14

Design Methodology with System Generator in Simulink of a FHSS Transceiver on FPGA

Santiago T. Pérez[1], Carlos M. Travieso[1],
Jesús B. Alonso[1] and José L. Vásquez[2]
[1]*Signals and Communications Department,*
University of Las Palmas de Gran Canaria
[2]*Department of Computer Science, University of Costa Rica*
[1]*Spain*
[2]*Costa Rica*

1. Introduction

This study aims to describe a design method for Field Programmable Gate Array (FPGA) (Maxfield, 2004) applied, in particular, to the design of a Frequency Hopping Spread Spectrum (FHSS) transceiver (Simon et al., 1994). Simulink (MathWorks, 2011) is a tool integrated in Matlab, which allows the design of systems using block diagrams in a fast and flexible way. Xilinx is one of the most important FPGA manufacturers and provides System Generator (Xilinx, 2011), it is a design environment over Simulink for FPGA based on the method described. The design is based on a previous FHSS transceiver designed for indoor wireless optical communications made with discrete components (Pérez et al., 2003). One of the improvements in the proposed system is the physical integration.

2. The physical device

Initially, there were several alternatives for the system hardware. In principle, an Application Specific Integrated Circuit (ASIC) can be used (Maxfield, 2004), but to configure these devices must be sent to the manufacturer, which increases development time and makes more expensive the prototype. This technology achieves good physical performances: low area, low power consumption and minimal delays.

At the other extreme Digital Signal Processor (DSP) can be used which are very cheap (Maxfield, 2004). The DSPs do not have the best physical performances; normally they occupy maximum area, have high power consumption and maximum delay. In fact, when the volume of calculus is high, easily they do not have real time response. This is because the architecture is rigid, in both data and operations formats.

In the middle are FPGA, which have a reasonable cost for the design of prototypes; in general an intermediate cost to the two previous cases. The FPGA have significant physical benefits, without reaching the performances of ASIC. FPGAs have benefits outweigh the

DSP because the FPGA final architecture can be configured in a fully flexible way, in both data and operations size.
It must be emphasized that FPGA are integrated circuits reprogrammable by the designer and can be used for different projects, or in a project during its different phases. The FPGAs are also available in the market on printed circuit boards, with power and programming connectors, auxiliary memories and input-output pins; this avoid to design and construct the printed circuit board, and makes it ideal for prototyping design.

3. Design methodology

A transceiver can be designed using discrete electronic components. In general, the overall design is not flexible and highly dependent on technology and available devices, has long design time, occupies large area, has high power consumption and high delays and low maximum operating frequency.
In general, the trend is to integrate the design in a digital integrated circuit and place around the necessary external components; this eliminates the previous inconveniences. It must be emphasized that these designs can be easily portable between devices, even from different manufacturers. This portability is possible because the design can be described with a standard hardware description language (HDL).
In digital systems, when floating point arithmetic is used, the range and precision can be adjusted with the number of bits of exponent and mantissa, it is then possible to obtain a wide range and high precision in this type of representation. However, floating point operations require many hardware resources and long time execution (Hauck & DeHon, 2008). On the other hand, the fixed point arithmetic requires fewer hardware resources, but the range and precision can be improved only by increasing the number of bits. If the number of bits is constant, to increase the range causes a decrease in the precision. It is possible to use fixed point arithmetic in most applications when the range of signals is known or can be determined by statistical methods. In fixed point arithmetic the 2's complement representation is used because its arithmetic rules are simpler than the 1's complement representation.
Ordinarily the systems can be designed using a standard hardware description language: VHDL (Very High Speed Integrated Circuit Hardware Description Language) (Pedroni, 2004) or Verilog (Palnitkar, 2003). Manual coded of complex systems using one of these languages is little flexible and has a great design time. To solve these problems several design programs have been developed. One of them is the System Generator from Xilinx, which is installed in Simulink.

3.1 System generator

When System Generator is installed some Blocksets (Fig. 1) are included in Simulink of Matlab. Each block is configured after opening its dialog window, this permits fast and flexible designs. Basically, System Generator allows minimizing the time spent by the designer for the description and simulation of the circuit. On the other hand, the design is flexible; it is possible to change the design parameters and check quickly the effect on the performances and the architecture of the system. The functional simulation is possible even before the compilation of the model designed. The compilation generates the files of the structural description of the system in a standard hardware description language for the Integrated System Environment (ISE) for Xilinx FPGAs.

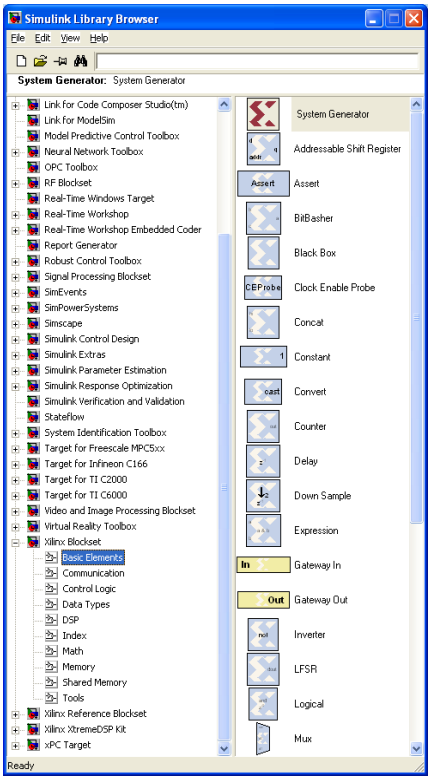

Fig. 1. System Generator Blocksets in Simulink

The FPGA boundary in the Simulink model is defined by Gateway In and Gateway Out blocks. The Gateway In block converts the Simulink floating point input to a fixed point format, saturation and rounding modes can be defined by the designer. The Gateway Out block converts the FPGA fixed point format to Simulink double numerical precision floating point format.

In the System Generator the designer does not perceive the signals as bits; instead, the bits are grouped in signed or unsigned fixed point format. The operators force signals to change automatically to the appropriate format in the outputs. A block is not a hardware circuit necessarily; it relates with others blocks to generate the appropriate hardware. The designer can include blocks described in a hardware description language, finite state machine flow diagram, Matlab files, etc. The System Generator simulations are bit and cycle accurate, this means results seen in a simulation exactly match the results that are seen in hardware. The Simulink signals are shown as floating point values, which makes easier to interpret them. The System Generator simulations are faster than traditional hardware description language simulators, and the results are easier for analyzing. Otherwise, the VHDL and Verilog code are not portable to other FPGA manufacturers. The reason is that System Generator uses Xilinx primitives which take advantages of the device characteristics.

System Generator can be used for algorithm exploration or design prototyping, for estimating the hardware cost and performance of the design. Other possibility is using

System Generator for designing a portion of a big system and joining with the rest of the design. Finally, System Generator can implement a complete design in a hardware description language. Designs in System Generator are discrete time systems; the signals and blocks generate automatically the sample rate. However, a few blocks set the sample rate implicitly or explicitly. System Generator supports multirate circuits and some blocks can be used for changing the sample rate.
Often an executable specification file is created using the standard Simulink Blocksets (see Fig. 2). The specification file can be designed using floating point numerical precision and not hardware detail. Once the functionality and basic dataflow have been defined, System Generator can be used to specify the hardware implementation details for the Xilinx devices. System Generator uses the Xilinx DSP Blockset from Simulink and will automatically invoke Xilinx Core Generator to generate highly optimized netlists for the building blocks. System Generator can execute all the downstream implementation tools to get a bitstream file for programming the FPGA device. An optional testbench can be created using test vectors extracted from the Simulink environment for using with Integrated System Environment simulators.

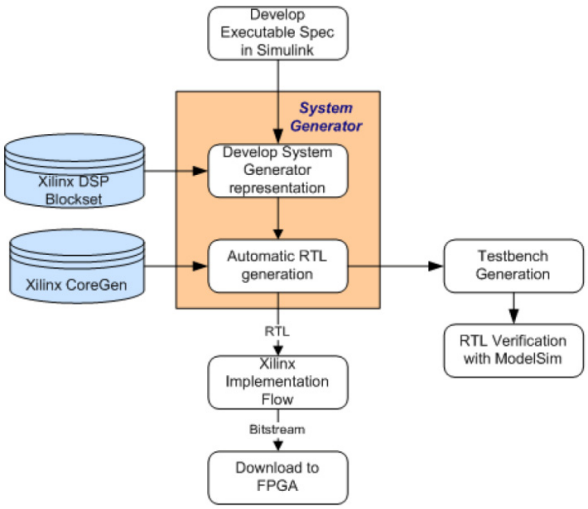

Fig. 2. System Generator design flow (download from www.xilinx.com)

Every system designed with System Generator must contain a System Generator block (Fig. 3); this block specifies how simulation and code generator can be used. Firstly, the type of compilation in the System Generator block can be specified to obtain: HDL netlist, Bitstream for programming, etc. Secondly, the FPGA type can be chosen. The target directory defines where the compilation writes the files of Integrated System Environment project. The synthesis tool specifies which tool is chosen for synthesizing the circuit: Synplify, Synplify Pro or Xilinx Synthesis Tool (XST). In the hardware description language the designer can choose between VHDL and Verilog. Finally, clock options defines the period of the clock, its input pin location, the mode of multirate implementation and the Simulink system period, which is the greatest common divisor of the sample periods that appear in the system. In the block icon display, the type of information to be displayed is specified.

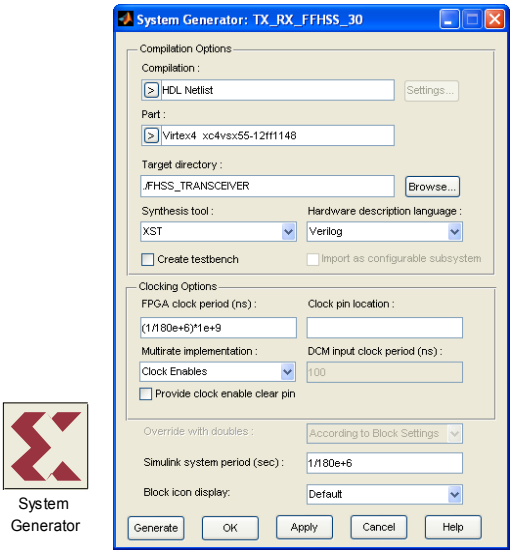

Fig. 3. System Generator block and its dialog window

3.2 Integrated system environment

In Xilinx Integrated System Environment it is possible to compile the hardware description language files, and simulate the system behavioral or timing analysis. Also the occupancy rate, power consumption and operating temperature of the FPGA are obtained. Afterwards the program file can be generated for the chosen device; this file can be downloaded from the computer to the board where the FPGA is included. Finally, the performance of the design system must be checked with electronic measure equipment.

When the designer clicks on Generate in dialog window of System Generator block, the structural description files in a hardware description language are obtained, and a project is created for Integrated System Environment. Now it is possible to check the syntax of the hardware description language files (Fig. 4). The first step in the compilation process is synthesizing the system. The synthesis tool used is Xilinx Synthesis Tool, it is an application that synthesizes hardware description language designs to create Xilinx specific netlist files called NGC (Native Generic Circuit) files. The NGC file is a netlist that contains both logical design data and constraints. The NGC file takes the place of both Electronic Data Interchange Format (EDIF) and Netlist Constraints File (NCF) files. In synthesis options optimization goal for area or speed can be fixed; by default, this property is set to speed optimization. Similarly, optimization effort can be established as normal or high effort; in the last case additional optimizations are performed to get best result for the target FPGA device. Synthesis report can be analyzed by the designer; moreover, the designer can view Register Transfer Level (RTL) schematic or technology schematic. After synthesizing the system, the design is implemented in four stages: translate, map, place and route. The translation process merges all the input netlists and design constraint information and outputs a Xilinx Native Generic Database (NGD) file. Then the output NGD file can be mapped to the targeted FPGA device family. The map process takes the NGD file, runs a design rule checker and maps the logic design to a Xilinx FPGA device. The result appears

in a Native Circuit Design (NCD) file, which is used for placing and routing. The place and route process takes a NCD file and produces a new NCD file to be used by the programming file generator. The generator programming file process runs the Xilinx bitstream generation program BitGen to produce a bit file for Xilinx device configuration. Finally, the configuration target device process uses the bit file to configure the FPGA target device. Behavioral simulations are possible in the design before synthesis with the simulate behavioral model process. This first pass simulation is typically performed to verify the Register Transfer Level or behavioral code and to confirm the designed function. Otherwise, after the design is placed and routed on the chip, timing simulations are possible. This process uses the post place and route simulation model and a Standard Delay Format (SDF) file. The SDF file contains true timing delay information of the design.

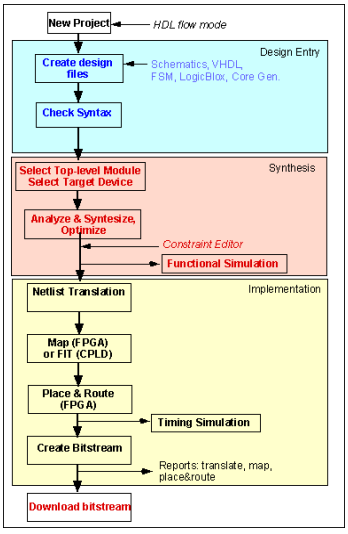

Fig. 4. Overview of design flow of Integrated System Environment (download from www.xilinx.com)

4. The transceiver

This chapter is based on a previous FHSS transceiver (Fig. 5) for wireless optical communications. The FHSS and analog synchronization signals were emitted by two separated Light Emitting Diodes (LED) to avoid adding them with discrete analog circuits.

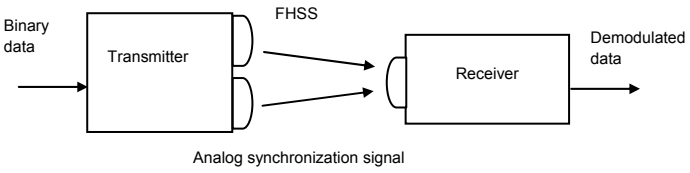

Fig. 5. Block diagram with FHSS transceiver designed previously

The core of the transmitter was a discrete Direct Digital Synthesizer (DDS) AD9851 from Analog Devices (Analog Devices, 2011). The discrete DDS (Fig. 6) is a digital system excepting the final Digital to Analog Converter; its output signal is a sinusoidal sampled signal at 180 MHz. The emitted FHSS signal was smoothed by the 100 MHz bandwidth of the optical emitter. In the DDS used, the output frequency is fixed by the expression (1), where f_{DDS_CLK} is the frequency of the DDS clock (180 MHz), N is the number of bits of the tuning word (32 bits) and Word is the decimal value of 32 bit frequency tuning word.

$$f_{out}=(Word \cdot f_{DDS\_CLK})/2^N \quad (1)$$

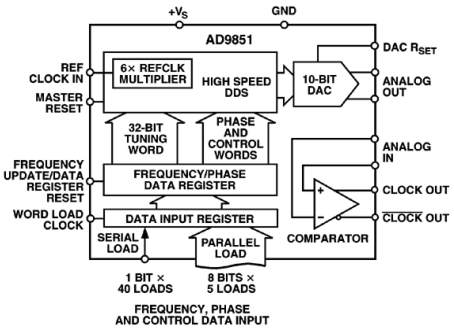

Fig. 6. Block diagram of discrete DDS AD9851 from Analog Devices (download from www.analog.com)

In the demodulator of the receiver, two similar discrete DDS were used as local oscillators. In the new design, the full transceiver with the previous methodology is described. The modulator matches with the transmitter designed previously, excepting the optical emitters and the output Digital to Analog Converter of the discrete DDS. In the same way, the two DDS in the demodulator were integrated in the FPGA. In the previous design, discrete analog filters were used. In the new design, these filters were integrated in the FPGA as digital filters. The new design methodology is improved by DDS block and filter design capabilities in System Generator. The Fig. 7 shows the new FHSS transceiver. In Fig. 8, the data in the transmitter and the demodulated data are shown. After the synchronization is reached in the receiver, the demodulation is executed perfectly.

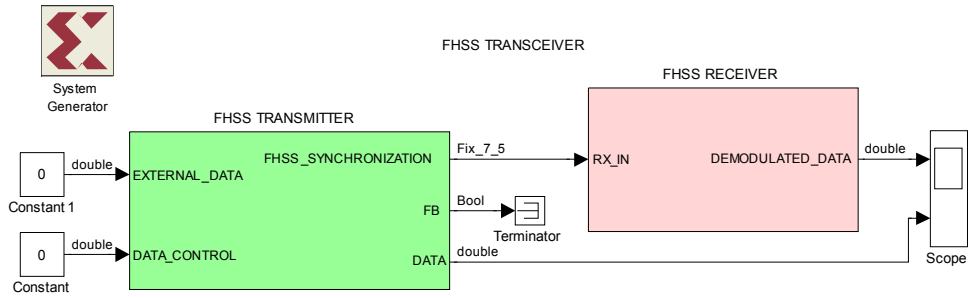

Fig. 7. Frequency Hopping Spread Spectrum transceiver

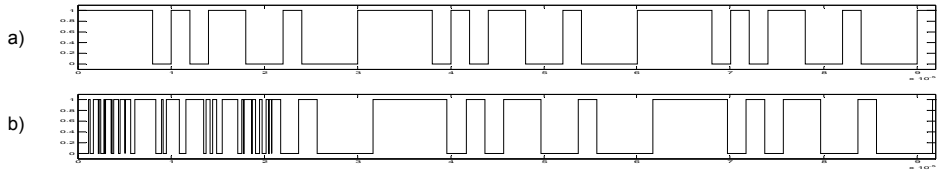

Fig. 8. Data signals in the transceiver: a) transmitted data, b) demodulated data

If Port Data Type is enabled in Simulink, after the system simulation the data types are shown in every point of the design. It can be: Bool (boolean); double, Simulink floating point format; UFix_m_n, unsigned m bits two's complement fixed point format with n fractional bits; Fix_m_n, signed m bits two's complement fixed point format with n fractional bits. Otherwise, the signals can be analyzed in different ways using Simulink Sinks blockset. First, the Scope block can be used; this was the method used for adjusting the transceiver, it is quick but not convenient for capturing signals. Secondly, signals can be captured with the To Workspace block, but these signals are only stored temporarily in Matlab. Finally, To File block keeps the captured signals in a mat file permanently; for this reason To File block was used to capture and present simulations of this design.

5. The transmitter

The block diagram of the designed transmitter is drawn in Fig. 9. It is composed of an internal data generator, a pseudorandom code generator, and two DDS, used to generate the FHSS and synchronization signals. An external clock of 180 MHz is needed for the system. In this transmitter it is possible to choose between internal or external binary data.

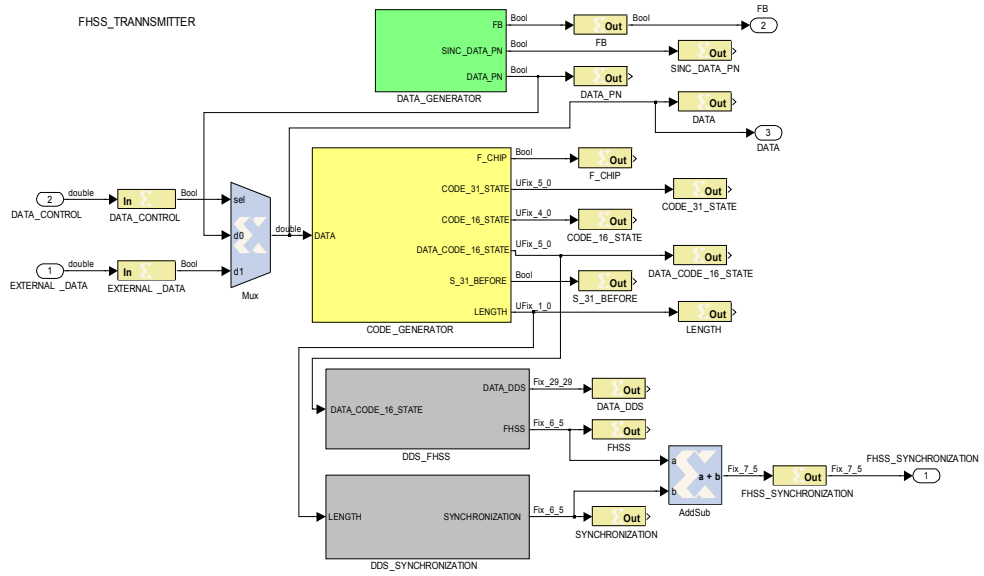

Fig. 9. Block diagram of FHSS transmitter designed with System Generator

5.1 Pseudorandom data generator

Application of the internal data generator (Fig. 10) avoids using an external data source; it was designed using a Linear Feedback Shift Register (LFSR) block as pseudorandom generator of 15 bits long at 500 kilobits per second. A pulse in the pseudorandom data generator is formed each time the sequence begins; this provides a high quality periodic signal to synchronize the oscilloscope. The LFSR block is configured with the dialog windows (Fig. 11). The clock, the data synchronization pulse and the pseudorandom data are shown in Fig. 12.

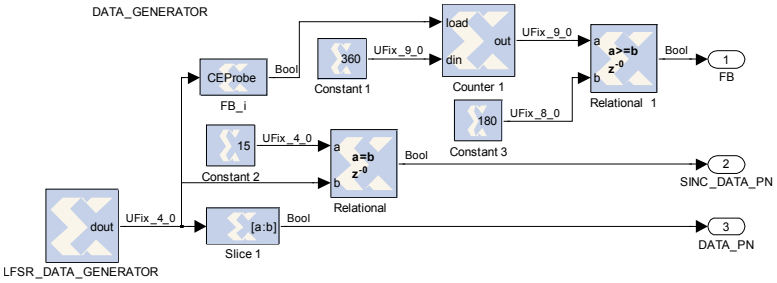

Fig. 10. Internal pseudorandom data generator

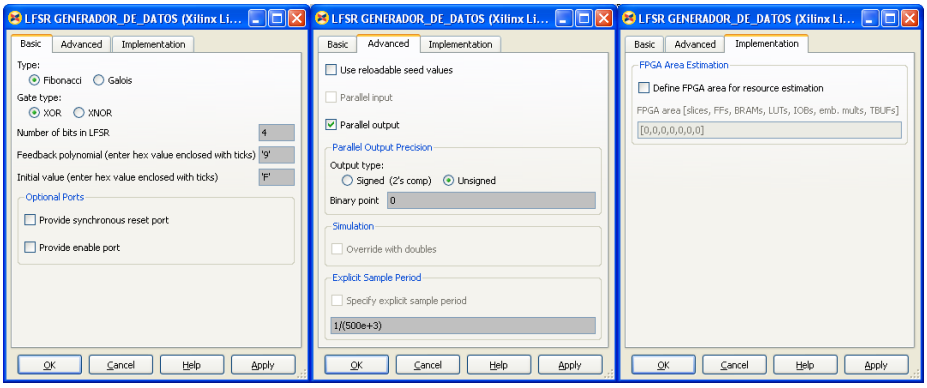

Fig. 11. Linear Feedback Shift Register dialog windows

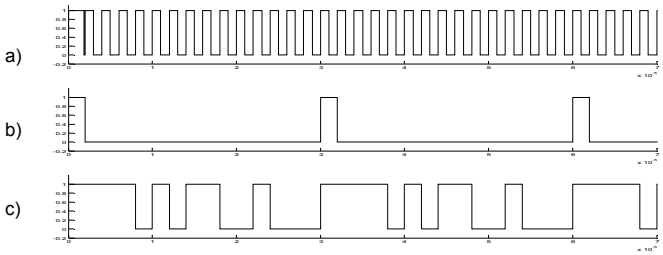

Fig. 12. Pseudorandom data generator signals: a) clock at bit rate, b) the data synchronization pulse, c) the pseudorandom binary data

5.2 Pseudorandom code generator

The pseudorandom code generator and its Simulink simulation signals are shown in Figures 13 and 14. The code rate is called chip frequency; its value is 1.5 Megachips per second. Consequently, three codes are generated by each data bit. The code generator is based on a Linear Feedback Shift Register of 31 states. In the pseudorandom code generator, a pulse is generated each time the sequence begins. A five bits word is obtained with the four most significant bits of the pseudorandom code generator and the data bit as most significant bit.

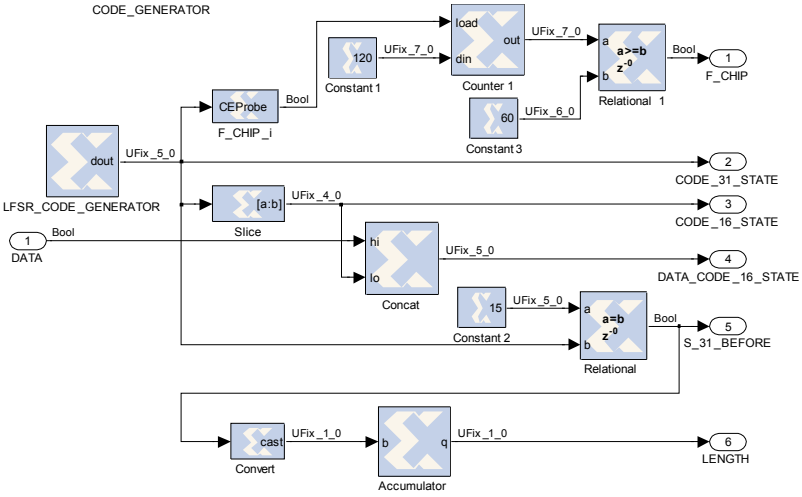

Fig. 13. Pseudorandom code generator

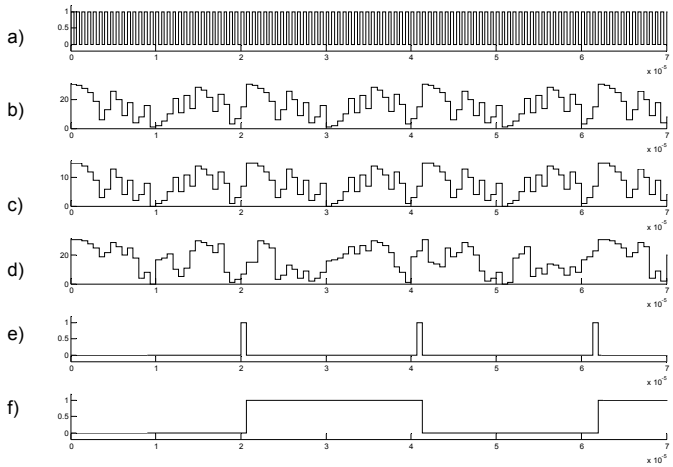

Fig. 14. Pseudorandom code generator signals: a) chip frequency, b) pseudorandom code 5 bits width, c) 4 most significant bits of pseudorandom code 5 bits width, d) data joined with 4 most significant bits, e) the stage previous to "11111", f) square signal which marks the code length

5.3 Frequency hopping spread spectrum signal generation

For each group of five bits (signal d in Fig. 14) a sampled sinusoidal signal is generated according to Table 1.

Code	Frequency (MHz)	Code	Frequency (MHz)
00000	24.384	10000	48.960
00001	25.920	10001	50.496
00010	27.456	10010	52.032
00011	28.992	10011	53.568
00100	30.528	10100	55.104
00101	32.064	10101	56.640
00110	33.600	10110	58.176
00111	35.136	10111	59.712
01000	36.672	11000	61.248
01001	38.208	11001	62.784
01010	39.744	11010	64.320
01011	41.280	11011	65.856
01100	42.816	11100	67.392
01101	44.352	11101	68.928
01110	45.888	11110	70.464
01111	47.424	11111	72.000

Table 1. Transmitted frequencies for the FHSS signal

In Fig. 15, the DDS generating the FHSS signal is shown. The DDS clock is the system clock (180 MHz). Therefore, a pure sinusoidal signal with an external filter can be synthesized until a bit less than 90 MHz.

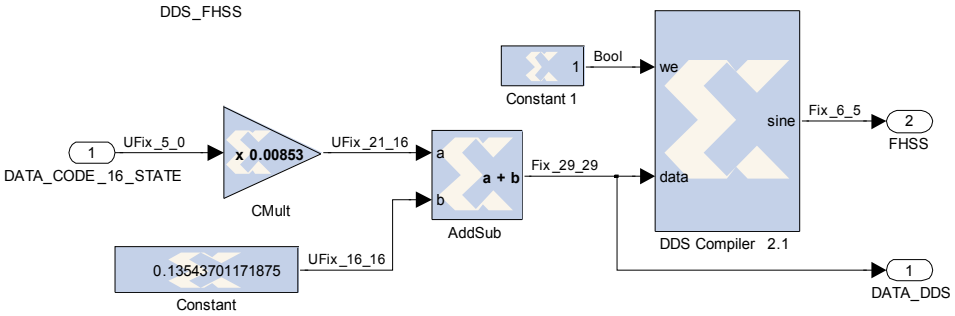

Fig. 15. Direct Digital Synthesizer generating the FHSS signal

The input data for the Xilinx DDS block is the synthesized frequency divided by the DDS clock. The equation (2) shows the meaning of this relation. Consequently, the DDS block

fixes the number of N bits according to the rest of the DDS parameters: spurious free dynamic range, resolution, implementation mode, etc.

$$data=f_{out}/f_{DDS\_CLK}=Word/2^N \quad (2)$$

Fig. 16 shows the dialog windows of the DDS block, where the designer can fix its parameters. This DDS acts like a frequency modulator.

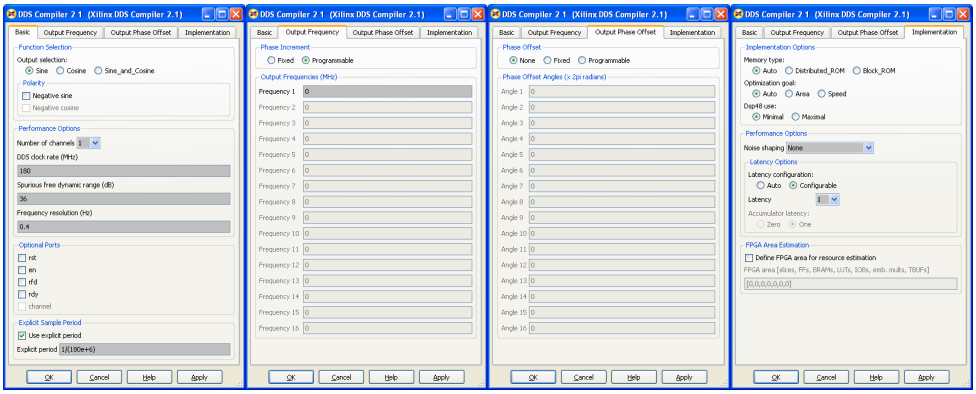

Fig. 16. Direct Digital Synthesizer block dialog windows for FHSS signal

The five bits input signal is transformed to the format of the input DDS block. The last operation is an unsigned fixed point integer to unsigned fixed point decimal conversion. In Fig. 17, five chip times of FHSS signal are shown. Three frequencies are generated by each data bit, therefore this is a Fast Frequency Hopping Spread Spectrum modulation.

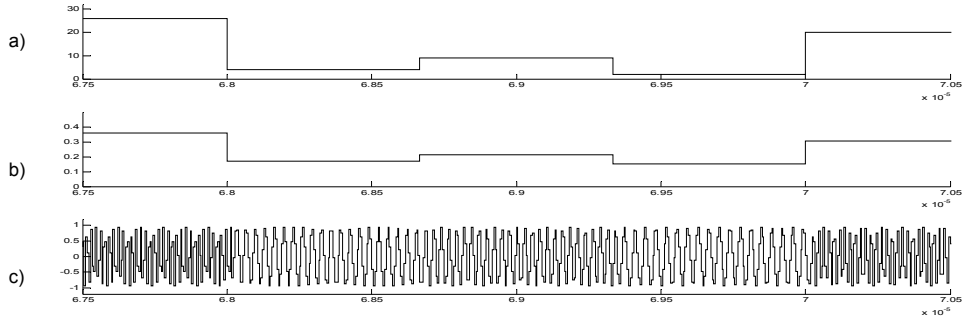

Fig. 17. Signals in Direct Digital Synthesizer generating the FHSS signal: a) five bits DDS input, b) input for Xilinx DDS block, c) FHSS signal

5.4 Synchronization signal generation and final adder

In the pseudorandom code generator, a square signal is generated with a 50% duty cycle (signal f in Fig. 14). This square signal has a semi-period with the same duration as the pseudorandom code length. The square signal is the DDS input (Fig. 18), it modulates in phase to a 9 MHz carrier (Fig. 19). The phase modulated signal carries information about the

beginning of the pseudorandom code; and about its chip frequency, because its carrier is a multiple of 1.5 MHz.

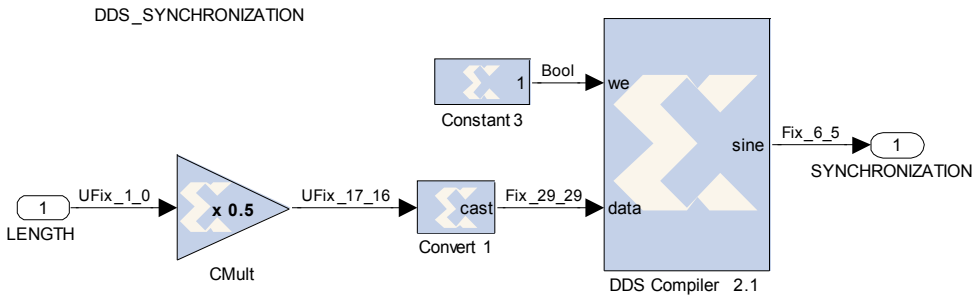

Fig. 18. Direct Digital Synthesizer for synchronization generation

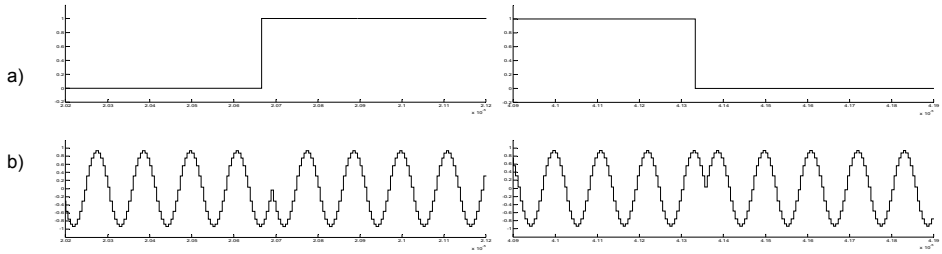

Fig. 19. Signals in Direct Digital Synthesizer that generates the synchronization signal: a) square input signal, b) synchronization signal

The Fig. 20 shows the dialog window of the DDS block. This Direct Digital Synthesizer acts like a phase modulator. In both Xilinx DDS blocks, the latency configuration is fixed to 1 for keeping the DDS delays to the minimum same value, this parameter specifies the delay as number of clock cycles.

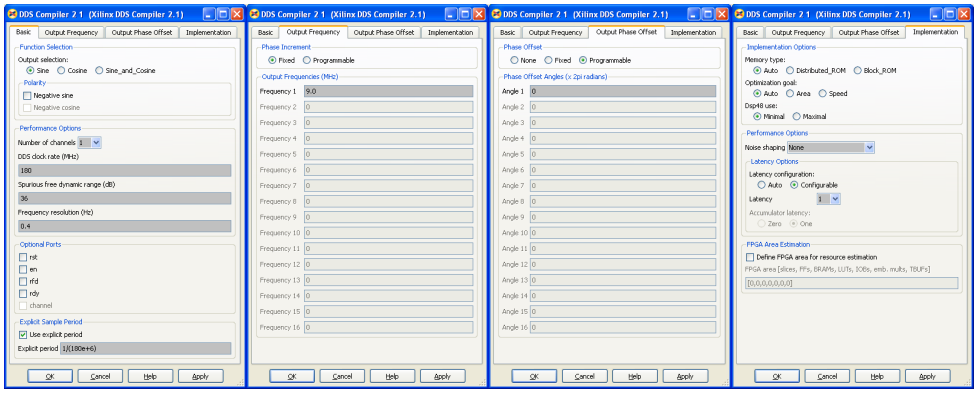

Fig. 20. Direct Digital Synthesizer block dialog windows for synchronization signal

Finally, the FHSS and the synchronization signals are added with an AddSub block, this new signal is the transmitter output (Fig. 21).

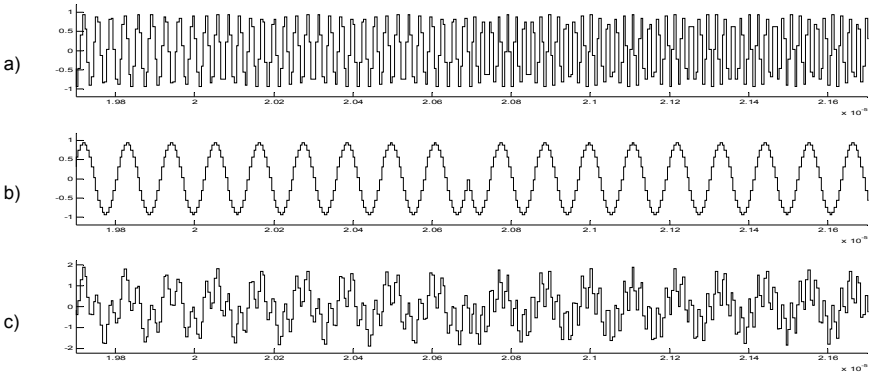

Fig. 21. Inputs and output of final adder: a) FHSS signal, b) synchronization signal, c) the above signals added together

6. The receiver

The receiver block diagram is shown in Fig. 22. The signal received from the transmitter enters in the splitting filter, FHSS and synchronization signals can be separated because they are multiplexed in frequency. The filtered synchronization signal is the input of the synchronization recovery, where the code is obtained in the receiver. The code recovered synchronizes the local oscillators. Finally, the local oscillators outputs and the FHSS filtered are introduced to the double branch data demodulator.

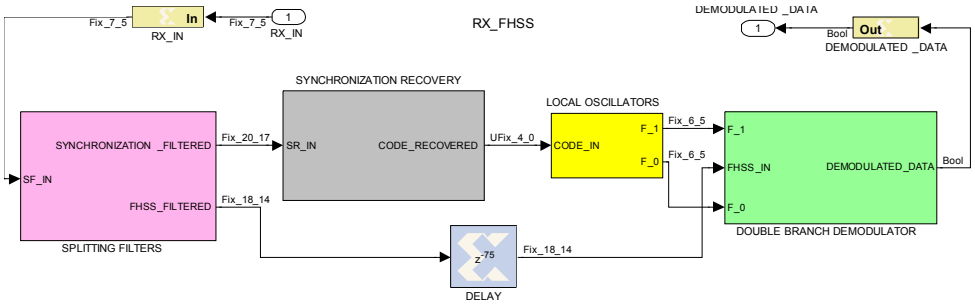

Fig. 22. Block diagram of FHSS receiver designed with System Generator

6.1 Splitting filters

The splitting filters block diagram and signals are drawn in Fig. 23 and 24 respectively. A Finite Impulse Response (FIR) high pass filter recovers the FHSS signal. It was designed using the Filter Design and Analysis Tool (Fig. 25), the filter's coefficients are used by Xilinx FIR Compiler block for being synthesized. In the same way, a band pass filter is designed to obtain the synchronization signal.

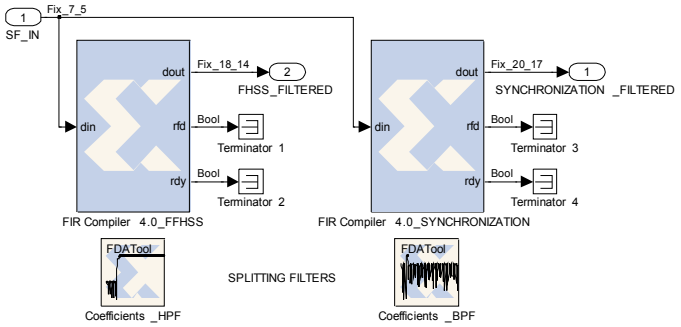

Fig. 23. Splitting filters block diagram

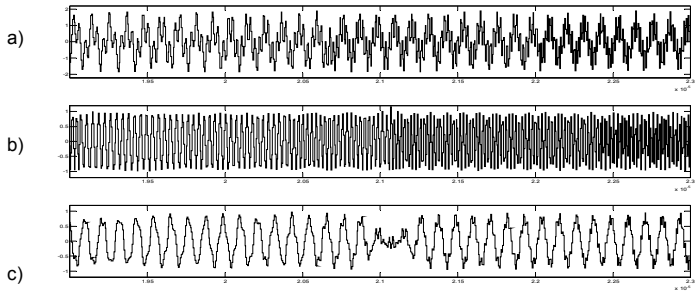

Fig. 24. Splitting filters signals: a) input, b) FHSS filtered, c) synchronization filtered

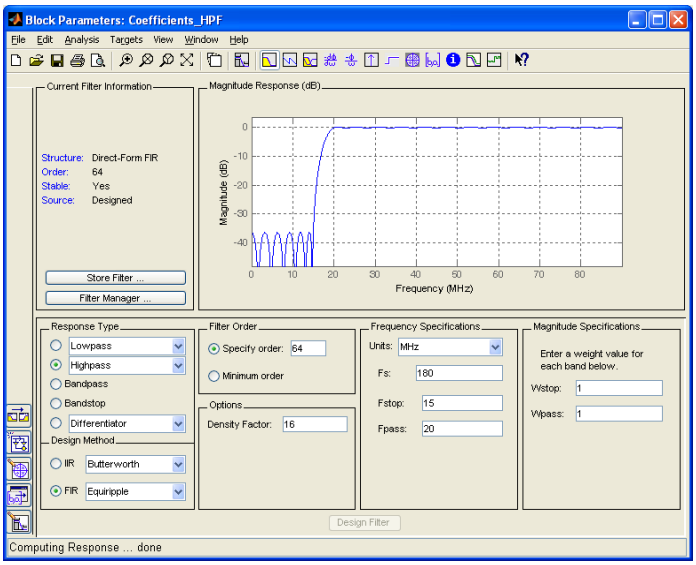

Fig. 25. Filter Design and Analysis Tool dialog window

6.2 Synchronization recovery

The input of this system is the synchronization filtered, in its output gets the most significant four bits of the pseudorandom code (Fig. 26). It is formed (Fig. 27) by a 9 MHz recover, a synchronous demodulator, a load and enable generators, and a Linear Feedback Shift Register code generator.

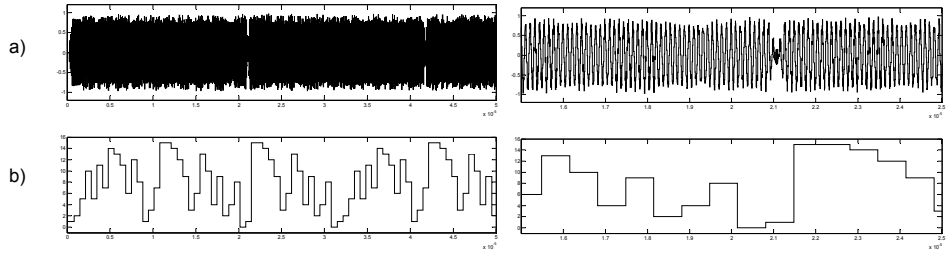

Fig. 26. Synchronization recovery signals: a) synchronization filtered, b) code recovered

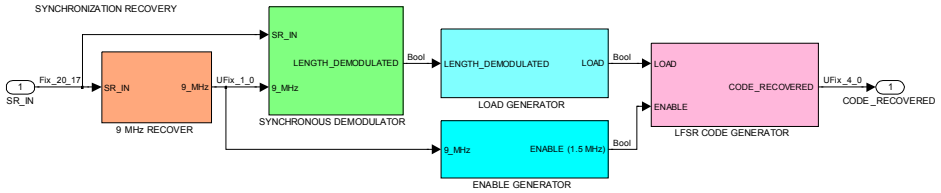

Fig. 27. Synchronization recovery block diagram

6.2.1 Carrier recover (9 MHz)

This system recovers the carrier of the synchronization signal (Fig. 28). Initially the phase-modulated signal is squared and filtered to get double the carrier frequency with an 18 MHz band pass filter (Fig. 29); the sample frequency is 180 MHz. The 18 MHz signal is squared by a comparator and a pulse is generated with each rising edge. Finally, an accumulator generates a 9 MHz squared signal with 50% duty cycle.

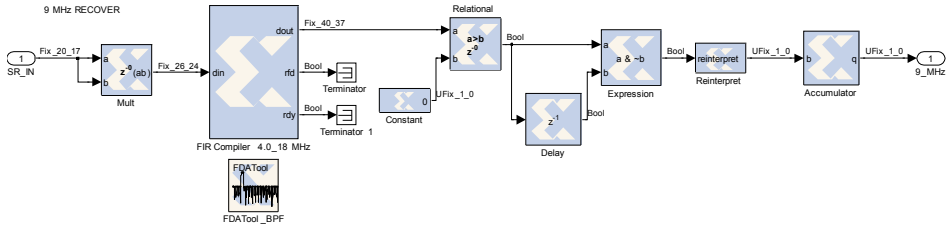

Fig. 28. Carrier recovery of 9 MHz block diagram

6.2.2 Synchronous demodulator

The block in Fig. 30 is a phase demodulator of the synchronization signal. The output indicates the length of the code with two consecutive edges of the signal (Fig. 31). The

unipolar square 9 MHz carrier is converted to bipolar; in this way, the multiplier output assumes non-zero values in each semicycle. The delay block for the carrier ensures the synchronous demodulation. The output of the low pass filter is introduced to a comparator to get the length signal demodulated.

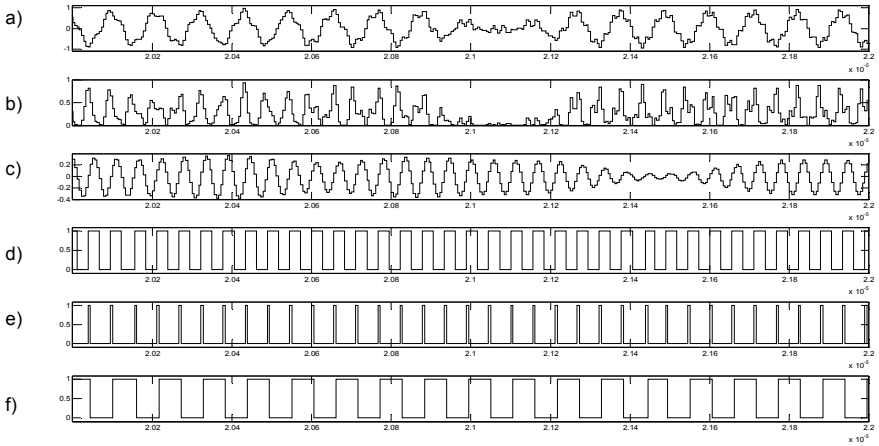

Fig. 29. Carrier recovery signals: a) synchronization filtered input, b) squared signal, c) 18 MHz filtered, d) 18 MHz square wave, e) pulse with rising edge, f) 9 MHz square wave

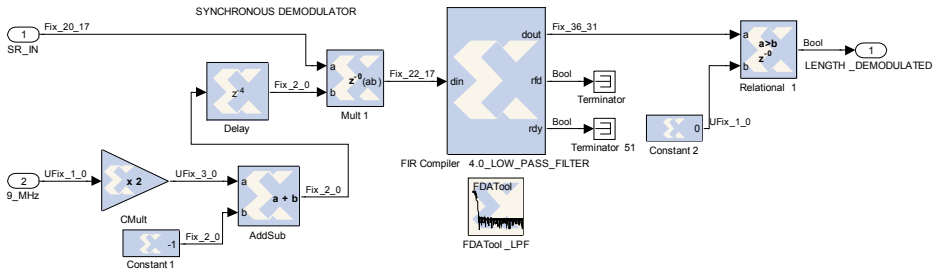

Fig. 30. Synchronous demodulator block diagram

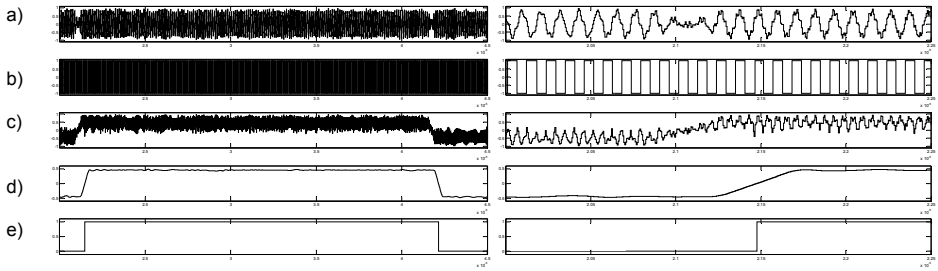

Fig. 31. Synchronous demodulator signals: a) synchronization input, b) 9 MHz multiplier input, c) multiplier output, d) filter output, e) length demodulated

6.2.3 Load generator

The circuit in Fig. 32 produces a pulse with the rising or falling edge at the input (Fig. 33). The output signal loads the initial value "11111" in the Linear Feedback Shift Register of the code generator in the receiver.

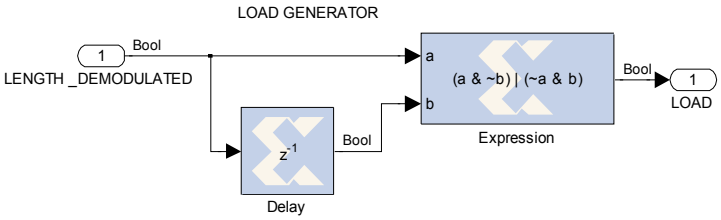

Fig. 32. Load generator

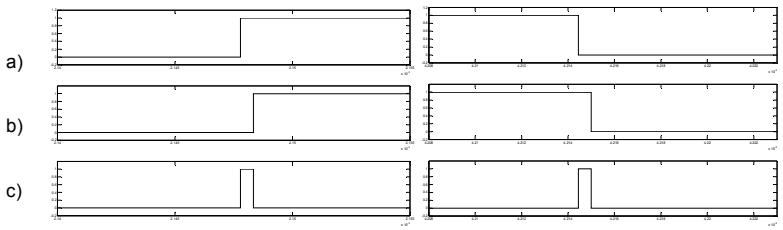

Fig. 33. Load generator signals: a) input, b) delayed input, c) output

6.2.4 Enable generator

The input of this system (Fig. 34) is the 9 MHz square carrier and generates a 1.5 MHz enable signal. A pulse is obtained with the rising edge at the input (Fig. 35). This signal is used as enable signal in a six states counter; a comparator checks when the counter output is zero. Finally, a pulse is generated with each rising edge of the comparator output. The output signal has the chip frequency, it will be used as input in a Linear Feedback Shift Register to recover the pseudorandom code.

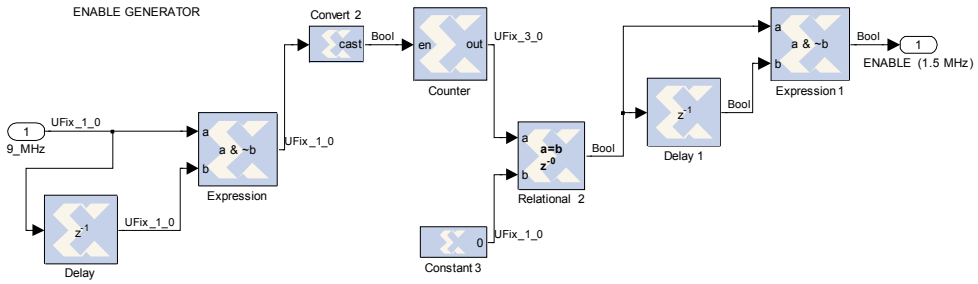

Fig. 34. Enable generator block diagram

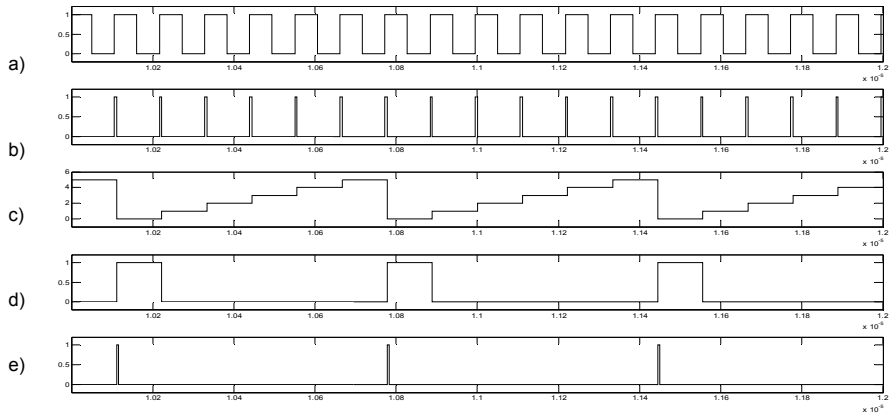

Fig. 35. Enable generator signals: a) 9 MHz input, b) internal pulse with the input rising edge, c) counter output, d) zero value in the counter output, e) enable generator output

6.2.5 Linear feedback shift register code generator

This system is a LFSR similar to the code generator in the transmitter (Fig. 36); with the exceptions of the load signal to initialize the "11111" value and the enable signal to generate the 1.5 MHz output rate. A delay block synchronizes the load and enable signal. The LFSR inputs and the value of the code recovered are shown in Fig. 37.

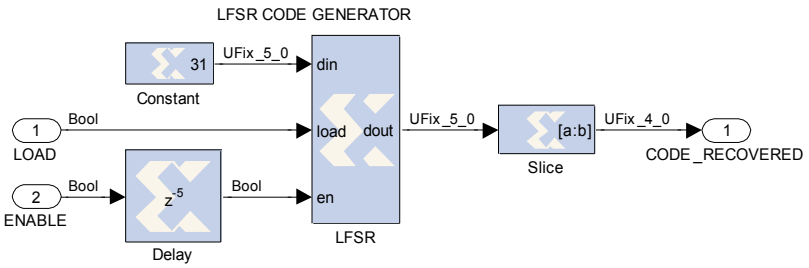

Fig. 36. Linear Feedback Shift Register code generator block diagram

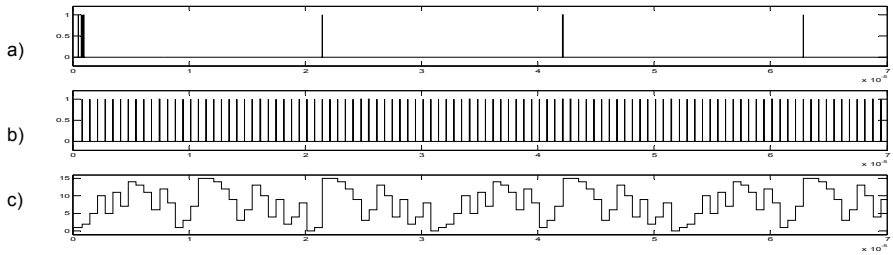

Fig. 37. Linear Feedback Shift Register code generator signals: a) LFSR load input, b) LFSR enable input, c) code recovered

6.3 Local oscillators

The code recovered is the local oscillators input (Fig. 38). The two oscillators were designed using two Direct Digital Synthesizer blocks, and the four bits input code must be converted to the input format of the DDS block. The frequency of the oscillator F_0 output (Fig. 39) is the transmitted frequency if the data in the transmitter is "0" minus 10.7 MHz; in other words, the left side of Table 1 minus 10.7 MHz. Consequently the value of the intermediate frequency in the receiver is 10.7 MHz. Similarly, the frequency of the oscillator F_1 output is the transmitted frequency if the data in the transmitter is "1" minus 10.7 MHz; in the same way, the right side of Table 1 minus 10.7 MHz.

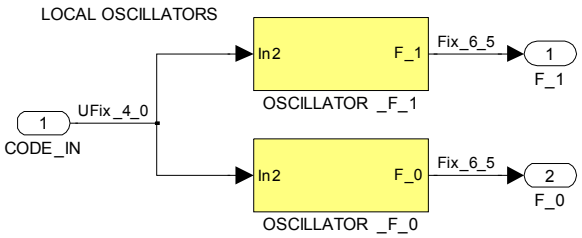

Fig. 38. Local oscillators block diagram

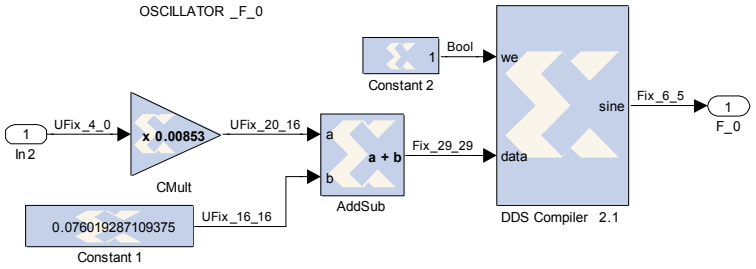

Fig. 39. Oscillator F_0 block diagram

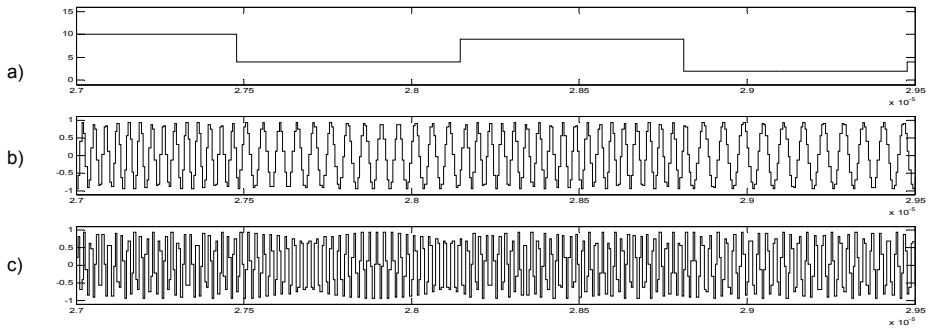

Fig. 40. Local oscillators signals: a) local oscillators input, b) oscillator F_0 output, c) oscillator F_1 output

6.4 Double branch demodulator

This demodulator is formed by two similar envelope detectors (Fig. 41). The inputs are the FHSS filtered signal and the local oscillators outputs. The FHSS filtered signal is delayed to keep the synchronization with the local oscillators frequencies. The top branch gets the waveform of the data and the bottom branch the inverter data. Lastly, the two outputs are compared and final output is the binary demodulated data.

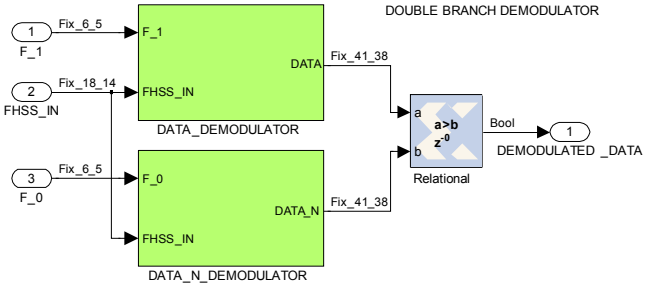

Fig. 41. Double branch demodulator block diagram

The Fig. 42 is the top branch block diagram. The mixer of the branch is the first multiplier and the intermediate frequency band pass filter. The second multiplier and the low pass filter is the envelope detector. The Fig. 43 shows the signals in the demodulator.

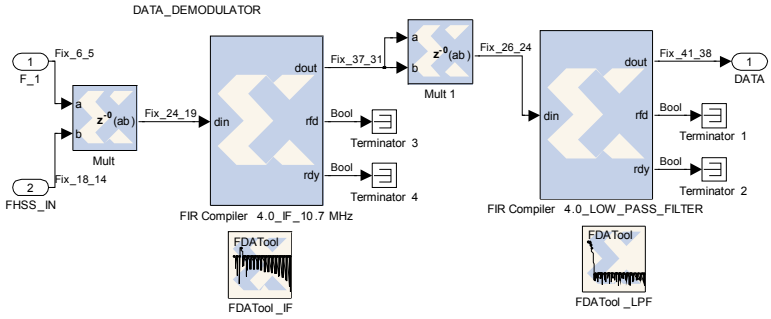

Fig. 42. Top branch demodulator block diagram

7. Channel simulation

Once the design of the transceiver has been finished, the performances can be tested inserting a channel between the transmitter and the receiver. For this purpose, an Additive White Gaussian Noise (AWGN) Simulink channel was chosen (Fig. 44). In this channel, the signal-to-noise power ratio is fixed by the designer. The Bit Error Rate (BER) was measured with the Error Rate Calculation block, where the delay between the data must be specified. Besides, the instant of synchronization in the receiver (20 microseconds) is indicated to start the bit error counter. This block generates three values: the first is the Bit Error Rate, the second is the number of errors, and the third is the number of bits tested. Finally, the BER is represented versus the signal-to-noise power ratio (Fig. 45).

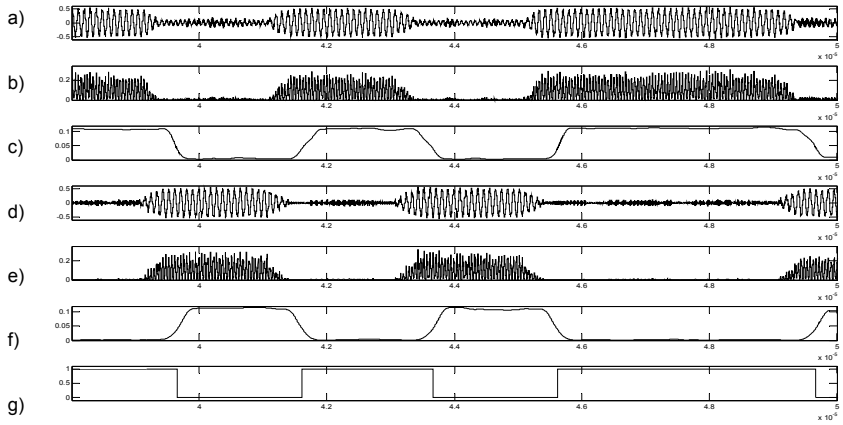

Fig. 43. Double branch demodulator signals: a) intermediate frequency filter output in the top branch, b) squared signal in the top branch, c) low pass filter output in the top branch, d) intermediate frequency filter output in the bottom branch, e) squared signal in the bottom branch, f) low pass filter output in the bottom branch, g) demodulated output

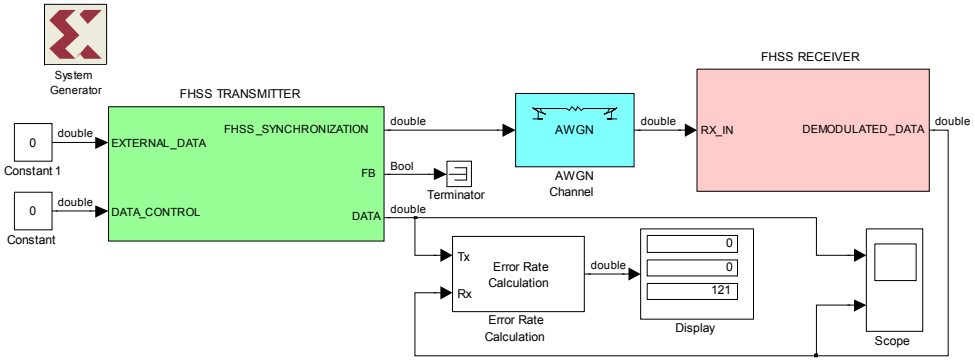

Fig. 44. Error rate calculation in presence of Additive White Gaussian Noise

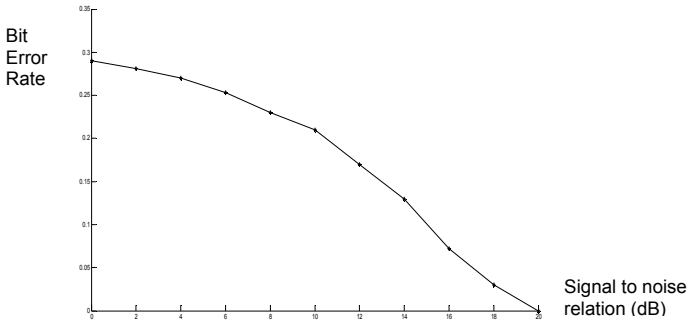

Fig. 45. Bit Error Rate represented versus the signal-to-noise power ratio (decibels)

8. Simulation and compilation with ISE

After the system has been simulated with Simulink, it can be compiled with System Generator. The chosen device is a Virtex 4 FPGA, and the hardware description language is Verilog. A project is then generated for Integrated System Environment, which includes the files for the structural description of the system. The syntax of the Verilog files can be checked, and the synthesis and behavioral simulation of the system can be executed (Fig. 46). Thereafter, the implementation of the design allows the timing simulation of the transceiver (Fig. 47). Lastly, the programming file is generated for the chosen FPGA.

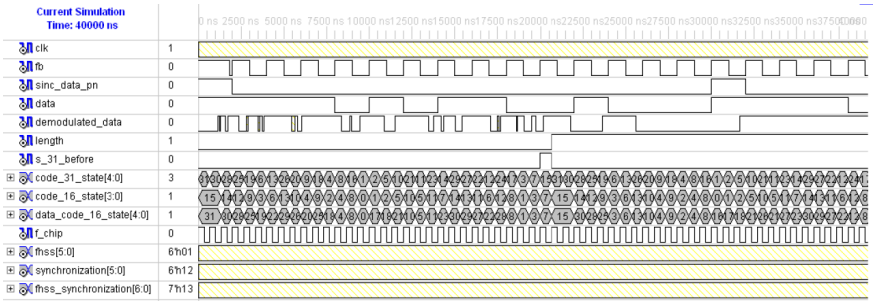

Fig. 46. A long behavioral simulation of the FHSS transceiver using ISE (40 microseconds)

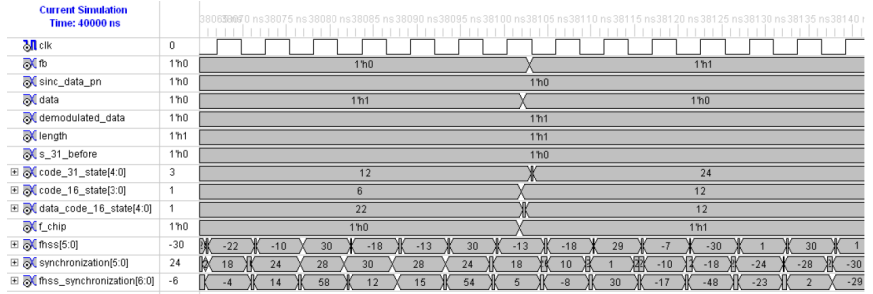

Fig. 47. Timing simulation of the FHSS transceiver using ISE (80 nanoseconds)

The Integrated System Environment software provides a power estimator that indicates a dissipation of 0.52 watts in the FPGA, and an estimated temperature of 31.4 degrees centigrade. The FPGA core is supplied with 1.2 volts and the input-output pins support the Low Voltage Complementary Metal Oxide Semiconductor (LVCMOS) volts standard. The design uses 491 of the 521 FPGA multipliers. The occupation rate of input-output pins in the FPGA is about 12.3%. However, this occupation rate can be reduced until 3.3% if internal signals are not checked.

9. Conclusions and future work

With this design methodology the typical advantageous features of using programmable digital devices are reached. Repeating a design consists in reprogramming the FPGA in the chosen board. The design and simulation times are decreased, consequently the time to

market is minimizing. The used tool permits great flexibility; in others words, the design parameters can be changed and new features can be checked within several minutes. The flexibility allows to change the Direct Digital Synthesizers and filters parameters and to check its performances. The Simulink simulations are easy to run, and the signals are shown in floating point format which make easier its analysis. These simulations are possible even before the compilation of the System Generator blocks to obtain the hardware description language files. With the System Generator it is possible to simulate the full transceiver, the transmitter and the receiver can be connected through a channel. Moreover, it is possible to simulate the transmission in presence of interference, distortion, multipath and other spread spectrum signals using different codes.

10. References

Analog Devices (2011). AD9851 DDS. URL: www.analog.com/static/imported-files/data_sheets/AD9851.pdf, active on April 2011

Hauck, S. & DeHon, A. (2008). *Reconfigurable Computing*, Elsevier, ISBN 978-0-12-370522-8, USA

MathWorks. (2011). Simulink. URL: www.mathworks.com/products/simulink, active on April 2011

Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*, Elseiver, ISBN 0750676043, New York, USA

Palnitkar, S. (2003).*Verilog HDL*. Prentice Hall, ISBN 9780130449115, USA

Pedroni, V. (2004). *Circuit Design with VHDL*, The MIT Press, ISBN 0-262-16224-5, USA

Pérez, S.; Rabadán, J.; Delgado, F.; Velázquez, J & Pérez, R. (2003). Design of a synchronous Fast Frequency Hopping Spread Spectrum transceiver for indoor Wireless Optical Communications based on Programmable Logic Devices and Direct Digital Synthesizers, *Proceedings of XVIII Conference on Design of Circuits and Integrated Systems*, pp. 737-742, ISBN 84-87087-40-X, Ciudad Real, Spain, November, 2003.

Simon, M.; Omura, J.; Scholtz, R. & Levitt, B. (1994). *Spread Spectrum Communications Handbook*, McGraw-Hill Professional, ISBN 0071382151, USA

Xilinx (2011). System Generator. URL: www.xilinx.com/tools/sysgen.htm, active on April 2011