super-resolution with adaptive regularization. In *Image Processing. Proceedings. International Conference on*, volume 1, pages 169–172, October 1997.

[LWBK02] Ligang Lu, Zhou Wang, A.C. Bovik, and J. Kouloheris. Full-reference video quality assessment considering structural distortion and no-reference quality evaluation of mpeg video. In *IEEE International Conference on Multimedia and Expo, Proc. of*, volume 1, pages 61–64, 2002.

[LZL94] Renxiangand Li, Bing Zeng, and M.L. Liou. A new three-step search algorithm for block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 4(4):438–442, August 1994.

[Mal06] Stephane Mallat. Super-resolution bandlet upconvertion for hdtv. Whitepaper, Let It Wave, May 2006. Let it wave whitepaper.

[Mat96] The MathWorks, Inc. *Matlab Language Reference*, December 1996.

[MB08] William Malpica and Alan C. Bovik. Range image quality assessment by structural similarity. In *Encyclopedia of Multimedia*, pages 755–757. Springer US, 2008.

[MC03] Gustavo Ivan Marrero Callico. *Real-time and low-cost super-resolution algorithms onto hybrid video encoders.* PhD thesis, Universidad de Las Palmas de Gran Canaria, Escuela Tecnica Superior de Ingenieros de Telecomunicacion, 2003.

[MEMS14] Pedram Mohammadi, Abbas Ebrahimi-Moghadam, and Shahram Shirani. Subjective and objective quality assessment of image: A survey. *Journal of Visual Communication and Image Representation*, 2014.

[Mil10] Peyman Milanfar, editor. *Super-Resolution Imaging (Digital Imaging and Computer Vision)*. CRC Press, September 2010.

[MJK12] Pingfan Meng, M. Jacobsen, and R. Kastner. FPGA-GPU-CPU heterogenous architecture for real-time cardiac physiological optical mapping. In *Field-Programmable Technology (FPT), 2012 International Conference on*, pages 37–42, December 2012.

[Mjo85] E. Mjolsness. *Neural networks, pattern recognition, and fingerprint hallucination*. PhD thesis, California Institute of Technology, 1985.

[MN13] B.N. Madhukar and R. Narendra. Lanczos resampling for the digital processing of remotely sensed images. In *International Conference on VLSI, Communication, Advanced Devices, Signals & Systems and Networking (VCASAN-2013), Proc. of*, volume 258 of *Lecture Notes in Electrical Engineering*, pages 403–411. Springer India, 2013.

[MP07] Stéphane Mallat and Gabriel Peyré. A review of bandlet methods for geometrical image representation. *Numerical Algorithms*, 44(3):205–234, 2007.

[NEC09] NEC Electronics America, Inc. *NEC μPD9245GJEC*, January 2009. Document No. 51082.

[NFMM13] M. Naylor, P.J. Fox, A.T. Markettos, and S.W. Moore. Managing the FPGA memory wall: Custom computing or vector processing? In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–6, September 2013.

[NHBS07] B. Narayanan, R.C. Hardie, K.E. Barner, and Min Shao. A computationally efficient super-resolution algorithm for video processing using partition filters. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(5):621–634, May 2007.

[Nik10] Nikon. Stochastic optical reconstruction microscopy (storm) imaging. Online, 2010. https://microscopyu.com/tutorials/flash/superresolution/storm/index.html.

[NM14] Kamal Nasrollahi and Thomas B. Moeslund. Super-resolution: A comprehensive survey. *Mach. Vision Appl.*, 25(6):1423–1468, August 2014.

[NSLZ07] Michael K Ng, Huanfeng Shen, Edmund Y Lam, and Liangpei Zhang. A total variation regularization based super-resolution reconstruction algorithm for digital video. *EURASIP Journal on Advances in Signal Processing*, 2007(1):074585, 2007.

[OII+14] H. Okuhata, R. Imai, M. Ise, R.Y. Omaki, H. Nakamura, S. Hara, and I. Shirakawa. Implementation of dynamic-range enhancement and super-resolution algorithms for medical image processing. In *Consumer Electronics (ICCE), 2014 IEEE International Conference on*, pages 181–184, January 2014.

[OIOS13] H. Okuhata, M. Ise, R.Y. Omaki, and I. Shirakawa. Implementation of super-resolution scaler for full hd and 4k video. In *Consumer Electronics Berlin (ICCE-Berlin), 2013. ICCEBerlin 2013. IEEE Third International Conference on*, pages 1–5, September 2013.

[OLL+04] EePing Ong, Weisi Lin, Zhongkang Lu, Susu Yao, and M. Etoh. Visual distortion assessment with emphasis on spatially transitional regions. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(4):559–566, April 2004.

[Oly14] Olympus. E-m5 mark ii: Boost your resolution. Online, 2014. http://www.olympus.co.uk/site/en/c/cameras/om_d_system_cameras/om_d/e_m5_mark_ii/index.html.

[PA98] A.J. Patti and Y. Altunbasak. Artifact reduction for POCS-based super resolution with edge adaptive regularization and higher-order interpolants. In *IEEE International Conference on Image Processing (ICIP), Proceedings of*, pages 217–221, October 1998.

[PC09] D. Patel and S. Chaudhuri. Performance analysis for image super-resolution using blur as a cue. In *Advances in Pattern Recognition, 2009. ICAPR '09. Seventh International Conference on*, pages 73–76, February 2009.

[PCC+14] A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G.P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P.Y. Xiao, and D. Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 13–24, June 2014.

[PE09] M. Protter and M. Elad. Super resolution with probabilistic motion estimation. *Image Processing, IEEE Transactions on*, 18(8):1899–1904, August 2009.

[PETM09] M. Protter, M. Elad, H. Takeda, and P. Milanfar. Generalizing the nonlocal-means to super-resolution reconstruction. *Image Processing, IEEE Transactions on*, 18(1):36–51, January 2009.

[PHS87] A. Puri, H.-M. Hang, and D.L. Schilling. An efficient block-matching algorithm for motion-compensated coding. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 12, pages 1063–1066, April 1987.

[PJ07] V Patanavijit and S Jitapunkul. A lorentzian stochastic estimation for a robust iterative multiframe super-resolution reconstruction with lorentzian-tikhonov regularization. *EURASIP Journal on Advances in Signal Processing*, 2007(1):034821, 2007.

[PKS87] S. Peleg, D. Keren, and L. Schweitzer. Improving image resolution using subpixel motion. In *Pattern Recognit. Lett.*, number 5(3), pages 223–226. 1987.

[PM96] Lai-Man Po and Wing-Chung Ma. A novel four-step search algorithm for fast block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(3):313–317, June 1996.

[PM08] Gabriel Peyre and Stephane Mallat. Orthogonal bandelet bases for geometric images approximation. *Communications on Pure and Applied Mathematics*, 61(9):1173–1212, 2008.

[PPK03] Sung Cheol Park, Min Kyu Park, and Moon Gi Kang. Super-resolution image reconstruction: a technical overview. *Signal Processing Magazine, IEEE*, 20(3):21–36, May 2003.

[PRZ03] L. C. Pickup, S. J. Roberts, and A. Zisserman. A sampled texture prior for image super-resolution. In *Advances in Neural Information Processing Systems*, pages 1587–1594, 2003.

[PS00] Thrasyvoulos N. Pappas and Robert J. Safranek. Perceptual criteria for image quality evaluation. In *in Handbook of Image and Video Processing*, pages 669–684. Academic Press, 2000.

[QdLCC+14] E. Quevedo, J. de La Cruz, G. Callico, F. Tobajas, and R. Sarmiento. Video enhancement using spatial and temporal super-resolution from a multicamera system. *Consumer Electronics, IEEE Transactions on*, 60(3):420–428, August 2014.

[RB05] S. Roth and M.J. Black. Fields of experts: a framework for learning image priors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proc. of*, volume 2, pages 860–867, June 2005.

[RC01] D. Rajan and S. Chaudhuri. Generation of super-resolution images from blurred observations using markov random fields. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on*, volume 3, pages 1837–1840, 2001.

[Ric04] Iain E. G. Richardson. *H.264 and MPEG-4 Video Compression*. John Wiley & Sons, Ltd, 2004.

[Ric14] Ricoh. Pentax k3 ii: Pixel shift resolution. Online, 2014. http://www.ricoh-imaging.co.jp/english/products/k-3-2/.

[RRYB13] R. Raghavendra, K.B. Raja, Bian Yang, and C. Busch. Comparative evaluation of super-resolution techniques for multi-face recognition using light-field camera. In *Digital Signal Processing (DSP), 2013 18th International Conference on*, pages 1–6, July 2013.

[SBPL11] J. Silvestre-Blanes and R. Perez-Llorens. Ssim and their dynamic range for image quality assessment. In *International Symposium ELMAR, Proc. of*, pages 93–96, September 2011.

[SHFC+08] A. Sanchez, P. Hernandez Fernandez, P. P. Carballo, G. M. Callico, and A. Nunez. Transaction level modeling for a super-resolution hardware-engine design. In *XXIII Conference on Design of Circuits and Integrated Systems DCIS 08*, 2008.

[SKC+13] S. Sriwilai, T. Kasetkasem, T. Chanwimaluang, T. Srinark, and T. Isshiki. A super-resolution mapping algorithm based on the level set method. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2013 10th International Conference on*, pages 1–6, May 2013.

[Ski05] John Skidgel. *Producing 24p Video*. Digital Video Expert Series. CMP Books, 2005.

[SKR07] K.V. Suresh, G.M. Kumar, and A.N. Rajagopalan. Superresolution of license plates in real traffic videos. *Intelligent Transportation Systems, IEEE Transactions on*, 8(2):321–331, June 2007.

[SLJT08] Qi Shan, Zhaorong Li, Jiaya Jia, and Chi-Keung Tang. Fast image/video upsampling. *ACM Trans. Graph.*, 27(5):153:1–153:7, 2008.

[SO89] Henry Stark and Peyma Oskoui. High-resolution image recovery from image-plane arrays, using convex projections. *Journal of The Optical Society of America A-optics Image Science and Vision*, 6:1715–1726, 1989.

[SOHW12] G.J. Sullivan, J. Ohm, Woo-Jin Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(12):1649–1668, December 2012.

[SS94] R.R. Schultz and R.L. Stevenson. A bayesian approach to image expansion for improved definition. *Image Processing, IEEE Transactions on*, 3(3):233–242, May 1994.

[SS96] R.R. Schultz and R.L. Stevenson. Extraction of high-resolution frames from video sequences. *Image Processing, IEEE Transactions on*, 5(6):996–1011, June 1996.

[SSBC10] K. Seshadrinathan, R. Soundararajan, AC. Bovik, and L.K. Cormack. Study of subjective and objective quality assessment of video. *Image Processing, IEEE Transactions on*, 19(6):1427–1441, June 2010.

[SSXS08] Jian Sun, Jian Sun, Zongben Xu, and Heung-Yeung Shum. Image super-resolution using gradient profile prior. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proc. of*, pages 1–8, June 2008.

[SSXS11] Jian Sun, Jian Sun, Zongben Xu, and Heung-Yeung Shum. Gradient profile prior and its applications in image super-resolution and enhancement. *Image Processing, IEEE Transactions on*, 20(6):1529–1542, June 2011.

[STdA13] G. Singla, F. Tobajas, and V. de Armas. Video super resolution algorithm implemented on a low-cost noc-based mpsoc platform. In *Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on*, pages 1–6, December 2013.

[Swa01] Stuart Swan. *An Introduction to System Level Modeling in SystemC 2.0*, 2001.

[Sze10] Richard Szeliski. *Computer Vision: Algorithms and Applications (Texts in Computer Science)*. Springer, 2011 edition, November 2010.

[SZTS03] Jian Sun, Nan-Ning Zheng, Hai Tao, and Heung-Yeung Shum. Image hallucination with primal sketch priors. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–729–36, June 2003.

[TCH+09] M. Thadani, P. P. Carballo, P. Hernandez, Gustavo M. Callico, and A. Nunez. ESL flow for a hardware H.264/AVC decoder using TLM-2.0 and high level synthesis: a quantitative study. In *VLSI Circuits and Systems IV*, volume 7363, pages 73630K–73630K–12, May 2009.

[TH84] R. Tsai and T. Huang. Multiframe image restoration and registration. *Advances in Computer Vision and Image Processing*, 1:317–339, 1984. JAI Press Inc.

[TH86] Qi Tian and Michael N Huhns. Algorithms for subpixel registration. *Comput. Vision Graph. Image Process.*, 35(2):220–233, August 1986.

[TK08] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, 4th edition, 2008.

[TM10] Jing Tian and Kai-Kuang Ma. Stochastic super-resolution image reconstruction. *Journal of Visual Communication and Image Representation*, 21(3):232 – 244, 2010.

[TM11] Jing Tian and Kai-Kuang Ma. A survey on super-resolution imaging. *Signal, Image and Video Processing*, 5(3):329–342, 2011.

[TMPE09] H. Takeda, P. Milanfar, M. Protter, and M. Elad. Super-resolution without explicit subpixel motion estimation. *Image Processing, IEEE Transactions on*, 18(9):1958–1975, September 2009.

[TSC+09] M. Thadani, T. Szydzik, P. P. Carballo, P. Hernandez, G. Marrero, and A. Nunez. ESL quantitative assessment of the optimization steps in an ESL flow for a hardware implementation of a h.264/avc decoder. In *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, 2009.

[vABVG+14] W. van Aarle, K.J. Batenburg, G. Van Gompel, E. Van de Casteele, and J. Sijbers. Super-resolution for computed tomography based on discrete tomography. *Image Processing, IEEE Transactions on*, 23(3):1181–1193, March 2014.

[VCB+10] A. Villa, J. Chanussot, J.A. Benediktsson, M. Ulfarsson, and C. Jutten. Super-resolution: an efficient method to improve spatial resolution of hyperspectral images. In *Geoscience and Remote Sensing Symposium (IGARSS), 2010 IEEE International*, pages 2003–2006, July 2010.

[VCT+10] L.G. Villanueva, G.M. Callico, F. Tobajas, S. Lopez, V. De Armas, J.F. Lopez, and R. Sarmiento. Medical diagnosis improvement through image quality enhancement based on super-resolution. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 259–262, September 2010.

[VR77] G. J. Vanderbrug and A. Rosenfeld. Two-stage template matching. *IEEE Trans. Comput.*, 26(4):384–393, April 1977.

[WB02] Zhou Wang and A.C. Bovik. A universal image quality index. *Signal Processing Letters, IEEE*, 9(3):81–84, March 2002.

[WBS05] Zhou Wang, Alan C. Bovik, and Hamid R. Sheikh. *Structural Similarity Based Image Quality Assessment*, chapter 7, pages 225–241. Signal Processing and Communications. CRC Press, Inc, 2005.

[WBSS04] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, April 2004.

[WHK+14] Qiang Wu, Yajun Ha, A. Kumar, Shaobo Luo, Ang Li, and S. Mohamed. A heterogeneous platform with GPU and FPGA for power efficient high performance computing. In *Integrated Circuits (ISIC), 2014 14th International Symposium on*, pages 220–223, December 2014.

[WL07] Zhou Wang and Qiang Li. Video quality assessment using a statistical model of human visual speed perception. *Optical Society of America, Journal of the*, 24:B61–B69, December 2007.

[WLB02] Zhou Wang, Ligang Lu, and A.C. Bovik. Video quality assessment using structural distortion measurement. In *International Conference on Image Processing, Proc. of*, volume 3, pages III–65–III–68, 2002.

[WLBR97] Kyoung Won Lim and Jong Beom Ra. Improved hierarchical search block matching algorithm by using multiple motion vector candidates. *Electronics Letters*, 33(21):1771–1772, October 1997.

[WSB03a] Z. Wang, E.P. Simoncelli, and AC. Bovik. Multiscale structural similarity for image quality assessment. In *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1398–1402, November 2003.

[WSB03b] Zhou Wang, Hamid R. Sheikh, and Alan C. Bovik. Objective video quality assessment. In *The Handbook of Video Databases: Design and Applications*, pages 1041–1078. CRC Press, 2003.

[WT03] Xiaogang Wang and Xiaoou Tang. Face hallucination and recognition. In *Audio- and Video-Based Biometric Person Authentication*, volume 2688 of *Lecture Notes in Computer Science*, pages 486–494. Springer Berlin Heidelberg, 2003.

[Xil07] Inc Xilinx. Virtex-ii platform FPGAs: Complete data sheet. Product Specification v3.5, Xilinx, Inc, November 2007.

[Xil09] Inc Xilinx. Virtex-5 family overview. Product Specification v5.0, Xilinx, Inc, February 2009.

[Xip15] Xiph.org. Xiph.org video test media [derf's collection]. Online, 2015.

[XSW09] Zhiwei Xiong, Xiaoyan Sun, and Feng Wu. Web cartoon video hallucination. In *16th IEEE International Conference on Image Processing (ICIP)*, pages 3941–3944, November 2009.

[XSW10] Zhiwei Xiong, Xiaoyan Sun, and Feng Wu. Robust web image/video super-resolution. *Image Processing, IEEE Transactions on*, 19(8):2017–2028, August 2010.

[XZZ14] Xiong Xu, Yanfei Zhong, and Liangpei Zhang. Adaptive subpixel mapping based on a multiagent system for remote-sensing imagery. *Geoscience and Remote Sensing, IEEE Transactions on*, 52(2):787–804, February 2014.

[YB06] Jiangang Yu and B. Bhanu. Super-resolution restoration of facial images in video. In *Pattern Recognition (ICPR). 18th International Conference on*, volume 4, pages 342–345, 2006.

[YMB11] Chen Yue-Meng and I.V. Bajic. Spatio-temporal super-resolution from compressed video employing global and local motion. In *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on*, pages 907–912, August 2011.

[YU14] Zihan Yu and Kiichi. Urahama. Iterative method for inverse nonlinear image processing. In *Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E97-A, pages 719–721, February 2014.

[YZS12] Qiangqiang Yuan, Liangpei Zhang, and Huanfeng Shen. Multiframe super-resolution employing a spatially weighted total variation model. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(3):379–392, March 2012.

[ZF03] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21:977–1000, 2003.

[ZH12] Xiao Zeng and Hua Huang. Super-resolution method for multiview face recognition from a single image per person using nonlinear mappings on coherent features. *Signal Processing Letters, IEEE*, 19(4):195–198, April 2012.

[ZHD12] Di Zhang, Jiazhong He, and Minghui Du. Morphable model space based face super-resolution reconstruction and recognition. *Image Vision Comput.*, 30(2):100–108, February 2012.

[ZLRG12] Yue Zhuo, Jiaying Liu, Jie Ren, and Zongming Guo. Nonlocal based super resolution with rotation invariance and search window relocation. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 853–856, March 2012.

[ZM00] Shan Zhu and Kai-Kuang Ma. A new diamond search algorithm for fast block-matching motion estimation. *Image Processing, IEEE Transactions on*, 9(2):287–290, February 2000.

[ZM07] M.V.W. Zibetti and J. Mayer. A robust and computationally efficient simultaneous super-resolution scheme for image sequences. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(10):1288–1300, October 2007.

[ZP00] A. Zomet and S. Peleg. Efficient super-resolution and applications to mosaics. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 1, pages 579–583, 2000.

[ZPJ08] Xuesong Zhang, Silong Peng, and Jing Jiang. An adaptive learning method for face hallucination using locality preserving projections. In *Automatic Face Gesture Recognition, 2008. FG '08. 8th IEEE International Conference on*, pages 1–8, September 2008.

[ZRAP01] A. Zomet, A. Rav-Acha, and S. Peleg. Robust super-resolution. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Proc. of*, volume 1, pages I–645–I–650, 2001.

[ZZLH12] Haichao Zhang, Y. Zhang, H. Li, and T.S. Huang. Generative bayesian image super resolution with natural image prior. *Image Processing, IEEE Transactions on*, 21(9):4054–4067, September 2012.

TESIS DOCTORAL