Microprocessors and Microsystems 104 (2024) 104987

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Abeto: An automated benchmarking tool to manage heterogeneous IP core databases

Antonio J. Sánchez[a], Yubal Barrios[a,*], Lucana Santos[b], Roberto Sarmiento[a]

[a] Institute for Applied Microelectronics, Univ. of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, 35017, Spain
[b] European Space Research and Technology Centre, European Space Agency, Noordwijk, 2201, The Netherlands

ARTICLE INFO

Keywords:
Computer aided engineering
Electronic design automation
Database management
IP cores
System-level design

ABSTRACT

System-level design makes use of building blocks, known as soft IP cores, to build complex developments. The usage of these IP cores allows to reduce design and verification time, and also to save costs. However, the use of third-party IP cores tends to present difficulties because of a lack of standardization in their organization, distribution and management, which derive in heterogeneous databases. Most of the time, system developers need to describe some additional code to enable the integration, verification and validation of the IP core, which is not available as part of their distribution. This implies acquiring a deep knowledge of each IP core, often with a large learning curve.

In this work Abeto is presented, a new software tool for IP core databases management. It allows to easily integrate and use a heterogeneous group of IP cores, described in VHDL, with a unified set of instructions or commands. In order to do so, Abeto requires from every IP core some side information about its packaging and how to operate with the IP. Currently, Abeto provides support for a set of well-known EDA tools and has been successfully applied to the European Space Agency portfolio of IP cores for benchmarking purposes. To demonstrate its performance, mapping results for these IP cores on the novel NanoXplore BRAVE FPGA family are provided.

1. Introduction

Digital electronics and circuits have been continuously growing in complexity during the last decades. This has motivated a variety of design methodologies (e.g., system-level design or high-level synthesis), and common practices in both industry and academia, aiming at reducing development costs, design effort and time-to-market. Among these techniques, collaborative design based on reusing previous developments or building blocks, commonly known as Intellectual Property (IP) cores, is a trend for different applications, including the space industry [1]. The adoption of these IP cores is normally managed through their licensing from other vendors or by adapting open-source alternatives. Plenty of IP cores are available in either commercial [2,3] or open-source repositories [4,5].

All these practices imply a know-how transfer from IP developers to potential users, so the designs not only are required to be functional, but also understandable and manageable by users [6,7]. Nevertheless, there has been historically a lack of standardization in the way that IP cores and their auxiliary components are developed, packaged and documented. This makes the IP reuse not straightforward, having each IP core a unique learning curve and usually leading to extra efforts adding Electronic Design Automation (EDA) tool support for each design.

Both the industry and academia have proposed some solutions to facilitate the exchange and reuse of IP cores. On the one hand, standards have been proposed for IP metadata documentation, such as those from the Accellera initiative [8]. Among them, IP-XACT [9], an XML-based standard for IP packaging and interchange intended for highly-automated design environments (e.g., Xilinx Vivado), must be highlighted as it has been adopted by IEEE. IP-XACT allows to link tools into an automated system development framework, which is able to understand, configure and launch different IP processes related to their metadata, but not to functional features such as simulation or synthesis. However, the application and adoption of such standards is not general yet, mainly for the deep knowledge required for their usage.

On the other hand, there are several open-source IP management applications developed [10–12], including IP-XACT front-ends [13–15]. Among the academic applications stand out FuseSoC [10], an open-source package manager which incorporates a build tool for Hardware Description Languages (HDLs) and thought for System-on-Chip (SoC) development; and Kaktus2 [13], an IP-XACT front-end incorporating

* Corresponding author.
E-mail address: ybarrios@iuma.ulpgc.es (Y. Barrios).

https://doi.org/10.1016/j.micpro.2023.104987
Received 9 March 2023; Received in revised form 24 October 2023; Accepted 13 December 2023
Available online 27 December 2023

extensions for reuse of software components of IP cores. However, these IP managers tend to be rigid in the expected IP cores format, usually requiring a noticeable adaptation effort to the tool requirements.

Due to the increasing number of open-source IP management tools, initiatives have appeared which try to establish a common ground in this field. In particular, EDA[2] [16] proposes a conceptual model for the different abstraction layers present in EDA tools and tool chains. It also provides a common framework for open-source Python-based EDA management tools, where different front-ends and workflows can be integrated.

In an attempt to reuse previous designs and thus reduce development time and costs, the European Space Agency (ESA) provides a portfolio of IP cores [17], described in VHDL, which can used in different system developments for future space missions. This portfolio is continuously growing, adding new IP cores that fulfill the demands of potential applications with interest for the space industry. Each IP core has its own database, including all the necessary components and their associated information to enable the reusability of the IP. However, each database is produced by a team that follows its own workflow, resulting on different structures and implementation processes. This results in an heterogeneous IP core database with a lack of standardization.

In this work we introduce Abeto (Automated Benchmarking Tool), a software IP management tool intended to integrate heterogeneous groups of IP cores and operate with them in a unified manner. As use case, this tool is applied to better organize the ESA portfolio of IP cores. However, Abeto has been designed with versatility in mind, adapting its performance to other IP databases and allowing extending the catalog of supported EDA tools. The different stages of the IP workflow, such as configuration, simulation or synthesis, can be configured and launched from Abeto in a transparent way from the user point of view. This allows to accelerate iterations in the design flow in case that changes in the IP cores are introduced or new target technologies are targeted. Although Abeto is currently provided as an stable tool, it is constantly evolving to incorporate new features that can be required by potential users.

The rest of the paper is structured as follows. Section 2 gives a brief overview of Abeto, while Section 3 explains the use of the Abeto through its command-line interface. Section 4 introduces the auxiliary IP core description files necessary for IP integration in Abeto. An example of the Abeto usage targeting a simple IP core is then described in Section 5. Later, Section 6 provides a preliminary experience about the tool usage and results with the ESA portfolio of IP cores. Finally, Section 7 draws some conclusions.

2. Overview

Abeto is a software tool intended to serve as an IP management framework, which has been fully developed in Python and supports UNIX-based systems (tested in Ubuntu 18.04 and 20.04, and also in Red Hat 7) and Windows 10. The main idea behind Abeto consists in unifying the design flow steps (e.g., configuration, simulation, synthesis) of different IP cores – potentially heterogeneous – under a common set of commands, thus making most of IP-specific details transparent to the user. Fig. 1 illustrates this concept. Although Abeto has been initially developed to manage the ESA IP cores portfolio, it is versatile enough to manage any other IP core repository if it is properly adapted to be understood by the tool.

Abeto has also the capability of interacting with several EDA tools. Currently, the following third-party tools are supported: Mentor Questasim and GHDL for simulation; Xilinx ISE and Vivado, Synopsys Synplify and the novel NanoXplore NXmap for FPGA synthesis; and Synopsys Design Compiler for ASIC synthesis. Support for the NanoXplore BRAVE family, the first high-performance, radiation-hardened reprogrammable European FPGA [18], demonstrates the goodness of the Abeto tool to be adapted to new technologies that are not widely

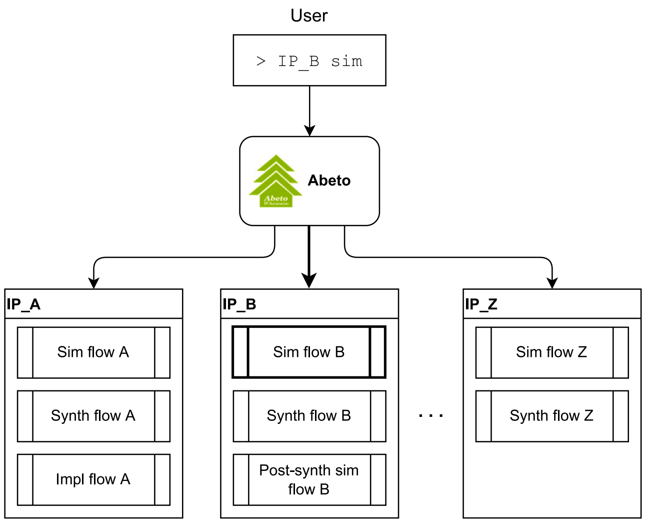

Fig. 1. Abeto overview.

supported by commercial EDA tools because of their reduced number of users. This list of tools can be extended in the near future, in case other tools will be identified as interesting for specific IP databases and their potential users. These supported third-party EDA tools can be invoked by Abeto to run different steps of the design flow. To ease this interaction, Abeto provides an auxiliary package with methods that can be called in these third-party EDA tools to access the Abeto IP database and automatize several operations.

Abeto comprises an IP core database, which centralizes the management of soft IP cores, and a Command-Line Interface (CLI), which allows operating with the IP cores integrated in the database for benchmarking purposes. IP cores can be attached and detached from the Abeto database in a plug & play fashion using available commands in the Abeto CLI. In order to do so, and taking into account the potentially heterogeneous nature of the IP cores Abeto shall handle, the tool requires some information of each IP core to be integrated on its database: its directory structure and a list of which operations can be performed with that IP, which must be provided with specific formats. These are jointly denoted as IP description files. Abeto is distributed along with an example IP core, a Gray counter of configurable width, named as *dummyIP*. This IP core is pre-loaded in the Abeto IP database, including a self-checking testbench and the necessary scripts for its operation through Abeto. The purpose of the *dummyIP* is to provide to IP developers a way to organize their IP database to be integrated as part of the ESA IP portfolio. A brief tutorial of Abeto using this IP is later explained in Section 5.

Once installed, Abeto designates a directory to allocate the Abeto IP database. From here, IP cores integrated in Abeto are accessed and operated, either directly if IP cores are installed in the same directory, or by means of symbolic links if they are installed elsewhere in the workstation. It is important to not move the location of IP cores in the workstation once they are integrated in the Abeto IP database, otherwise Abeto will not be able to access them. If that happens, it is necessary to first remove the IP core from the Abeto IP database and include it again, indicating the new location of the IP core. In addition, Abeto stores a log file with the contents of the Abeto IP database. This file is dynamically updated with any change in the IP database, and it allows to restore the IP database each time Abeto is launched. Fig. 2 illustrates the organization of the Abeto IP database.

By default, Abeto operates with IP cores assuming they have a predefined directory structure. This default directory structure is shown in Fig. 3. If the directory tree of an IP does not match this baseline structure, Abeto must be informed of the real directory structure for a proper operation, as explained later in Section 4.1.

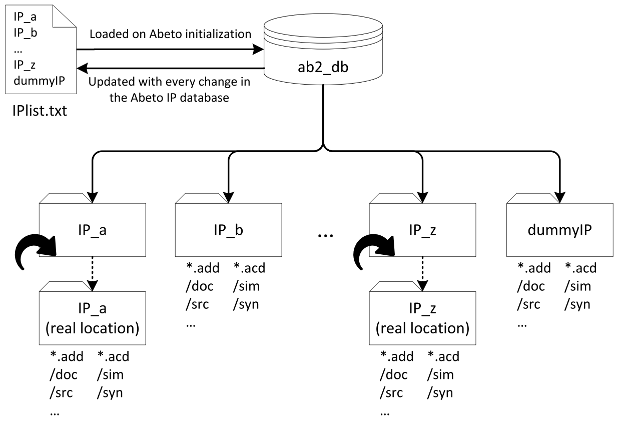

Fig. 2. Abeto IP database.

```
IP-root
├─ src                  └─ syn
│  ├─ core                 ├─ synplify
│  ├─ utils                │  ├─ scripts
│  └─ tb                   │  ├─ reports
├─ doc                     │  └─ outputs
├─ bin                     ├─ dc
└─ sim                     │  ├─ scripts
   ├─ scripts              │  ├─ reports
   ├─ stimuli              │  └─ outputs
   ├─ config               └─ brave
   ├─ reference               ├─ scripts
   ├─ reports                 ├─ reports
   └─ outputs                 └─ outputs
```

Fig. 3. Baseline IP directory structure.

3. Abeto command-line interface

There are two main categories of commands supported in Abeto. There is a set of built-in commands, with the purpose of configuring Abeto and managing the IP database, while the other group of commands allows operating with each one of the IPs integrated in the Abeto database and launching the different design flow steps. In addition, Abeto incorporates a built-in IP configuration tool. All these are introduced in the next subsections. Besides, the CLI incorporates a batch mode and a special feature to support the execution of any command multiple times in sequence with argument variations, thus reducing the user intervention for some repetitive tasks, such as the generation of multiple configurations of the same IP or synthesizing it applying a succession of constraint files.

Runtime errors, either originated by Abeto itself or by any invoked EDA tool, are captured by Abeto. Then an appropriate error message is displayed, and control returns to the user by the Abeto CLI.

3.1. Abeto initialization

Abeto is launched by running the corresponding executable (depending on the OS) from Abeto installation directory. It can be executed with either one argument (the path to an Abeto shell script) or none, which starts Abeto in batch or manual mode, respectively. In any case, an initialization process begins where Abeto generates the necessary file structure (if it did not previously exist) and loads all the IPs already integrated in the Abeto IP database in the last session. Abeto stores the list of IPs in a text file. This file is read to know which IPs must be loaded during the initialization process. For each one of these IPs, Abeto internally loads its directory structure, as well as the list of available IP commands. Once the initialization is completed, if Abeto was launched in the manual mode, the Abeto prompt is displayed, enabling the use of the command-line interface; otherwise, if Abeto is launched in batch mode, it automatically executes the commands in the provided shell script in sequence and closes itself upon termination.

3.2. Built-in commands: Managing the IP database

Two ways to add IP cores to the Abeto IP database are provided. On the one hand, already designed IP cores are included using the *addIP* command (`addIP <IP identifier> <path>`). The tool expects in this case to find and parse the corresponding IP description files. On the other hand, the directory structure for a new IP core can be generated through the command *newIP* (`newIP <IP identifier> <path>`) to ease the integration of new IP cores in Abeto at the beginning of the design flow. Both cases require to provide the IP core location and a unique IP identifier, which is used to invoke the IP core in any subsequent commands within Abeto. With any of these changes, the internal list of IP cores is updated.

In case that changes are made to the IP description files of any IP core integrated in the Abeto database, they can be reloaded (*refreshIP*) to make Abeto aware of these changes. Other commands allow to remove IP cores from the Abeto IP database (*removeIP*), query information about them (*IPtree* to know the IP database organization, *help* or *where* command to find the IP location on the workstation), or to set configuration settings.

3.3. IP operation

IP cores are operated through a unified set of commands. These commands cover different steps of the IP design flow, from IP configuration (*config*) to synthesis (*syn*), including compilation (*build*), simulation (*sim*) and post-synthesis/post-P&R simulations (*ps-sim*, *par-sim*). Additional commands allow to validate results (*verify*), display IP documentation and/or reports (*doc*, *report*) and clean intermediate files (*clean*). However, not all these commands are necessarily available for all the IPs and, in addition, each IP may execute these commands on its particular way.

To solve these issues, every IP must provide a command dictionary file (more details in Section 4.2), which determines the IP commands which are enabled for that IP, the format these commands have (i.e., the number and name of additional arguments, if they are mandatory or optional, and a list of valid values for each argument) and the handling these commands receive specifically for the IP.

Nevertheless, there are some IP commands that have a predefined default behavior and they become available for each IP if certain conditions are met, even if they are not declared in the IP command dictionary or if this file is missing.

IP commands in Abeto are invoked by prepending the IP identifier, followed by any additional arguments, if required (i.e., with the form `<IP identifier> <IP command>`. For example, to simulate the *dummyIP*, the command `dummyIP sim` should be introduced. The IP commands a particular IP supports and the use of every IP command can be queried through the built-in help.

3.4. Batch mode

In addition to the manual (interactive) mode, Abeto provides a batch mode in which commands from a given shell script are automatically executed in sequence. This mode is entered by executing the Abeto launcher along with an extra argument: the path to an Abeto shell script. An Abeto shell script is a text file, with extension .acs, which includes a set of Abeto commands with one command per line. Commands in .acs files must be typed exactly as if they were entered manually by the Abeto CLI. An .acs file can be also provided to the default *config* command to enter the built-in configuration tool in batch mode. In batch mode, when all commands from an .acs file are exhausted, Abeto is automatically closed.

3.5. Abeto configuration tool

Abeto incorporates a built-in configuration tool, with the purpose of editing IP configuration parameters before simulations or synthesis runs in case that the IP does not provide its own configuration mechanism. This is accessed through the default *config* IP command, which is enabled as long as the corresponding IP description file identifies one or more source files which contain IP configuration parameters.

The Abeto configuration tool is an independent CLI inside Abeto with its own set of commands. It incorporates a simple VHDL syntax analyzer to extract the constant declarations in the given source code as configurable parameters. These parameters can be queried and modified through commands. IP configuration changes can be saved either by overwriting the original source file or by generating replicas of the configuration file. If any syntax error is found or no configuration parameters are found in the target configuration file, an error is raised and the configuration tool is closed.

When Abeto it is running in batch mode, the access to the Abeto built-in configuration tool is slightly different than through the CLI. In this case, attempting to enter the configuration tool in “manual” mode has the effect of resuming the execution of the same Abeto shell script inside the configuration tool. If the configuration tool is then closed, the same script continues executing in Abeto.

3.6. Multiple command execution through argument lists

To avoid users the necessity of introducing commands individually, Abeto offers the possibility of executing multiple commands from a single command-line instruction by using argument lists. This mechanism allows reducing the user intervention for some repetitive tasks.

Argument lists are enclosed by curly brackets (“{}”), using the vertical slash character (|) to separate different values of the same arguments. If a command includes an argument list it is first unrolled, generating multiple instances of the same command with the elements values in the argument list, and then all commands are executed in sequence. All Abeto commands support this feature, including IP commands or commands in the Abeto configuration tool.

4. IP core description files

Every IP to be integrated in Abeto should provide two auxiliary files: a Database Definition file (ADD), which provides information about the directory structure of the IP core and its integration in the Abeto database; and a Command Dictionary file (ACD), which includes the list of supported IP commands for that IP and how must they be executed. These files are required by Abeto to handle a heterogeneous group of IP cores in a common framework and they must be located in the IP root directory.

These auxiliary files are validated every time the corresponding IP core is loaded in the Abeto IP database, either during the initialization phase of a new Abeto session (see Section 3.1) or by using the built-in Abeto commands *addIP* or *refreshIP* (see Section 3.2). Any error found in the validation of these files causes the corresponding IP core to be rejected from the IP database, along with an error message indicating the cause of such error.

4.1. Abeto database definition

An ADD file is a text file, with extension *.add*, which informs about how the directory structure of a given IP is arranged. In particular, the ADD file must reflect the differences of the IP directory structure with respect to the baseline directory structure from Fig. 3. If the ADD file is not provided for a given IP, the baseline directory structure is presumed. In addition, other side information required for the IP core integration in the Abeto IP database shall be included, such as external dependencies or IP-specific environment variables.

```
remove bin
remove sim
remove syn
from src
    remove core
    remove utils
    remove tb
end
rename doc docu
rename src sources
add testbench is_trackable
```

(a)

```
Example-IP
├─ docu
├─ sources
└─ testbench
```

(b)

Fig. 4. (a) ADD file example; (b) resulting directory structure.

ADD files are comprised by a set of directives, which are parsed and executed in sequential order from the beginning to the end of the file. From the baseline directory structure and the modifications stated in the ADD files, Abeto internally builds an image of the directory structure of each IP in the IP database. Abeto works with the internal image of the directory structure to access the IP and operate with it, assuming that it matches the directory tree of the IP distribution. Fig. 4 shows an example on how to re-define the IP directory structure through ADD files.

Because the IP directory structure can be modified at will, a mechanism to denote the purpose of every folder in the IP directory structure is required. Directory flags are used with this aim. These flags allow to define which folders have trackable content, documentation files and executable files or scripts, among others. The baseline IP directory structure includes a predefined flag setting for each one of its directories, but this can be modified by means of directives in the ADD file. Unless otherwise stated, the properties indicated by these directory flags affect solely to the content of the corresponding directory (i.e., these properties are not propagated to children directories).

4.2. Abeto command dictionary

An ACD file is also a text file, with extension *.acd*, which informs Abeto about the IP commands available for a given IP and provides instructions about how to execute them. An example of an ACD file is given in Fig. 5. ACD files have two parts: header and body.

The ACD header indicates the supported IP commands that need a dedicated handling for the corresponding IP, along with the expected format of such IP commands. The list of IP commands to be defined in the ACD header must be written using one line per IP command. The format of these commands is highly configurable. Each IP command may define any number of arguments and they can be either mandatory or optional, and a list of supported values that can be defined for each argument. Argument definitions come after the IP command reserved word, in the same line. An argument is declared by the ‘$’ character, followed by the name identifying the argument (i.e., the argument identifier). Multiple arguments can be defined for any given IP command.

```
--Available Abeto commands--
build
clean
syn
sim $CONFIG={(0-11)|all}
--Command dictionary--
build: make compile
clean: make clean
syn:   make syn
sim $CONFIG=all: make all
sim:   make $CONFIG.sim
```

Fig. 5. Example ACD file.

The ACD body contains the handling of the commands declared in the ACD header, using a target-recipe format similar to a makefile. Targets are the IP commands introduced in Section 3.3, while the recipes constitute the actual set of instructions to execute for that target.

Targets defined in an ACD file may include requisites concerning command arguments. Thus, different handlers for the same IP command can be defined, depending on the value of its arguments or whether certain optional arguments are used or not. When an IP command is introduced through the Abeto CLI, only the first target in the ACD file whose requisites are met is executed, ignoring the rest.

The recipe of any target may include an arbitrary number of commands, including calls to EDA tools, make targets, shell commands, execution of scripts and binary files, and even calls to other Abeto IP commands. A recursion limit is implemented in order to avoid cyclic dependencies between IP commands, since an IP command is allowed to invoke other IP commands. IP commands execution will fail if such recursion limit is reached.

When multiple translation rules are defined for the same IP command by means of argument selectors, just the first rule found in the ACD body compatible with the current argument values is executed, ignoring the rest of the translation rules even if they also match. It is therefore a good practice to put first in the ACD body the more restrictive rules. It is also possible to include a last translation rule for the IP command with no argument selectors at the end of the ACD body, which would trigger for all the cases not covered with the previous translation rules.

In the case of scripts and binary files, it is necessary to grant them execution permissions in Abeto through the corresponding ADD file. Recipes may include wildcards to insert the values of the command arguments introduced through the Abeto CLI. By default, all commands in a recipe will be executed from the IP root directory, but the working directory can be modified on a command-by-command basis in the ACD file. It is also possible to redirect the standard input and output individually for each command in a recipe.

5. Abeto use case: dummyIP

By default, the *dummyIP* is included in Abeto to help new users to get familiarized with the IP operation by means of the Abeto command-line interface. As it is part of the Abeto IP database, it will be automatically loaded the first time the Abeto framework is launched. The *dummyIP* is provided along with the auxiliary IP description files required for its integration in Abeto. These files define the set of allowed commands for the *dummyIP*, and perform a minimal configuration of the IP file tree.

This section also aims to explain how to work with this IP core in Abeto, serving as a tutorial. The following subsections explains the different actions which can be performed with the *dummyIP*.

```
Abeto > help dummyIP

Supported IP commands for dummyIP:
========================================
clean config par-sim ps-sim report sim syn
```

Fig. 6. List of available IP commands shown after introducing the *help* instruction.

```
Abeto > help dummyIP clean

Removes intermediate files
Usage: dummyIP clean [TARGET_DIR(optional)]
TARGET_DIR={all|synplify|dc|brave|sim|config|vivado}
           (default:all)
```

Fig. 7. *help* usage to show IP command details.

```
Abeto > help dummyIP config

Configures IP
Usage: dummyIP config [CFILE(optional)]
       [SCRIPT(optional)] [OUTDIR(optional)]
CFILE=tb_dummy_cfg.vhd (default:tb_dummy_cfg.vhd)
```

Fig. 8. *config* command details.

5.1. Help command

The *help* command can be used in several ways, providing information about built-in commands, available commands for a particular IP core and further details about them. In this case, the two latest options give important information about *dummyIP*. In the first place, it is possible to use it to show the list of available commands for this IP core passing its identifier as an argument, returning the output shown in Fig. 6.

Then, to get more details about each one of these commands, *help* command can be used by passing the IP identifier followed by the command name as arguments. For example, to ask Abeto how *clean* command shall be used and which are its functions, this variation of the command should be used, as reflected in Fig. 7. Firstly, the command function is described, whilst the information about its usage (command line) is shown then.

5.2. Configuration command (config)

To obtain a description and usage information about the configuration command of the *dummyIP*, *help* command can be used as described in previous subsection and obtaining the output shown in Fig. 8.

As it is observed, the *config* command provides a way to configure the IP, which is done through the Abeto built-in configuration tool (see Section 3.5). Optional arguments SCRIPT and OUTDIR allow to run the Abeto built-in configuration tool in batch mode, feature which is not covered in this tutorial. The target configuration file CFILE is an optional argument, although when omitted just the list of target configuration files is displayed. Configuration files are VHDL files which contain configuration parameters of the IP core. The *dummyIP* just includes one configuration file, as next shown in Fig. 9.

If just the configuration file is passed as argument to the *config* command, the Abeto built-in configuration tool is launched. First, the configuration tool parses the configuration file in order to extract the IP core configuration parameters, as long as they are declared as VHDL constants. If parsing succeeds, a welcome message is displayed and the application prompt changes to indicate that the built-in configuration tool has been entered. The new prompt also displays the name of the

```
Abeto > dummyIP config

*** Warning: target file missing
List of valid targets for command dummyIP config:
    tb_dummy_cfg.vhd
```

Fig. 9. *config* command output without arguments.

```
Abeto > dummyIP config tb_dummy_cfg.vhd

Launching configuration tool with file
src/tb/tb_dummy_cfg.vhd
Abeto configuration tool
Institute for Applied Microelectronics (IUMA)

Abeto config - dummyIP(tb_dummy_cfg.vhd) > help

Documented commands (type help <topic>):
========================================
close discard get help list load save set
```

Fig. 10. *config* command output with valid configuration file.

```
Abeto config - dummyIP(tb_dummy_cfg.vhd) > list

 List of configuration parameters:
   w (integer) : Counter width
   tclk (time) : Clock period
   post_syn (integer) : Enables the use of a
   synthesized model of the IP in simulations
```

Fig. 11. List of configuration parameters in the *dummyIP*.

```
Abeto config - dummyIP(tb_dummy_cfg.vhd) > get

    w = 3
    tclk = 10 ns
    post_syn = 0
```

Fig. 12. Use Case 1: initial parameter values.

IP (*dummyIP*) and the source file (*tb_dummy_cfg.vhd*) being configured. Using the *help* command again, the system will show the user which commands are available within the configuration tool. This is reflected in Fig. 10.

In the case of the *dummyIP*, its configuration parameters are the counter width (w), the clock period (tclk) and a flag to select between functional simulation or simulation with a synthesized netlist (post_syn). This information can be accessed through the *list* command, as shown in Fig. 11.

Two examples are provided to easen the understanding of how every configuration command works:

5.2.1. Case 1: Changing the counter width value to 7 and expand the clock period to 20 ns

Firstly, the current values for all parameters are checked by running the *get* command. This is reflected in Fig. 12.

Since the current configuration does not match the target one, the user runs *set* command to change parameter values, starting with the counter width. Then, to make sure Abeto is making the changes properly, the *get* command is invoked. In this case, two values are given for parameter w: the current value still appearing in the configuration file (3) and the new value to be applied (7). Changes are not applied until a *save* command is run. These steps are shown in Fig. 13.

```
Abeto config - dummyIP(tb_dummy_cfg.vhd) > set w 7
Configuration values updated

Abeto config - dummyIP(tb_dummy_cfg.vhd) > get
    w = 3 => 7
    tclk = 10 ns
    post_syn = 0

Abeto config - dummyIP(tb_dummy_cfg.vhd) > save
Target configuration file already exists.
Do you want to overwrite it? (y/n): y
 Configuration saved

Abeto config - dummyIP(tb_dummy_cfg.vhd) > get
    w = 7
    tclk = 10 ns
    post_syn = 0
```

Fig. 13. Use Case 1: changing and verifying the new value of the w parameter.

As the *save* command is executed without any arguments, changes are applied to the original configuration file, which is reflected in the configuration values obtained when calling again *get* after *save*. Alternatively, by specifying a path as a command argument, a new copy of the configuration file can be generated to store configuration changes at the designated path.

5.2.2. Case 2: Checking configuration values and changing the clock period value in case it is not 20 ns

The first step is to check the clock period value using *get* command and passing it tclk as an argument. As the reported value is not the expected one, it is replaced using the *set* command. However, before saving changes, the user finally decides to change the clock period to 15 ns. Therefore, the user discards changes using *discard* command, sets the new value and save changes. By using the *-force* flag with the *save* command, overwriting of existing files is forced without asking the user for permission. After that, the configuration tool can be closed. This process is summarized in Fig. 14.

5.3. Simulation commands (sim, ps-sim, par-sim)

As same as *config* command, the *help* command can be used to obtain a description and usage information about the simulation command of the *dummyIP*. The *dummyIP* simulation can be performed using either Questasim (vsim) or GHDL. Target simulation tool must be specified as command argument, as shown in Fig. 15.

The optional argument CSV_FILE allows to run an exhaustive verification campaign using the configurations defined in a CSV file. If provided, it must denote the path to an existing CSV file with the format defined in the *load* command from the Abeto built-in configuration tool. If such file exists, first the Abeto built-in configuration tool is invoked in batch mode to generate the configuration files corresponding to each test configuration defined in the CSV file, and then all test configurations are launched sequentially.

In addition to functional simulation, it is possible to run simulations with synthesized and place&routed models. With this aim, *ps-sim* and *par-sim* commands can be used, respectively.

The *ps-sim* command allows to generate post-synthesis models with either Synopsys Synplify or Xilinx Vivado tools, depending on the value of the argument SYNTH_TOOL. On the other hand, the *par-sim* command does not accept any arguments: synthesis and place&route is

```
Abeto config - dummyIP(tb_dummy_cfg.vhd) > get tclk
    tclk = 10 ns

Abeto config - dummyIP(tb_dummy_cfg.vhd) > set tclk
               20 ns
Configuration values updated

Abeto config - dummyIP(tb_dummy_cfg.vhd) > get tclk
    tclk = 10 ns => 20 ns

Abeto config - dummyIP(tb_dummy_cfg.vhd) > discard
Unsaved configuration values discarded

Abeto config - dummyIP(tb_dummy_cfg.vhd) > get tclk
    tclk = 10 ns

Abeto config - dummyIP(tb_dummy_cfg.vhd) > set tclk
               15 ns
Configuration values updated

Abeto config - dummyIP(tb_dummy_cfg.vhd) > get tclk
    tclk = 10 ns => 15 ns

Abeto config - dummyIP(tb_dummy_cfg.vhd) > save -force
Configuration saved

Abeto config - dummyIP(tb_dummy_cfg.vhd) > close
Closing Abeto configuration tool...
```

Fig. 14. Use Case 2: Checking, changing and discarding the value of the tclk parameter.

```
Abeto > help dummyIP sim
Simulates IP
Usage: dummyIP sim [TOOL] [CSV_FILE(optional)]
       TOOL={vsim|ghdl}

Abeto > dummyIP sim vsim

[...simulator console output...]

Simulation successful!!
# reading modelsim.ini
```

Fig. 15. Launching a *dummyIP* simulation using Questasim.

always performed with Xilinx Vivado (it is planned to extend the support to other EDA tools). Both commands operate in the same fashion: first, they perform synthesis of the IP core for a set of configurations defined in the CSV file with path syn/$tool/scripts/configurations.csv. Then, every synthesized model is simulated with Questasim, using the same CSV file to configure the testbench.

5.4. Synthesis command (syn)

Synthesis can be run with any of the following synthesis tools, selected by means of the TOOL argument: Synopsys Design Compiler (dc), Synopsys Synplify (synplify), NanoXplore NanoXmap (brave) and Xilinx Vivado. For the latter case, there are two options allowing to run just logic synthesis (vivado) or synthesis plus place&route (vivado-par). Compatibility with new NanoXplore Impulse tool is ensured, since most of the ACD and ADD file content already developed for NanoXmap can be reused.

For example, Fig. 16 shows how to run the synthesis using Synplify, after launching the *help* command to know the *syn* command options.

```
Abeto > help dummyIP syn
Synthesizes IP
Usage: dummyIP syn [TOOL] [CSV_FILE(optional)]
       TOOL={dc|synplify|brave|vivado|vivado-par}

Abeto > dummyIP syn synplify
```

Fig. 16. Launching a *dummyIP* synthesis using Synplify.

```
Abeto > help dummyIP report
Displays simulation/synthesis results
Usage: dummyIP report [FILE_PATH(optional)]

Abeto > dummyIP report
*** Warning: target file missing
List of valid targets for command dummyIP report:
{sim/reports/verification_report.txt|
syn/dc/command.log|
syn/dc/reports/check_design_dummyIP.log|
syn/dc/reports/synth_area_dummyIP.rpt|
syn/dc/reports/synth_cells_dummyIP.rpt|
syn/dc/reports/synth_power_dummyIP.rpt|
syn/dc/reports/synth_timing_dummyIP.rpt}

Abeto > dummyIP report syn/dc/reports/
        synth_area_dummyIP.rpt
```

Fig. 17. Opening a *dummyIP* synthesis report.

The optional argument CSV_FILE allows to synthesize the IP core for several configurations in an automated way. If provided, it must denote the path to an existing CSV file with the format defined in the *load* command from the Abeto built-in configuration tool. If such file exists it is parsed and synthesis is run multiple times, applying one of the configurations defined in the CSV file at a time. By now, this mode of operation is compatible just with the synthesis tools Synopsys Synplify and Xilinx Vivado.

5.5. Report command (report)

The *report* command can be used to show simulation and synthesis reports. The argument FILE_PATH is the path to the report file to be displayed, relative to the IP root. Unlike other command arguments, here the list of available target files cannot be provided through the *help* command, as it may dynamically change with simulation and synthesis runs. However, the list of available reports can be obtained by executing the *report* command without arguments.

Since the *dummyIP* follows the Abeto baseline IP directory structure (see Fig. 3), we can find several reports within *sim* and *syn* folders after running simulation or synthesis, respectively. For example, after running the synthesis using Design Compiler, several report files within the sim/dc/reports folder. And if we want to show the area report, we could use the *report* command as reflected in Fig. 17. This command opens the selected report using the default application associated to the report file extension.

6. Application to the ESA IP cores portfolio

Finally, Abeto is employed to organize, manage and launch a set of the ESA IP cores to demonstrate the strengths of this novel tool. ESA maintains a collection of soft IP cores concerning common functionality used in space applications and distributes them under licenses, with the aims of guaranteeing the availability of some key functional blocks, reducing costs of large developments and promoting the use

Table 1
Implementation results on BRAVE FPGA family.

IP	HurriCANe [19]	Spwb2-3 [20]	Spw-v12 [21]	LEON2FT [22]	SHyLoC [23]	SpaceFibre [24]	AHBR [25]	FTADDR [26]
Part	NG-MEDIUM				NG-LARGE			
Carry cells	157 (2%)	109 (2%)	100 (2%)	866 (11%)	3855 (12%)	516 (2%)	82 (1%)	196 (1%)
Registers	500 (2%)	301 (1%)	359 (2%)	12033 (38%)	3674 (3%)	1899 (2%)	607 (1%)	967 (1%)
BRAMs	0 (0%)	1 (2%)	0 (0%)	16 (29%)	97 (51%)	12 (7%)	0 (0%)	0 (0%)
DSPs	0 (0%)	0 (0%)	0 (0%)	0 (0%)	8 (3%)	0 (0%)	0 (0%)	0 (0%)
LUTs	742 (3%)	413 (2%)	385 (2%)	12410 (39%)	6943 (6%)	4933 (4%)	622 (1%)	1790 (2%)
Clk freq. (MHZ)	60.48	91.13	89.5	46.8	31.7	48.57	91.84	47.75

of standardized functions and protocols. Among the available IPs are communication bus controllers [19–21,24] and bridges [25], microprocessors [22], Single Event Effect (SEE) mitigation modules or data compression units [23].

However, there are not unified criteria on how to document and package these IPs and, consequently, scripts and commands used along the design flow greatly vary among them. Moreover, documented technology mapping results tend to become out-dated along time as new space-graded (i.e., robust against radiation effects) technologies emerge. From the potential users point of view, having access to up-to-date results (or the capability of generating them) is crucial to evaluate if an IP is adequate for the target application. Because of these reasons, the ESA IP cores portfolio constitutes a suitable use case for Abeto.

We have chosen a subset of 15 IPs from the ESA portfolio to integrate in the Abeto IP database. This subset demonstrates the strengths of Abeto to manage a quite heterogeneous database, which includes most of the designs commonly integrated on electronics on-board satellites. Some of these IPs are listed below, which are the ones whose results are summarized in Table 1:

- *HurriCANe*. Peripheral controller for CAN protocol.
- *Spwb2-3 and Spw-v12*. Two different implementations of the SpaceWire protocol [27].
- *LEON2FT*. Fault-tolerant SPARC microprocessor for space missions.
- *SHyLoC*. Hyperspectral image compressor.
- *SpaceFibre*. Controller for SpaceFibre protocol [28].
- *AHBR*. AMBA AHB to AMBA AHB bus bridge.
- *FTADDR*. Fault-tolerant DDR memory controller.

The integration process for every IP consisted in, first, studying the IP core documentation and packaging, with the aim of understanding how the workflow is implemented for the IP. Then, we have elaborated the corresponding IP description files (i.e., the ADD and ACD auxiliary files) to reflect the IP packaging organization and support as much as possible of the operation modes available for each IP core. Next, we have incorporated the IPs to the Abeto database and checked that every step in the IP workflow is successfully launched through Abeto. Finally, we have enabled the synthesis flow of every IP for NanoXplore NXmap tool and obtained implementation results through Abeto for the BRAVE family. BRAVE stands for “Big Re-Programmable Array for Versatile Environments” and it is a European initiative supported by ESA, CNES, and NanoXplore to develop a range of rad-hard, SRAM-based FPGAs. Three rad-hard devices are planned: NG-MEDIUM, NG-LARGE, and NG-ULTRA, with increasing fabric size and number of logic resources. A combination of radiation hardening by process layout, architecture (EDAC), and circuit design (TMR flip-flops and DMR clock-tree), together with a background scrubber to preserve the integrity of the internal configuration, are used to provide a rad-hard fabric [18]. Results for NanoXplore technology were not originally available for any of the IP cores under study. Results are provided for both NG-MEDIUM and NG-LARGE devices, depending on the IP demands in terms of resources utilization. It is worth to mention that, once the synthesis flow (or any other design flow step) has been enabled for any IP, it can be executed by just running a pair of commands in Abeto.

A representative subset of those mapping results is shown in Table 1. They have been generated using NXmap 3.9.0.5. Most of the ESA IP cores present several configuration options which affect both the occupation and performance results. In such cases a single representative configuration set has been chosen. In general terms, these results are consequent with those reported by each IP core in its associated documentation for other target FPGAs, taking into account the technological differences between them and the BRAVE FPGA family. This assessment provides the first IP benchmark available in the state-of-the-art on the recently developed BRAVE technology. Besides, mapping results have been obtained for other space-grade FPGA technologies. These results, which can be found in [29], can serve as a guide to potential users for evaluating the suitability of these IP cores for their application.

7. Conclusions

This work has presented Abeto, a software tool for IP workflow automation, intended to manage IP core databases in a unified way and to accelerate iterations of the IP cores design flow for benchmarking purposes. This tool aims at solving the lack of standardization in which steps of the IP design flow are implemented among different IP developers, with minimal modifications over the original IP structure. In order to do so, a pair of auxiliary files must be provided for each IP: a database definition and a command dictionary.

A tutorial is provided to ease the understanding of the different IP generic commands supported by the Abeto tool. This tutorial makes use of a simple IP core, denoted as *dummyIP*, which is provided together with the Abeto distribution. In addition, Abeto has been validated against a subset of the ESA portfolio of IP cores, which constitute an heterogeneous group of cores as result of different activities. This use case demonstrates the versatility of our tool. Furthermore, first mapping results available in the state-of-the-art are provided on the novel NanoXplore BRAVE FPGA family for a subset of the IP cores present in the ESA portfolio, providing a reference point for future space missions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work has been supported by the European Space Agency (ESA) through the contract 4000127452/19/NL/GLC/vr.

References

[1] I. Tuomi, The Future of Semiconductor Intellectual Property Architectural Blocks in Europe, JRC Scientific and Technical Reports, European Comission (JRC), 2009.

[2] Design&Reuse, 2023, URL https://www.design-reuse-embedded.com/.

[3] Synopsys IP, 2023, URL https://www.synopsys.com/designware-ip.html.

[4] Software & hardware open repository for embedded systems, 2014, URL http://www2.imse-cnm.csic.es/shores.

[5] OpenHW group, 2023, URL https://www.openhwgroup.org/.

[6] D. Gajski, A.-H. Wu, V. Chaiyakul, S. Mori, T. Nukiyama, P. Bricaud, Essential issues for IP reuse, in: Proceedings 2000. Design Automation Conference. (IEEE Cat. No.00CH37106), 2000, pp. 37–42, http://dx.doi.org/10.1109/ASPDAC.2000.835067.

[7] J. Haase, Design methodology for IP providers, in: Design, Automation and Test in Europe Conference and Exhibition, 1999. Proceedings (Cat. No. PR00078), 1999, pp. 728–732, http://dx.doi.org/10.1109/DATE.1999.761211.

[8] Accellera Systems Initiative, Accellera standards, 2022, URL https://www.accellera.org/downloads/standards.

[9] IEEE Computer Society, IEEE standard for IP-XACT, standard structure for packaging, integrating, and reusing IP within tool flows, in: IEEE Std 1685-2014 (Revision of IEEE Std 1685-2009), 2014, pp. 1–510, http://dx.doi.org/10.1109/IEEESTD.2014.6898803.

[10] Olof Kingdren, Invited paper: A scalable approach to IP management with FuseSoC, in: First Workshop on Open-Source Design Automation, OSDA, 2019, pp. 1–6, URL https://osda.gitlab.io/19/kindgren.pdf.

[11] K. Mohajerani, XEDA, 2020, URL https://pypi.org/project/xeda/.

[12] legoHDL, 2021, URL https://github.com/c-rus/legoHDL.

[13] A. Kamppi, L. Matilainen, J.-M. Määttä, E. Salminen, T.D. Hämäläinen, Extending IP-XACT to embedded system HW/SW integration, in: 2013 International Symposium on System on Chip, SoC, 2013, pp. 1–8, http://dx.doi.org/10.1109/ISSoC.2013.6675264.

[14] T. Schattkowsky, T. Xie, W. Mueller, A UML frontend for IP-XACT-based IP management, in: 2009 Design, Automation & Test in Europe Conference & Exhibition, 2009, pp. 238–243, http://dx.doi.org/10.1109/DATE.2009.5090664.

[15] F. Herrera, H. Posadas, E. Villar, D. Calvo, Enhanced IP-XACT platform descriptions for automatic generation from UML/MARTE of fast performance models for DSE, in: 2012 15th Euromicro Conference on Digital System Design, 2012, pp. 692–699, http://dx.doi.org/10.1109/DSD.2012.51.

[16] P. Lehmann, U. Martinez-Corral, EDA^2, 2016, URL https://github.com/edaa-org.

[17] European Space Agency (ESA), ESA IP Cores Website, 2022, URL http://www.esa.int/TEC/Microelectronics/SEMVWLV74TE_0.html.

[18] NanoXplore, From eFPGA cores to RHBD System-On-Chip FPGA, in: SEFUW: SpacE FPGA Users Workshop, 4th ed., 2018, pp. 1–53, URL https://indico.esa.int/event/232/contributions/2137/attachments/1820/2121/2018-04_NX-From_eFPGA_cores_to_RHBH_SoC_FPGAs-JLM-v2.pdf.

[19] European Space Agency (ESA), CAN IP, 2008, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/CAN.

[20] European Space Agency (ESA), SpaceWire-b, 2009, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/SpWb.

[21] European Space Agency (ESA), SpW-AMBA, 2003, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/SpW-AMBA.

[22] European Space Agency (ESA), LEON2-FT, 2015, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/LEON2-FT.

[23] European Space Agency (ESA), SHyLoC IP core, 2017, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/SHyLoC_IP_Core.

[24] European Space Agency (ESA), SpaceFibre port IP core, 2019, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/SpaceFibre_Port_IP_Core.

[25] European Space Agency (ESA), AHBR, 2008, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/AHBR.

[26] European Space Agency (ESA), Fault tolerant DDR controller, 2020, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/Fault_Tolerant_DDR_Controller_FTADDR.

[27] European Cooperation for Space Standardization, SpaceWire – Links, Nodes, Routers and Networks (ECSS-E-ST-50-12C Rev.1), ECSS, 2019.

[28] European Cooperation for Space Standardization, SpaceFibre – Very High-Speed Serial Link (ECSS-E-ST-50-11C), ECSS, 2019.

[29] European Space Agency (ESA), Automated BEnchmarking TOol (ABETO) for ESA IP Cores, 2022, URL https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Microelectronics/Automated_BEnchmarking_TOol_ABETO_for_ESA_IP_Cores.

Antonio J. Sánchez is graduated on Industrial Engineering by Universidad Carlos III de Madrid (UC3M) in 2011. Then he joined the Microelectronic Design and Applications research group in the Electronic Technology Department of the UC3M, where he worked as assistant researcher. During this period, he awarded his Master on Advanced Electronic Systems in 2013 and his Ph.D. degree on Electric, Electronic and Automatic Engineering in 2017, both by the UC3M. Currently he is a member of the Integrated Systems Design Division of the Institute for Applied Microelectronics (IUMA) in the University of Las Palmas de Gran Canaria. His research at IUMA is focused on the hardware implementation of algorithms for hyperspectral image processing for space applicationsHe was a Visiting Researcher with the European Space Research and Technology Centre, The Netherlands. Additionally, his research interests include fault tolerant hardware design, approximate computing, and formal verification methods.

Yubal Barrios was born in Las Palmas de Gran Canaria, Spain, in 1993. He received the Telecommunications Engineering degree by the University of Las Palmas de Gran Canaria in 2016. He obtained the MSc. and the Ph.D. in Telecommunications Technologies in 2017 and 2022, respectively, by the same University. He has been funded by the Institute for Applied Microelectronics (IUMA) since 2017, where he has conducted his research activities at the Integrated Systems Design Division in the context of hardware implementations for hyperspectral image compression on FPGAs and MPSoCs. In 2019, he was invited as Visiting Researcher by the Microelectronics Section of the European Space Research and Technology Centre (ESTEC), core of the European Agency (ESA) located in Noordwijk, the Netherlands. His current research interests include the development of efficient algorithms for on-board hyperspectral image compression and reconfigurable hardware architectures optimized in terms of throughput, memory usage and power consumption. He has co-authored several scientific papers published in specialized journals and international conferences.

Lucana Santos received the Telecommunications Engineering degree from the University of Las Palmas de Gran Canaria, in 2008, and the Ph.D. degree from the Integrated System Design Division, IUMA, in 2014. She was a Visiting Researcher with the European Space Research and Technology Centre, The Netherlands. She has participated actively in industrial projects in the field of hardware architectures for hyperspectral and multispectral image compression on GPUs and FPGAs for Thales Alenia Space España and the European Space Agency. Since 2018, she has been with the Data Systems and Microelectronics Division at the European Space Agency. She is currently a member of the CCSDS Multispectral/Hyperspectral Data Compression working group. She has co-authored several scientific papers and has been a Reviewer of major international journals in her research areas. Her current research interests include hardware architectures for on-board data processing, reconfigurable architectures, and hardware/software co-design methodologies.

Roberto Sarmiento is Full-Professor at the Electronics and Telecommunication Engineering School at University of Las Palmas de Gran Canaria, Spain, in the area of Electronic Engineering. He contributed to set this school up, he was the Dean of the Faculty from 1994 to 1998 and Vice-Chancellor for Academic Affairs and Staff at the ULPGC from 1998 to 2003. He is a co-founder of the Research Institute for Applied Microelectronics (IUMA) and Director of the Integrated Systems Design Division of this Institute. He has published more than 90 journal papers and more than 160 conference papers. Roberto Sarmiento has been awarded with five six years research periods by the National Agency for the Research Activity Evaluation in Spain. He has participated in more than 60 projects and research programmes funded by public and private organizations. His current research interest is related to electronics system on-board satellites.