ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
ELECTRÒNICA I INFORMÀTICA LA SALLE

PROJECTE FI DE MASTER

MASTER EN TELECOMUNICACIÓ

Embedded and DSP design tools integration for Xilinx FPGA development

ALUMNE	PROFESSOR PONENT
Carles Estevadeordal Serra	Ricard Aquilué

ACTA DE L'EXAMEN DEL PROJECTE FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Carles Estevadeordal Serra

va exposar el seu Projecte de Fi de Master, el qual va tractar sobre el tema següent:

Embedded and DSP design tools integration for Xilinx FPGA development

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Projecte amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL　　　　VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Agraïments

M’agradaria agrair al meu ponent, el Ricard Aquilué l’ajuda donada durant la realització d’aquest projecte.

Index

Abstract

The aim of this *projecte de final de master* is to design a firmware with VHDL and the Matlab System Generator for Xilinx DSP of a test application, which mixes several of the Xilinx tools to create the design. The target FPGA is a Spartan 6 from a SP605 test board.

In addition, this memory contains the main characteristics of the firmware design, a description of the tools used and the methodologies available to create the Firmware. Furthermore, it provides some theoric background and in some cases, it is useful as a guide to implement a similar solution to the one proposed.

Furthermore, there is a description of the findings in the System Generator design field and also with the interaction in the Xilinx FPGA design flow. There have been described the tests which were performed to ensure the correctness of the design.

Resum

Per aquest projecte, primer de tot es fa una descripció de les capacitats de la tecnologia d'FPGA i les eines emprades. S'ha donat especial rellevància a la tecnologia de celes i les capacitats de síntesi de les FPGA Xilinx ja que influeixen en gran mesura en la metodologia de disseny.

L'objectiu d'aquest projecte es crear un disseny RTL i un firmware en VHDL per a un FPGA Xilinx emprant Xilinx System Generator a l'entorn Matlab Simulink. Tot i això, per aconseguir aquest objectiu s'ha portat a terme un exhaustiu estudi de l'entorn de disseny utilitzat. Ja que, com en la majoria d'entorns de disseny dels fabricants d'FPGAs, la interacció entre varis programes que treballen en diferents aspectes del disseny firmware és complex. Per això s'estudien les interaccions entre aquests programes i les seves capacitats especifiques dins del procés complet de disseny SoC.

Aquesta memòria dona especial rellevància a l'estudi del Xilinx System Generator y dels passos necessaris per exportar i simular els dissenys resultants. Una de les raons principals per aquest interès és el fet que aquest entorn aporta un bon nivell d'abstracció al disseny lògic facilitant l'accés a aquest a gent acostumada a treballar en el disseny de sistemes de processat digital del senyal en entorn Matlab Simulink. Tot i això aquesta guia no pretén substituir a l'ajuda pròpia del System Generator, sinó ser-ne un complement, per això també es donen algunes indicacions de com accedir a aquesta sempre que sigui necessari.

En quant al System Generator, s'ha fet un estudi detallat dels conceptes de disseny, blocs i funcions disponibles per realitzar aquests tipus de dissenys. Addicionalment, s'han detallat els mètodes d'interconnexió i el sistema de data rates que existeix entre diferents blocs; ja que el sistema de busos i tipus es diferent dels utilitzats per els llenguatges HDL i representen la principal font de problemes utilitzant System Generator.

A més a més, s'ha comprovat que les màquines d'estats fetes en Matlab amb un arxiu m-code integrat dins del System Generator s'implementen correctament. Ja que aquesta metodologia de disseny és interessant degut al fet que és més dissenyar una màquina d'estats d'aquesta manera és més senzill i menys propens a errors. Per això s'han dut a terme suficients simulacions per assegurar que la implementació és correcte.

Finalment, també s'han realitzat tests en una placa SP605 per assegurar que el disseny es sintetitza correctament i per validar la metodologia de disseny utilitzada per crear el firmware.

1 - Introduction

For this project, first of all, a brief description of the FPGA technology, tools and capacities used is made. Special attention to the Xilinx cell technology and device implementation capabilities is done, since it significantly affects the design methodology.

The main objective of this project is to create an RTL design and a firmware with VHDL for a Xilinx FPGA using the Xilinx System Generator in a Simulink Matlab environment. Nevertheless, to achieve this objective an accurate analysis and study of the design environment is required. Due to the fact that, as in most FPGA vendor ISE's, the interaction between various programs which work in different aspects of the IC design is complex. Therefore, the interaction process, tools and capabilities are also among the main fields of study of this thesis.

This thesis also does an exhaustive exploration of the Simulink design methodology integrated with Xilinx System Generator and the steps needed to export and simulate the resulting design. This design methodology offers very good abstraction capabilities and offers a familiar interface for those used to work in digital signal processing with Matlab. Nevertheless, this document does not pretend to replace the System Generator help, but being a complement, therefore some tips to access to this help whenever necessary are provided.

Regarding to the System Generator, a detailed study of the design concepts, blocs and functions available to perform this type of implementations is done. In addition, the interconnection methods available and data rates between blocks have been detailed- since the busses and types are different from the ones used by HDL languages and they are the main source of design errors when using System generator.

Furthermore, the correct design of a FSM, using an m-code file integrated into the System Generator design, is also viewed; given the fact that it is far simpler and less error prone to design a FSM from a script like language such as Matlab, than in VHDL. Additionally, an extensive simulation has been done to assure that the correct implementation has been also performed.

Finally, lab tests using the Xilinx SP605 board were performed to ensure that the design is properly synthesised and to validate the methodology used to achieve the firmware.

2 – Tools and design methodology

2.1 – Overview

Xilinx was the first viable FPGA company, it was founded in Silicon Valley in 1984 **[5]**. In 2010, Xilinx has around 3000 employees and leads the FPGA market with 1800 million $ revenues in 2010, which represents more than the 50 % of the PLD market **[6]**. It develops products for many markets form common electronic devices to the aerospace industry. Their current set of products also include a digital design development software, a wide range of IP cores which may be embedded indo designs made using Xilinx FPGAs and complementary electronic products such as test boards and kits **[13]**.

From the first FPGA made by Xilinx, the XC2064 (1985) **[7]**, to actual FPGAs, the complexity and features have been increased dramatically, as it can be seen in Figure 1 and Figure 2; where the Basic logic blocs of XC2064 and Spartan 6 FPGA family are shown, which is the FPGA used for this project. Nevertheless, the main FPGA characteristics enabling the low-level implementation of logic functions from HDL designs, which enable to embed high-speed custom designs into a programmable microchip, remain the same.

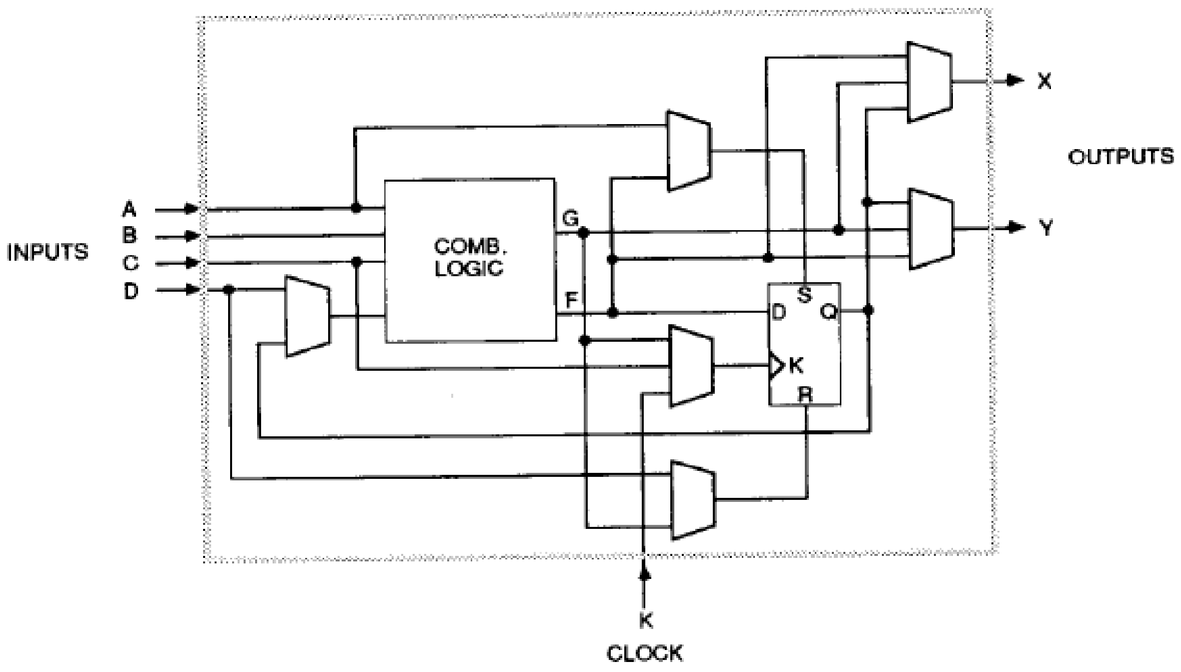


Figure 1. XC2064 Cell.

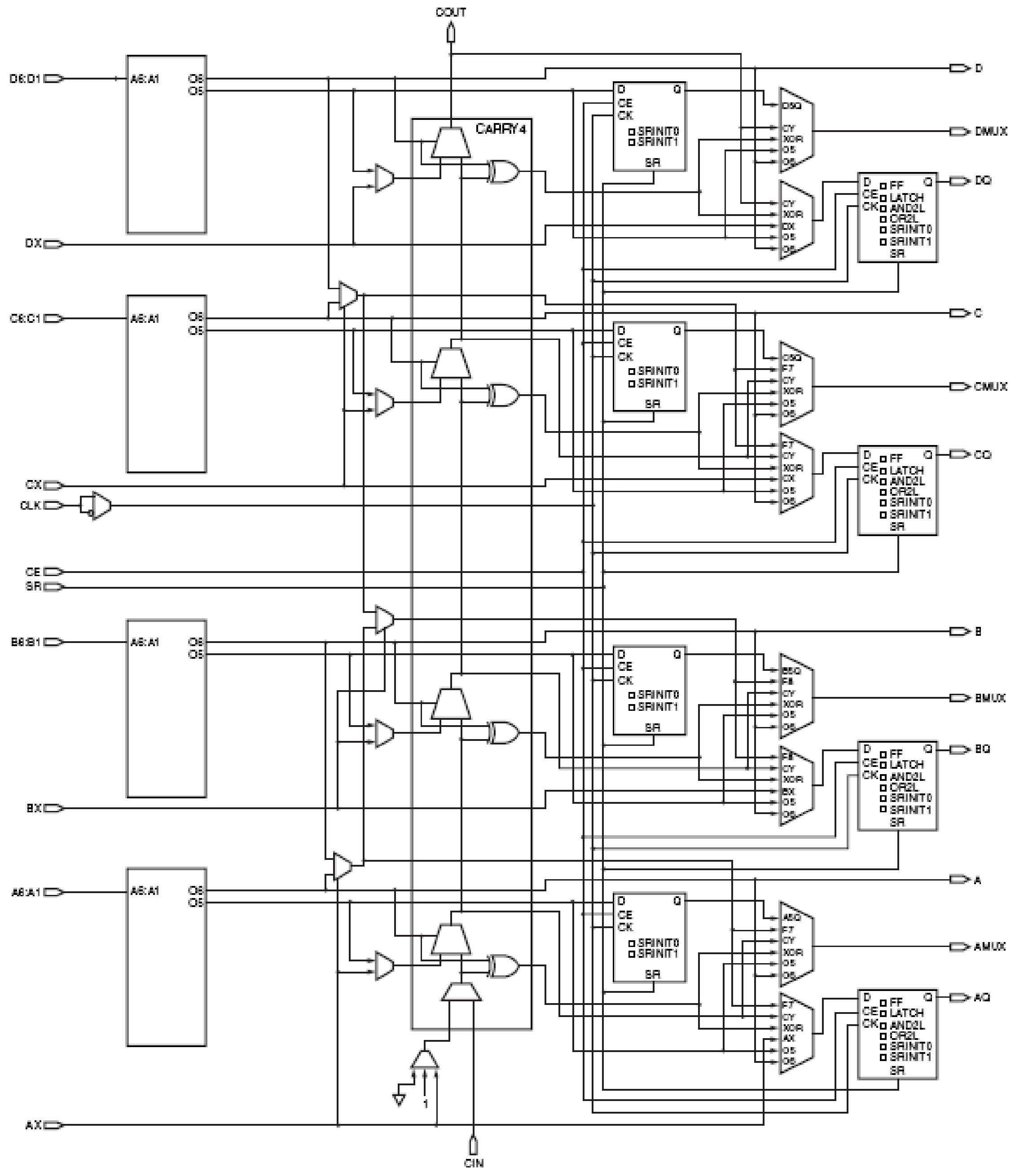


Figure 2. Spartan 6 SLICEL [12].

Modern FPGAs offer a very high degree of versatility having integrated in those devices a superset of technologies such as microprocessors, microcontrollers and IPs. As a microprocessor, Xilinx offer the Microblaze microprocessor which is implemented with standard blocks **[8]**, adding the easiness of programming and versatility of a microprocessor to the high processing capabilities and speed of a FPGA. Regarding to the Picoblaze; it is a very compact, full featured, 8 bits micro-controller that facilitate to integrate control capabilities to the FPGA design **[9]**. In some FPGAs, there have been implemented even hard coded microprocessors such as PowerPc microprocessors in some Virtex FPGA series. Xilinx also offers a wide base of IPs of own craft- 296 made by Xilinx and 299 made by 3rd party vendors which add a lot of power and programming easiness to FPGA design trough the Xilinx System Generator **[10]**.

Xilinx offer a wide variety of products, which range from the programmable devices and development cards to the software necessary to synthesize complex electronic solutions. This full set of tools associated to the FPGAs is what makes FPGAs such a powerful and versatile device. Xilinx ISE enables the design and simulation, and the test boards allow the development and testing of the designed application to finally implement that solution into a production board, in an error prone environment. In Figure 3 a custom board designed to hold a Virtex 5 FPGA can be seen.

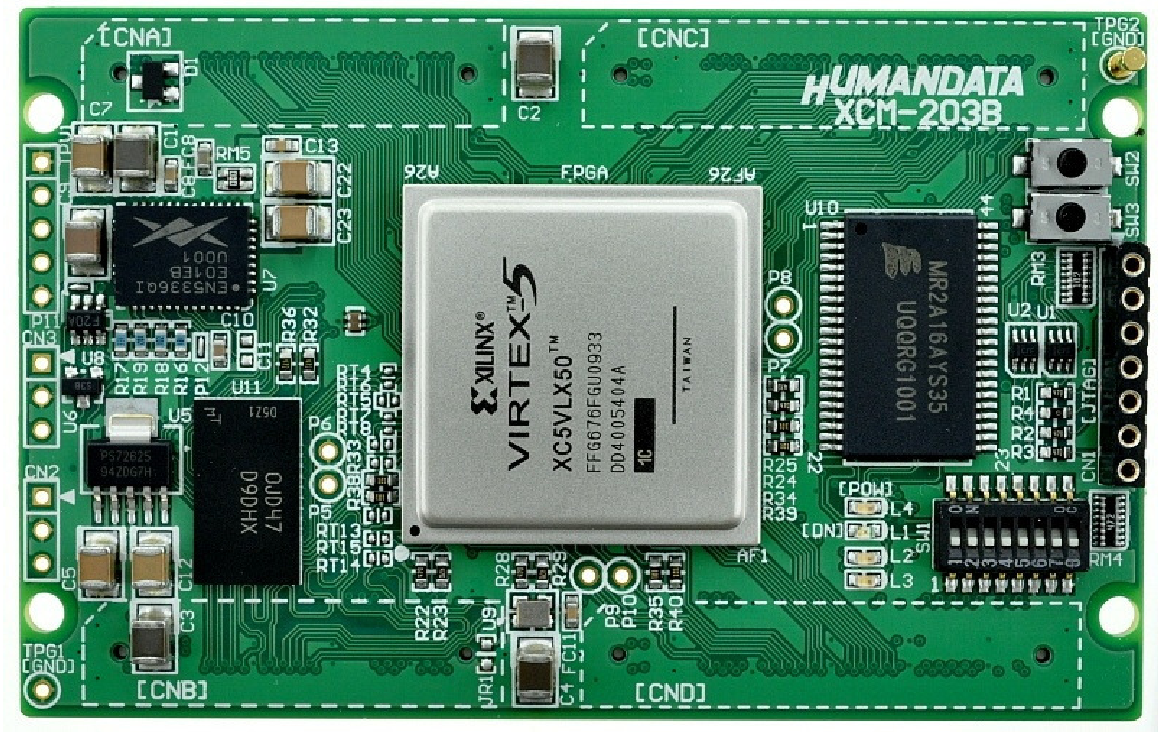


Figure 3. Virtex 5 board.

Due to the high performance and capabilities of Xilinx FPGAs it is possible to find them in many advanced commercial electronic applications such as motherboards, PCIe Cards, video acceleration cards, mobile phones, etc...

In Figure 4 there can be seen a Xilinx FPGA used as a main controller of a computer motherboard.

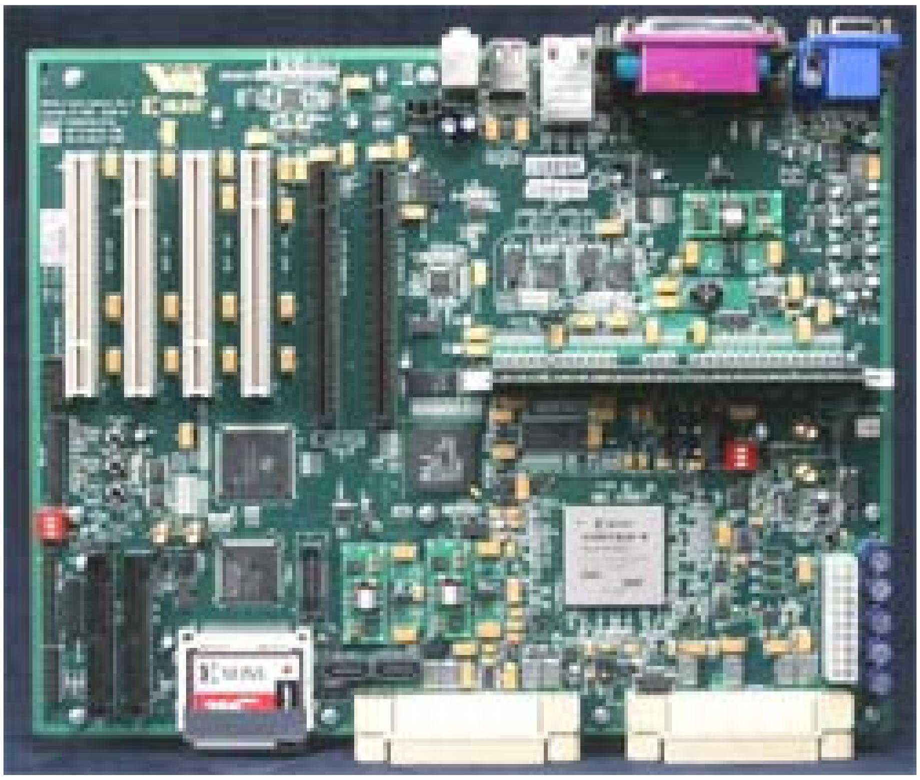


Figure 4. Motherboard using a Xilinx FPGA.

2.2 – Xilinx FPGA

The FPGA used to test this project is a Spartan-6 XC6SLX45T, this FPGA is installed in a Spartan®-6 FPGA SP605 Evaluation Kit test board. This test board is a stable and reliable development environment that allows the testing of the FPGA main features.

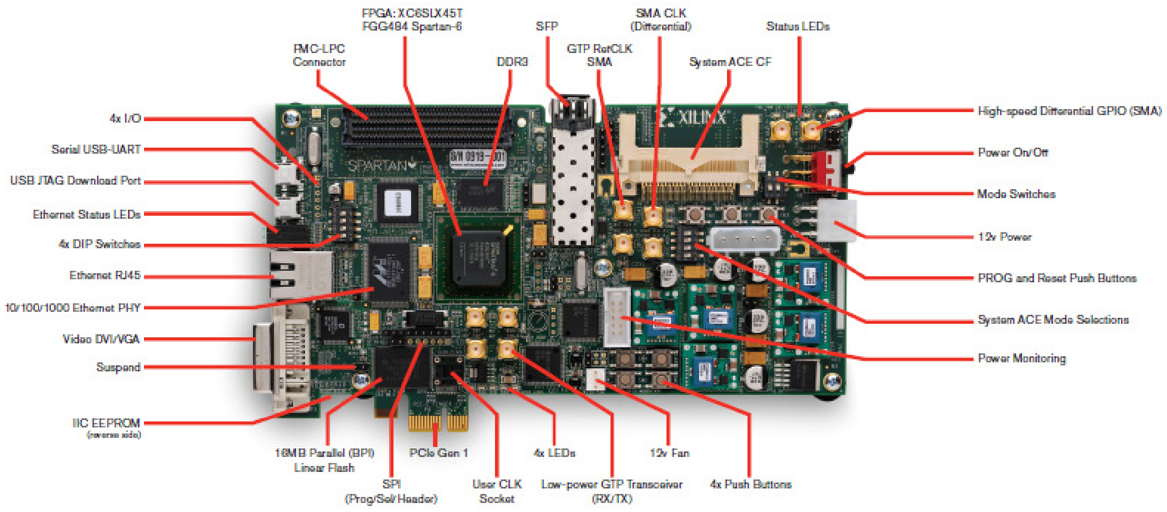


Figure 5. Spartan®-6 FPGA SP605 Evaluation Kit test board.

The Spartan-6 XC6SLX45T is a 45 nm low-power copper process technology FPGA with a core voltage of 1.2V **[11]**. This FPGA can is can perform quite sophisticated tasks even holding a small microcontroller or microprocessor such as the ones provided for Xilinx as IPs, in addition to other high performance designs such as high rate signal processing. A brief description of its resources can be seen at Figure 6.

Number of Slice Registers	54,576
Number of Slice LUTs	27,288
Number of bonded IOBs	296
Number of DSP48A1s	58
Number of PLL_ADVs	4

Figure 6. Outline of Spartan-6 XC6SLX45 resources.

The Spartan®-6 FPGA SP605 Evaluation Kit is designed to be connected to into a computer PCIe slot and expands the functionality of the FPGA providing a wide set of interconnection ports; which allow to use the FPGA in conjunction with other designs that provide any of those ports.

Among the ports provided in the SP605 there are: a PCIe port, a USB JTAG port, SMA connectors, Multi-Gigabit GTP MGTs Transceivers, a SFP Module Connector, a 10/100/1000 Tri-Speed Ethernet PHY, an USB-to-UART Bridge and an IIC Bus **[6]**. A detailed description of those ports, their interfaces and configuration options can be consulted at the Xilinx, SP605 Hardware User Guide. As an outline, in Figure 7 a block diagram of the connection ports and banking of the SP605 is provided.

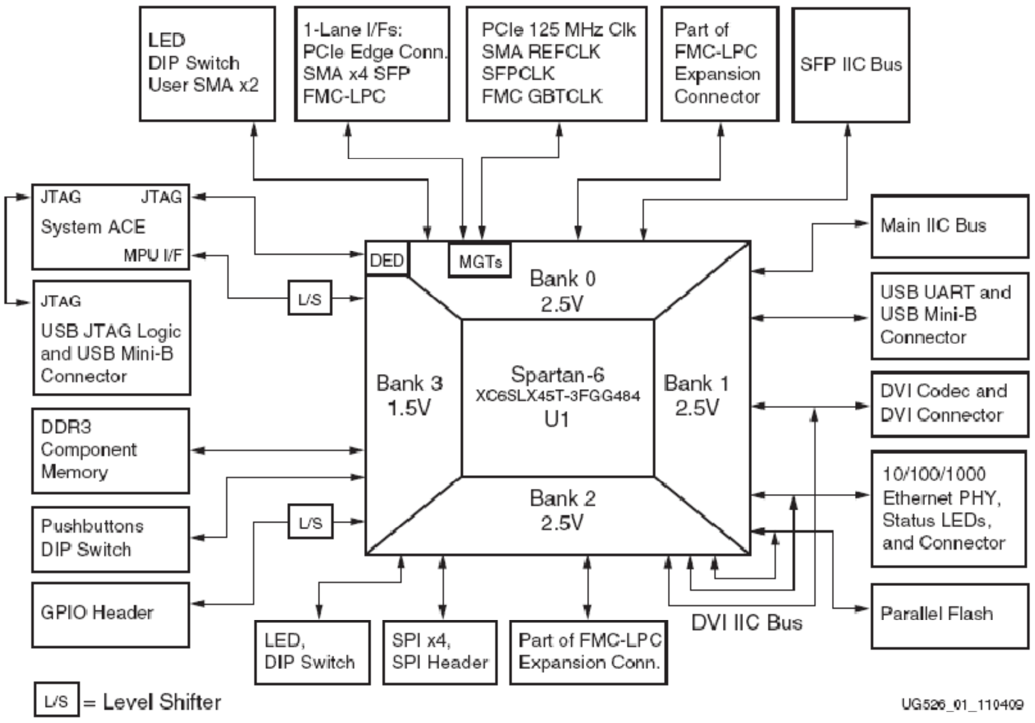


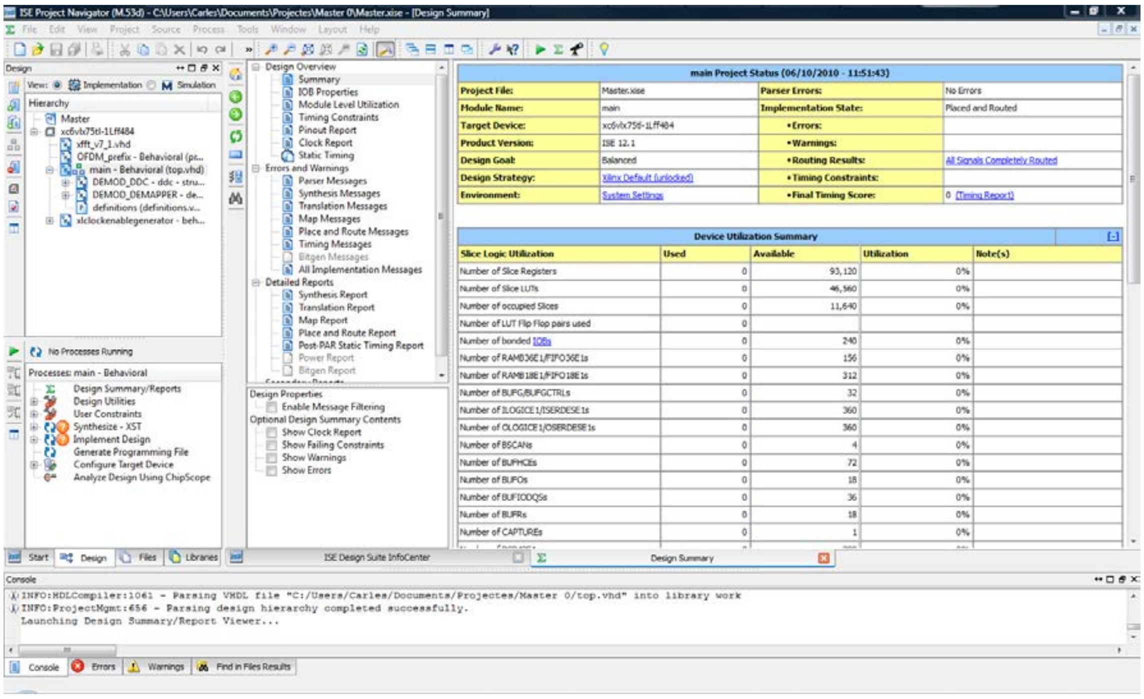
Figure 7. SP605 Connectivity features and banking [14].

2.3 – Xilinx design and simulation tools

The set of tools used for this project include Xilinx ISE, Matlab with Xilinx System Generator and the Xilinx ISE Simulation Tool.

They all have a place in the design process and the main design tool continues to be the Xilinx ISE, since any design made with Matlab has to be imported as a VHDL or Verilog entity and the final pin out configuration has to be done with Plan Ahead. In addition, the designs made with Xilinx System Generator cannot handle all the functions that a VHDL can do, therefore they are intended to simplify some design parts but in most cases, it may be necessary to integrate them into more complex VHDL designs.

In Figure 8 the Xilinx ISE main window is shown, it allows to add the System Generator generated VHDL entities with the *Add Source…* option, then they may be integrated and simulated in the design as desired.



main Project Status (06/10/2010 - 11:51:43)			
Project File:	Master.xise	**Parser Errors:**	No Errors
Module Name:	main	**Implementation State:**	Placed and Routed
Target Device:	xc6vlx75tl-1Lff484	**•Errors:**	
Product Version:	ISE 12.1	**•Warnings:**	
Design Goal:	Balanced	**•Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	
Environment:	System Settings	**•Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	0	93,120	0%	
Number of Slice LUTs	0	46,560	0%	
Number of occupied Slices	0	11,640	0%	
Number of LUT Flip Flop pairs used	0			
Number of bonded IOBs	0	240	0%	
Number of RAMB36E1/FIFO36E1s	0	156	0%	
Number of RAMB18E1/FIFO18E1s	0	312	0%	
Number of BUFG/BUFGCTRLs	0	32	0%	
Number of ILOGICE1/ISERDESE1s	0	360	0%	
Number of OLOGICE1/OSERDESE1s	0	360	0%	
Number of BSCANs	0	4	0%	
Number of BUFHCEs	0	72	0%	
Number of BUFOs	0	18	0%	
Number of BUFIODQSs	0	36	0%	
Number of BUFRs	0	18	0%	
Number of CAPTUREs	0	1	0%	

```
INFO:HDLCompiler:1061 - Parsing VHDL file "C:/Users/Carles/Documents/Projectes/Master 0/top.vhd" into library work
INFO:ProjectMgmt:656 - Parsing design hierarchy completed successfully.
Launching Design Summary/Report Viewer...
```

Figure 8. Xilinx ISE Project Navigator main window.

The Xilinx System Generator integration into Matlab Simulink is the main field of study of this project and a sample project can be seen in Figure 9; which shows as a background the standard Matlab window. At its central right part the Simulink library browser can be seen; where the blocks to design with System Generator are taken from. At the central left part of the figure, a design window with a basic design made generable for a Xilinx FPGA.

To start a design in Matlab Xilinx System Generator with Matlab R2009a and Xilinx ISE 12.1 Installed; the first step once Matlab has been opened is to open the Simulink Library dialog. The second step is to open or create a design. Once a design window is opened, the blocks from the Simulink library can be dropped into the design to be used; most of the synthesizable blocks are found in the Xilinx libraries called: Xilinx Blockset, Xilinx Reference Blockset and Xilinx XtremeDSP Kit. Marked with number four in Figure 9, it can be seen a basic design with the System Generator block at the bottom and a basic synthesizable design isolated by a Xilinx input and output from a standard Simulink Source and Display sink.

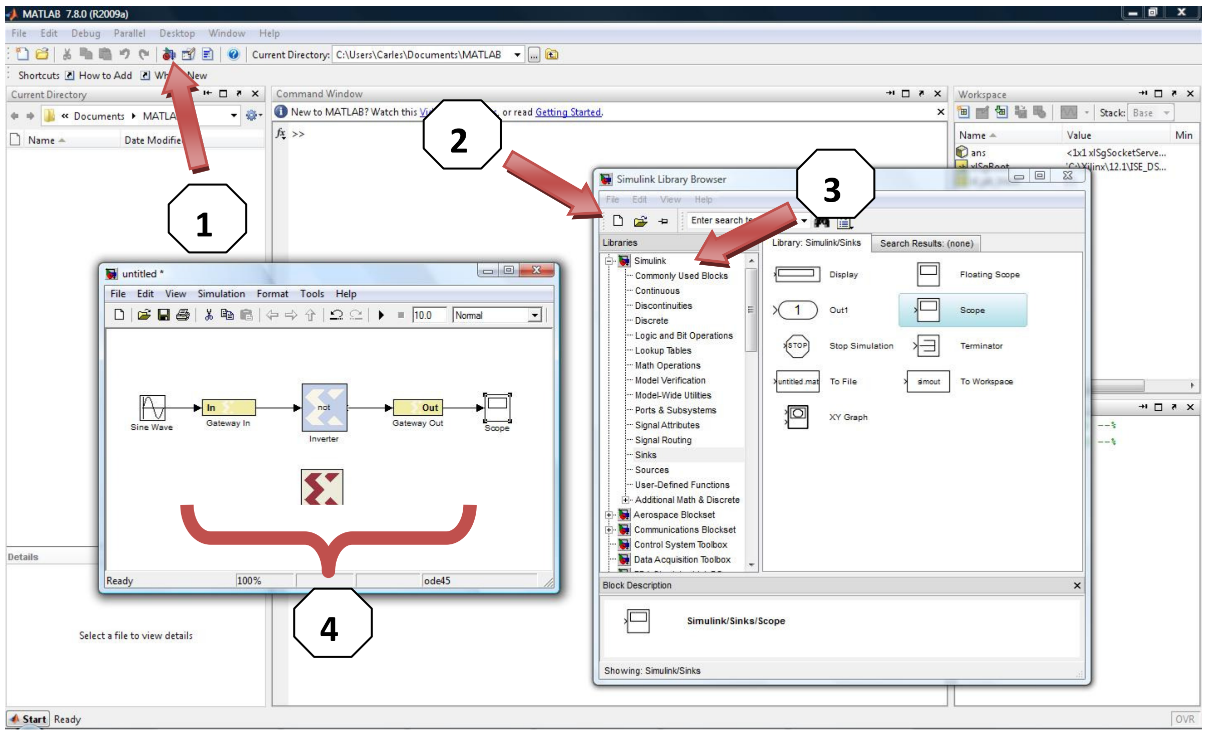
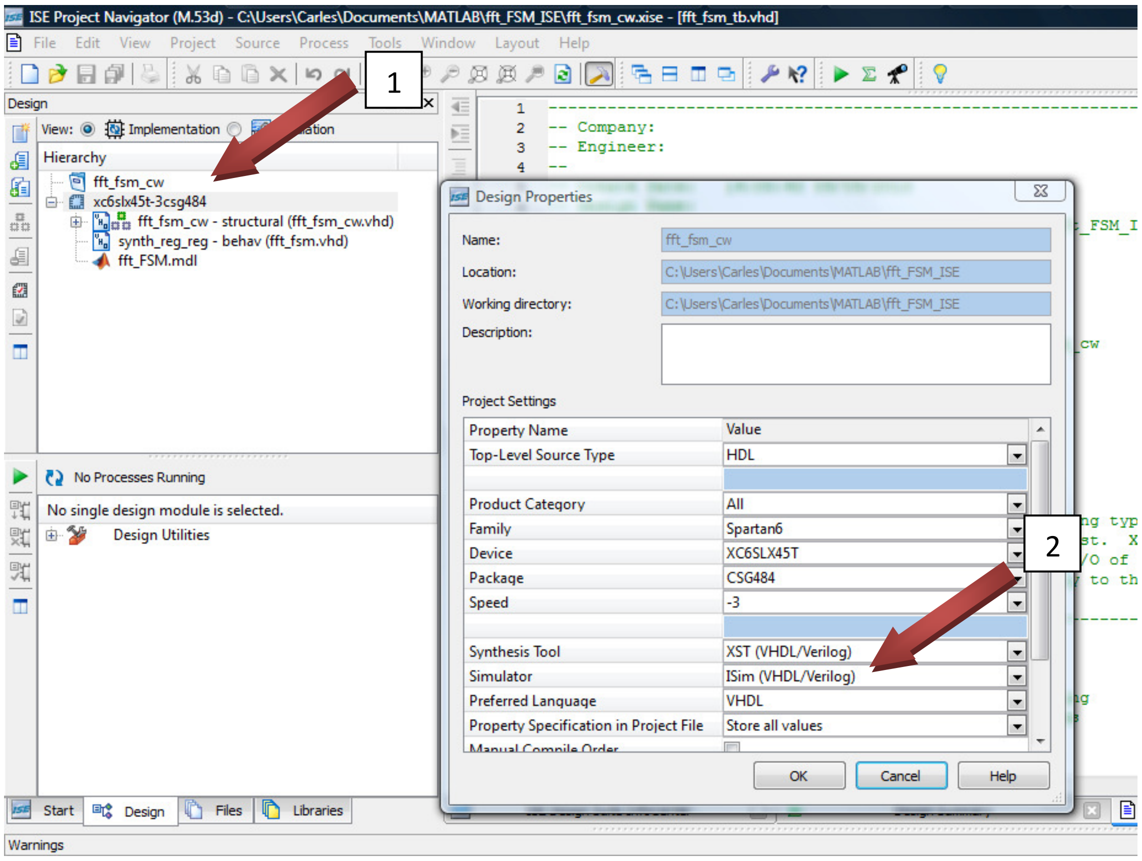


Figure 9. Matlab main window with an open Simulink Library dialog and a System Generator project.

Although it is possible to perform simulations of the design in Matlab, it is also necessary to use the Xilinx simulation Tool. It is also possible to perform simulations using a special version of ModelSim, despite the fact that ModelSim is a very versatile, scriptable and powerful simulation tool that is used by many FPGA vendors, it comes as an additional package and it was not available for this project.

If not set by default, the selection of ISim as the default Xilinx ISE simulator is quite tricky; since, if another simulation software is chosen and the license is not correctly set, the program will just fail to execute. To set ISim as the default ISE simulator first it is necessary to right clicking on the design FPGA model instance and choose Design

Properties, once the Design Properties dialog is open, ISim can be set in the Simulator option; as marked in the Figure 10.



ISE Project Navigator (M.53d) - C:\Users\Carles\Documents\MATLAB\fft_FSM_ISE\fft_fsm_cw.xise - [fft_fsm_tb.vhd]

- fft_fsm_cw
- xc6slx45t-3csg484
  - fft_fsm_cw - structural (fft_fsm_cw.vhd)
  - synth_reg_reg - behav (fft_fsm.vhd)
  - fft_FSM.mdl

No single design module is selected.

```
--------------------------------------------------------------------------
-- Company:
-- Engineer:
--
```

Design Properties

Name:	fft_fsm_cw
Location:	C:\Users\Carles\Documents\MATLAB\fft_FSM_ISE
Working directory:	C:\Users\Carles\Documents\MATLAB\fft_FSM_ISE
Description:	

Project Settings

Property Name	Value
Top-Level Source Type	HDL
Product Category	All
Family	Spartan6
Device	XC6SLX45T
Package	CSG484
Speed	-3
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	

Figure 10. Choose ISim as Xilinx ISE simulator.

Once the correct simulation enviroment is used, the simulation option in the project design tab may be choosen. At the simulation tab, the different test benches of the design are available to be chosen. In the processes window situated below, the simulator can be launched by double clicking on the corresponding optio of the tree; in Figure 11 the Behavioral Check Syntax has been made prior to the Simulate Behaioral Model option is chosen which will launch ISim to view te simulation of the current testbench.

The type of sumulation available (Behavioral, Post-Translate, Post-Map or Post-Route) depends on the compilation level achieved in the implementation tab, so prior to simulate it is necessary to run the processes needed for the desired level of simulation in the implementation Tab.

It has been observed during the simulation and exporting process from System Generator a bug in which the Test Bench choosen in the simulation tab is allways oberriden by the generated with System Generator, a workaround to solve this problem is to remove the ISim folder of the project root folder.

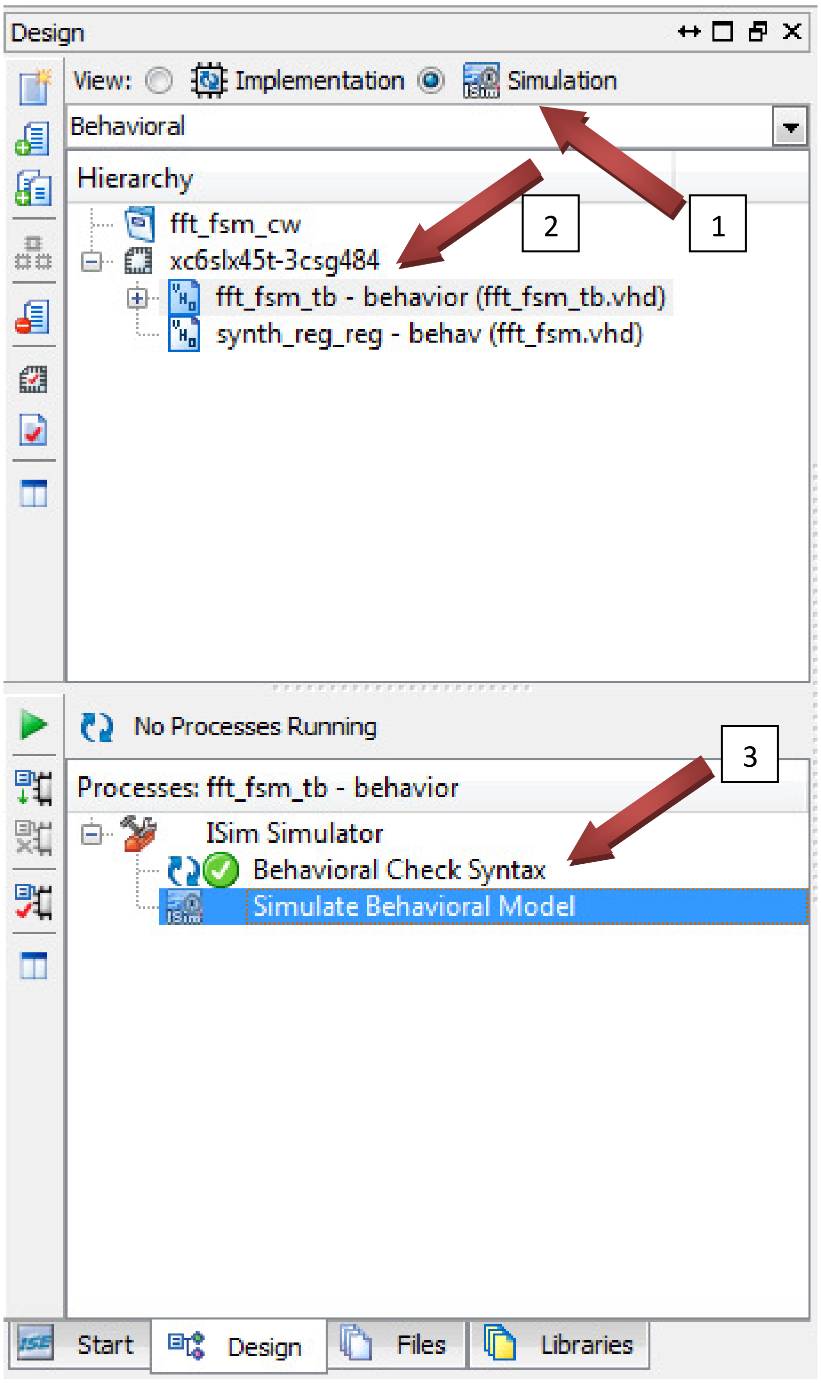
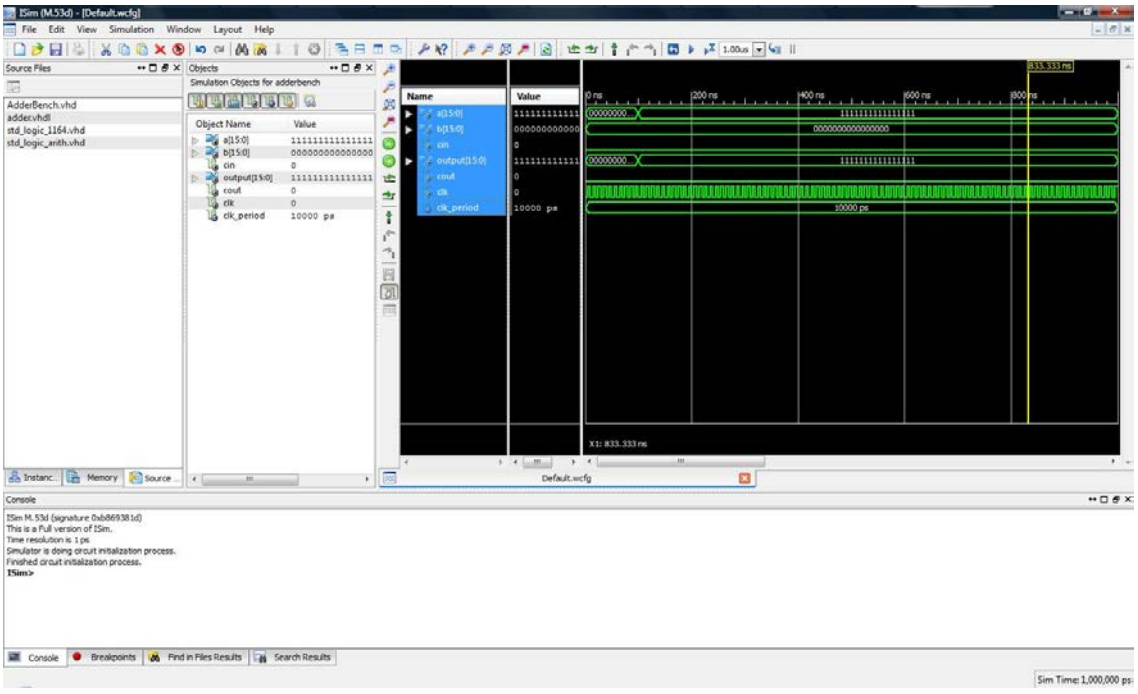

Design
View: Implementation Simulation
Behavioral
Hierarchy
fft_fsm_cw
xc6slx45t-3csg484
fft_fsm_tb - behavior (fft_fsm_tb.vhd)
synth_reg_reg - behav (fft_fsm.vhd)
1
2
No Processes Running
Processes: fft_fsm_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
3
Start
Design
Files
Libraries

Figure 11. Xilinx ISE- process to open a simulation with ISim.

Once the simulation process is launched, as seen in Figure 11, the simulation compilation process will start and an ISim instance will be launched which automatically runs the selected testbench. In Figure 12, ISim can be seen performing a simulation of a full adder. As it can be seen ISim and ModelSim share a similar aspect, configuration options and functionality; therefore, it is easy to adapt to this tool for those who come from other simulation environments.



Object Name	Value
a[15:0]	1111111111111111
b[15:0]	0000000000000000
cin	0
output[15:0]	1111111111111111
cout	0
clk	0
clk_period	10000 ps

```
ISim M.53d (signature 0xb869381d)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Figure 12. ISim simulation window.

2.4 – Matlab and Xilinx System Generator

2.4.1 – Matlab Simulink and Xilinx System Generator integration

The design environment for System generator embedded in Matlab Simulink is a graphical tool that allows an easy and quick design of common communication devices; it simplifies very much the work of bit scaling and IP interconnection, since these matters are almost automatically handled by the Simulink System Generator tool.

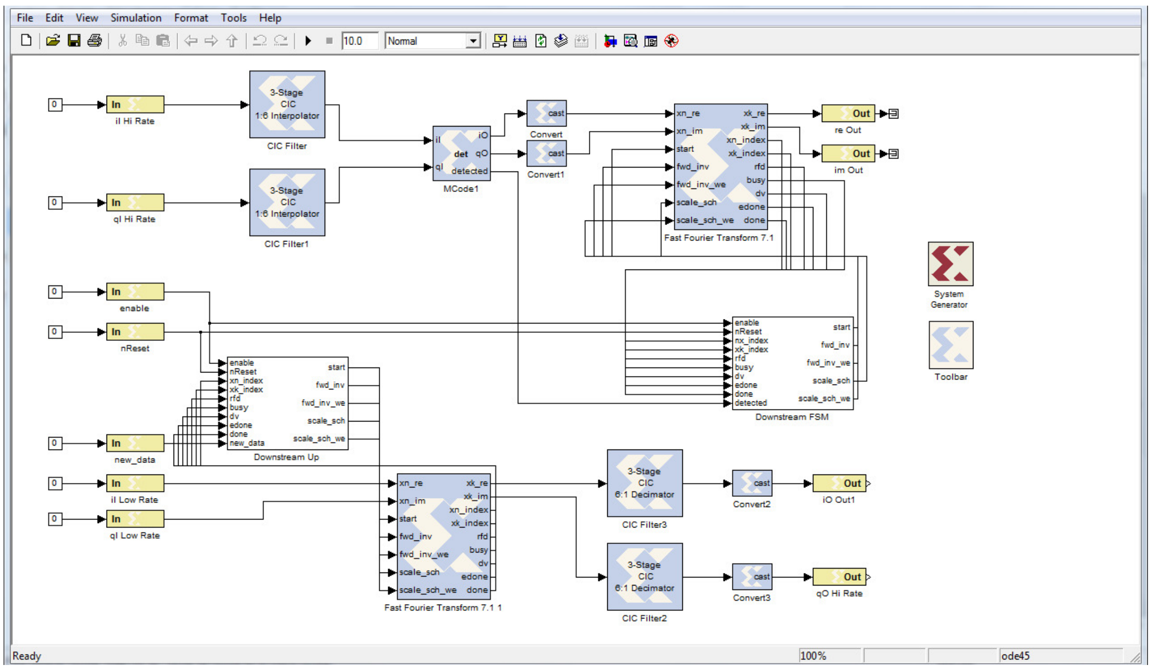


Figure 13. Xilinx System Generator Simulink sample design.

The Simulink environment simplifies the design process very much; since it is very easy to connect busses and wires between blocks and the sizing of the buses is done automatically, just some occasional casts performed over buses and the adjustment of the sample rate are required form time to time. Matlab software also allows to easily group blocks and connections to organize the designs and even to allow reusing common used modules in your designs.

The main design process using Xilinx System Generator is based in using the Xilinx Blocksets accessible from the Simulink Library browser. They represent, in the majority of the cases, IPs available with Xilinx ISE and the design process consists in aligning, connecting and configuring them to achieve the desired result. The Xilinx Blockset library of the Simulink Library browser can be seen at Figure 14.

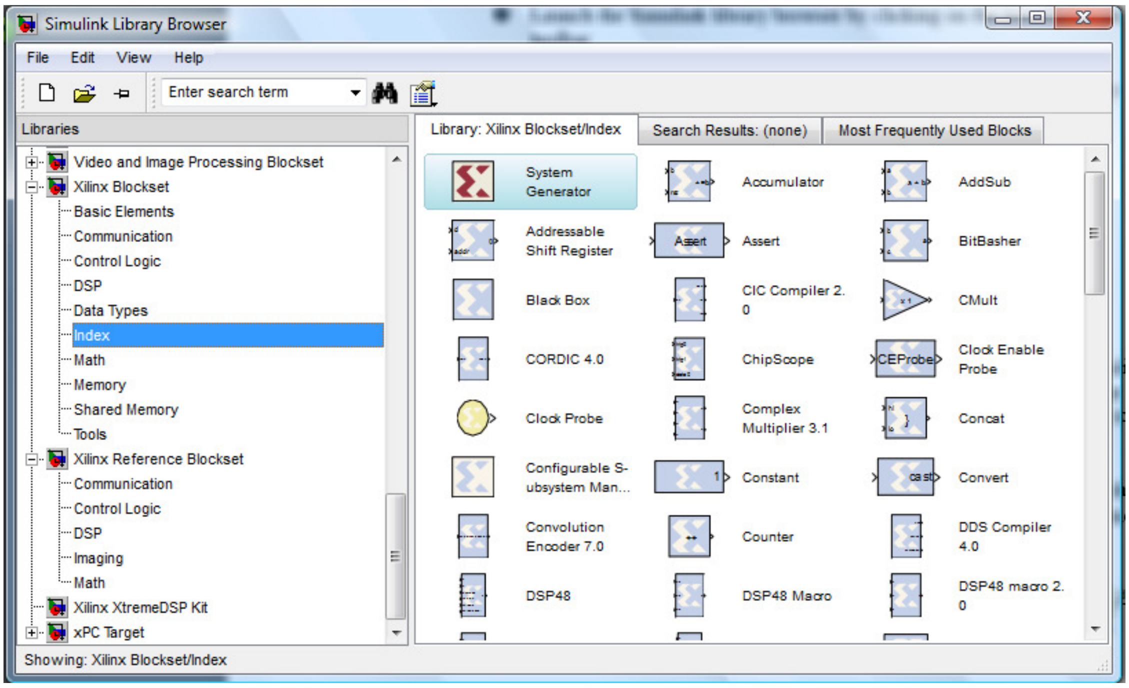


Figure 14. Xilinx Simulink Blockset Library.

In addition to the huge functionality achievable with the Xilinx blocks, it is possible to import VHDL entities and m files with the **Black Box** and the **M-Code** Xilinx blocks to use them in the Simulink design. On the contrary, for the system generator it is not possible to instantiate standard Simulink blocks apart from the Demux, From, Goto and Mux blocks. Nevertheless, it is possible to use them for simulation purposes.

As told before, the basics of designing a project with System generator consists in placing in the Simulink environment, blocks taken from the Xilinx block sets to elaborate a logic design. The simplest type of design is composed by a Xilinx System Generator block and it is divided per se in two parts with the Gateway In and Gateway Out blocks- one part which is synthesizable and one part which is not. In Figure 15 a simple design with the synthesisable part of the design in the blue circle.

The synthesisable part of the design must be separated from the non synthesisable part of the design by Gateway In and Gateway Out blocks from the Xilinx block set and in the synthesisable region, the blocks must be from the Xilinx blockset allowing few exceptions as described in section *2.4.2.5* – Can common Matlab Simulink blocks be synthesized by System Generator?.

The non synthesisable part of the design can contain any type of block and it can be used to compare the signal and to generate complementary test structures. This part of the design will be ignored when the generation step to map the design to the FPGA is performed.

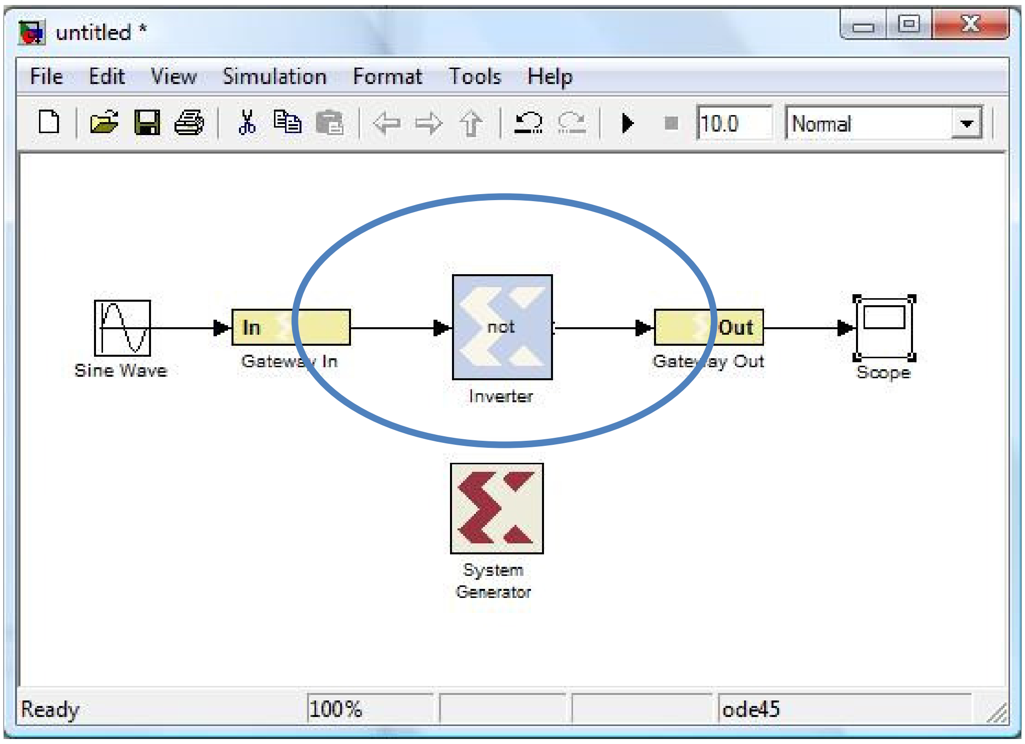
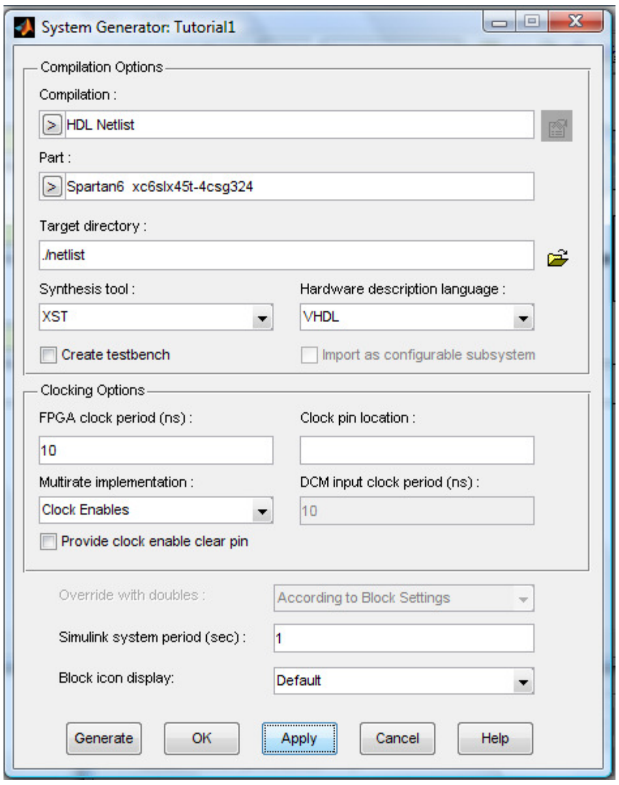


Figure 15. System generator basic design.

Once the design is made and all the pins and buses are closed it can be simulated as a normal Simulink model with the play button and it can be exported to the Xilinx ISE with the System Generator block. The System Generator block offers various options to configure the type of exported project desired, even allowing to choose a bitstream to program the FPGA. It is also possible to choose the target FPGA, the folder where to store the project and the HDL language to describe the generated blocks and TestBenches.

With the system generator dialog it is also possible to tune up some features of the Simulink blocks such as the Simulink system period and the Block icon display; which is very useful to debug data type errors, allowing to view the input and the output data types which is a common source of conflicts when interconnecting blocks.


System Generator: Tutorial1

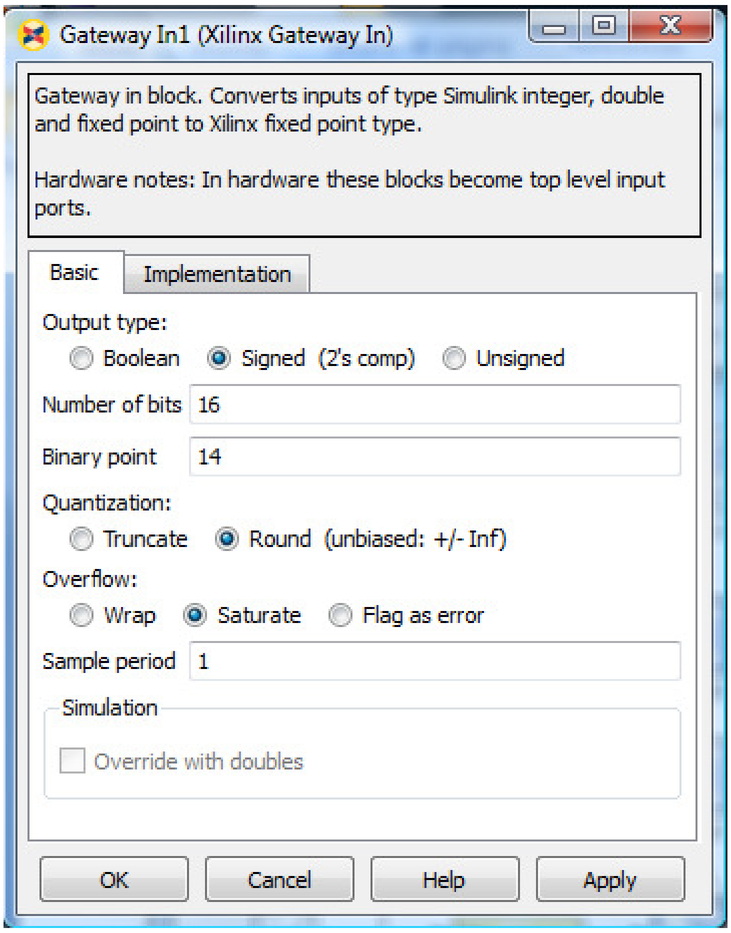
Compilation Options
Compilation : HDL Netlist
Part : Spartan6 xc6slx45t-4csg324
Target directory : ./netlist
Synthesis tool : XST
Hardware description language : VHDL
Create testbench
Import as configurable subsystem

Clocking Options
FPGA clock period (ns) : 10
Clock pin location :
Multirate implementation : Clock Enables
DCM input clock period (ns) : 10
Provide clock enable clear pin

Override with doubles : According to Block Settings
Simulink system period (sec) : 1
Block icon display: Default

Figure 16. System Generator Dialog.

Despite the fact that working with System Generator blocks simplifies very much the logic design process, it is still necessary to take care of the bus data types and sizes in addition to the sample time. These parameters are what enables system generator to instantiate the IPs and to manage the clock and data synchronization trough modules. Since some IPs expect specific data types and the sample time may change in some blocks. Double clicking on some blocks that have the ability to change those parameters allows editing the output of those blocks, as seen in the dialog of Figure 17. Nevertheless, the input is generally fixed, and the output is automatically set by the bloc; therefore, special blocks are provided to perform sample time and data type conversions.


Gateway In1 (Xilinx Gateway In)

Gateway in block. Converts inputs of type Simulink integer, double and fixed point to Xilinx fixed point type.

Hardware notes: In hardware these blocks become top level input ports.

Output type: Boolean [✓ Signed (2's comp)] Unsigned

Number of bits: 16

Binary point: 14

Quantization: Truncate [✓ Round (unbiased: +/- Inf)]

Overflow: Wrap [✓ Saturate] Flag as error

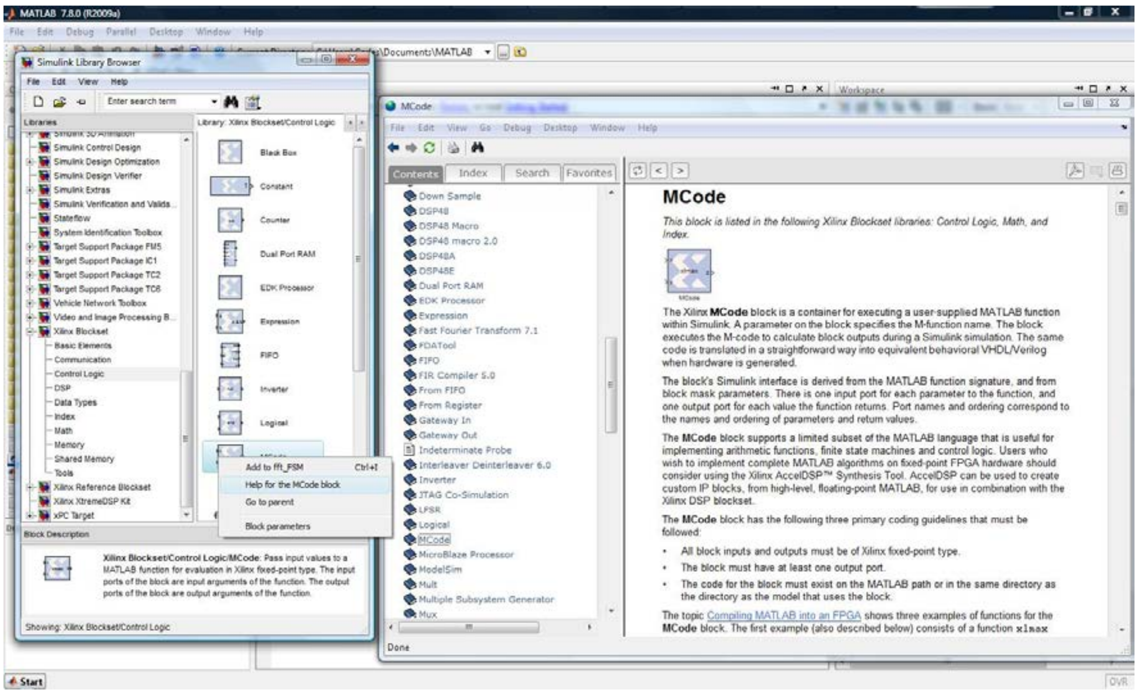
Sample period: 1

Simulation

☐ Override with doubles

Figure 17. Bus type edit dialog.

Additionally to the descriptions given in this thesis, Xilinx System Generator has a very complete built in help and block description, since this work does not pretend to substitute this source of information, whenever needed it is possible to obtain this type of information right clicking over the desired block in the Simulink Block Library or in the design and clicking at the help option of the menu; as seen in Figure 18. Moreover, from the System Generator menu all the Xilinx blocks can be consulted.



Block Description

Xilinx Blockset/Control Logic/MCode: Pass input values to a MATLAB function for evaluation in Xilinx fixed-point type. The input ports of the block are input arguments of the function. The output ports of the block are output arguments of the function.

Showing: Xilinx Blockset/Control Logic

MCode

This block is listed in the following Xilinx Blockset libraries: Control Logic, Math, and Index.

The Xilinx **MCode** block is a container for executing a user-supplied MATLAB function within Simulink. A parameter on the block specifies the M-function name. The block executes the M-code to calculate block outputs during a Simulink simulation. The same code is translated in a straightforward way into equivalent behavioral VHDL/Verilog when hardware is generated.

The block's Simulink interface is derived from the MATLAB function signature, and from block mask parameters. There is one input port for each parameter to the function, and one output port for each value the function returns. Port names and ordering correspond to the names and ordering of parameters and return values.

The **MCode** block supports a limited subset of the MATLAB language that is useful for implementing arithmetic functions, finite state machines and control logic. Users who wish to implement complete MATLAB algorithms on fixed-point FPGA hardware should consider using the Xilinx AccelDSP™ Synthesis Tool. AccelDSP can be used to create custom IP blocks, from high-level, floating-point MATLAB, for use in combination with the Xilinx DSP blockset.

The **MCode** block has the following three primary coding guidelines that must be followed:

- All block inputs and outputs must be of Xilinx fixed-point type.
- The block must have at least one output port.
- The code for the block must exist on the MATLAB path or in the same directory as the directory as the model that uses the block.

The topic Compiling MATLAB into an FPGA shows three examples of functions for the **MCode** block. The first example (also described below) consists of a function `xlmax`

Figure 18. System Generator Blocks Help.

2.4.2 – Matlab Simulink System Generator facts

When designing with System Generator toolset, some questions may arise on how to resolve common design aspects or about the System Generator environment itself. In order to aid solving these questions this section gives some explanations to the most common ones.

2.4.2.1 – What does the green System Generator blocks mean?

These blocks, in the current version of the Simulink Library, are formed by complex encode and decode modules which are licensed cores, available for purchase on the Xilinx website. Some of them have limitations related to supported devices where they may be implemented.

2.4.2.2 – What does the brown System Generator blocks mean?

These blocks allow the input or output of signals into the instantiable design, logically separating the compilable part by the System Generator of the design, from other blocks of the Matlab Simulink Libraries. Some of these blocks are the Input, the Output or the Signal Probe.

2.4.2.3 – Can m-code files be added to a design made with System Generator?

m-code blocks can be added with the Xilinx special block called M-Code Block which allows the association of an .m file to de block. This block generates the necessary connections to enable the interconnection of the M-Code Block with the rest of the design.

2.4.2.4 – Can VHDL or Verilog code be added to a design made with System Generator?

A self contained piece of VHDL or Verilog code may be added to a Simulink System Generator design with the Black Box Block. Once the project has been saved to a directory, the VHDL or Verilog code file has to be copied to the project directory root. The third step to be performed is to add a Black Box Block to the design, which will open a window to select the desired VHDL or Verilog file. After the selection is made the block will automatically generate the necessary ports in order to allow the interconnection of the code input and output signals with the rest of the design.

The Black Box Block does not correctly compile VHDL entities with generic values, this issue may be solved by manually changing, with the search and replace tool of a text editor, the generic variable name descriptor for the desired value and removing the generic declaration from the entity declaration.

2.4.2.5 – Can common Matlab Simulink blocks be synthesized by System Generator?

In general, Simulink blocks may be added to a design for simulation purposes; nevertheless, they cannot be mapped to hardware by the System Generator. However, there are some blocks which are fully supported and can be mapped to a hardware device which are the Demux block, the From block, the Goto Block and the Mux Block.

2.4.2.6 – Can mixed simulation with Simulink and ModelSim be performed?

The ModelSim block allows simulating the design using ModelSim after a simulation made with Simulink is performed; this feature may be useful in order to compare the results between the Simulink design and the resulting implementation.

In addition, the Create Testbench option may be selected when opening the System Generator, this will create at compile time an HDL testbench and the necessary script files for ModelSim. This is a very useful feature that simplifies verification process of the generated hardware. Furthermore, the files generated by the System Generator may be modified to allow specific hardware tests.

2.4.2.7 – What are the Xilinx Simulink blocks and how are they implemented?

They mainly implement Xilinx IPs. The Xilinx DSP blockset for Simulink is provided by over 90 Xilinx DSP building blocks. They include a wide variety of building blocks in order to allow the implementation of almost any design. Among these blocks there can be found from simple bocks such as adders, multipliers and multiplexers; to far more complex blocks such as FFTs, memories and forward correction blocks. All these blocks are implemented using the Xilinx IP core generators which intend to deliver optimized hardware implementations for a selected device.

2.4.2.8 – What is the block Sample Rate?

The block Sample Rate is the sample rate in which data arrives to a given block, this is a configuration parameter and must be correctly set in order to allow a correct design implementation. The typical sample rate is 1, therefore it must be the lowest sample rate of the design, if there are some parts of the design which work at higher sample rates it has to be correctly set at each block.

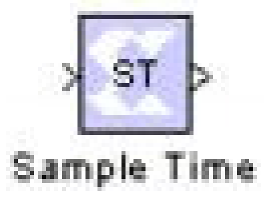


Figure 19. Sample time block.

The Sample Time Block, which may be seen at Figure 19, reports the normalized sample period of its input. This sample period used by the Xilinx blocks is not equivalent to the Simulink sample period which is used for simulation purposes, the sample period of the Xilinx blocks is implemented as a hardware constant instead. Special attention must be taken at the Input and Output Blocks of the Xilinx System Generator design to grant that its sample rate is equivalent to the minimum multiple of the design sample rates when blocks that change the sample rate are used, such as DUCs or DDCs.

In order to visualize the different sample rates of a design the following techniques may be applied. In one hand, the sample period may be seen configuring the Simulink window options available at *format -> sample time display*. In the other hand it may be seen implementing a Sample Time display block, with no hardware cost- such as the one in Figure 20 which is implemented with a Sample Time block, a Xilinx Out Block and a standard Simulink Display.

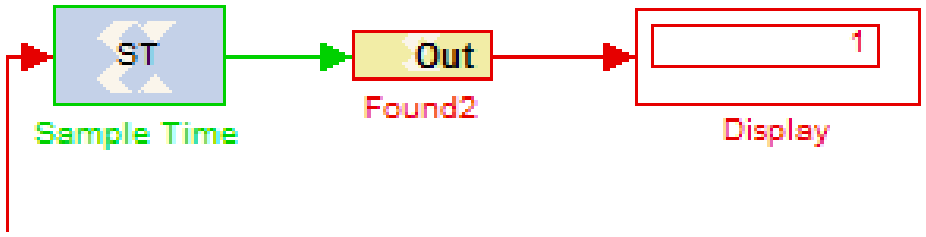


Figure 20. Sample Time view block.

2.4.2.9 – C/C++ support in Xilinx System Generator.

There is no block in the Xilinx System Generator that allows to directly import C/C++ functions to be translated in Hardware. Nevertheless, it is possible to use C/C++ code in a Xilinx FPGA implementing a Mocroblaze or a Picoblaze block, which may run C/C++ compiled programs.

2.4.2.10 – Picoblaze support

The 8 bit Xilinx microcontroller, which is implemented with basic building blocks may be instantiated in the system generator using the block called Picoblaze microcontroller. The System Generator environment handles some of the basic connection and configuration issues of the Picoblaze Microcontroller, which may simplify its use.

3.4.2.11 – Microblaze support

The small Xilinx microprocessor Microblaze can be instantiated with the block located at *Xilinx Blockset -> Control Logic, EDK Processor*. In order to allow its operation, shared memory blocks or shared FIFO blocks must also be instantiated in order to allow data exchange. In this case, Simulink environment will also handle some of the interconnection issues of the microprocessor, aiding in with its setup.

2.4.2.12 – FIR filter design with FDA tool and FIR compiler

The FDA Tool block works altogether with the FIR Compiler block. Once the FDA Tool block is placed in the design, double clicking on it opens an FDA Tool window where the desired filter response can be configured; a small preview of the filter response will be shown over the FDA Tool block.

In order to configure the FIR Compiler from an FDA Tool block the desired filter has to be set using the FDA Tool with a given precision; after this is done the option: File -> Export… has to be used in order to export the filter coefficients. In the Export dialog the variable name of the coefficients has to be set and it is recommended to enable the option Overwrite Variables, as it can be seen in Figure 21.

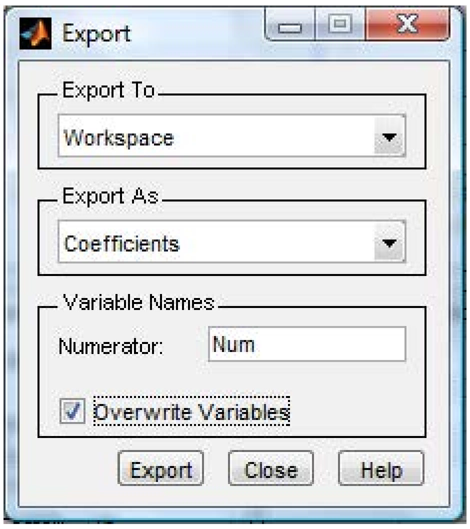
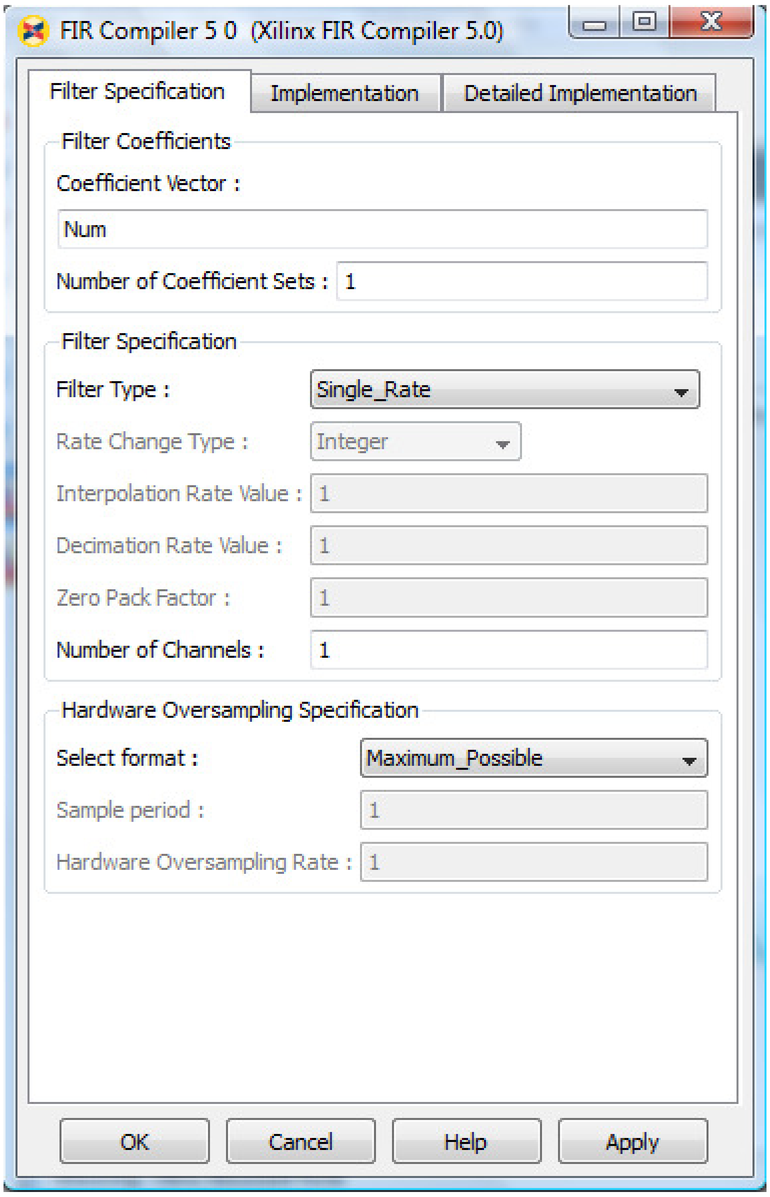

Export
Export To: Workspace
Export As: Coefficients
Variable Names
Numerator: Num
[✓] Overwrite Variables

Figure 21. FDA Tool Export dialog.

Regarding to associate the FIR Compiler and the FDA Tool filter the same name has to be used for the Variable Name, in the FDA Tool export dialog, and Coeficient Vector, in the FIR Compiler. As the variable used resides in the Matlab variables data space, each time the design is load the variable has to be exported anew from the FDA Tool.

There are some adjustments that have to be done in the FIR Compiler configuration dialog, which can be seen in Figure 22, in order to handle the FDA Tool input. In the Implementation Tab it is also necessary to change the Quantization Option to Maximize_Dynamic_Range, Filter architecture to Systolic_Multiply_Accumulate, Coefficient structure to Inferred, coefficient type to Signed, Quantization to Maximize_Dynamic_Range, Coefficient width to 16, Select Best Precision Fraction Length, Number of paths to 1 and finally Output rounding mode to Full_Precision.

There are many configuration options available for the FDA Tool and the Fir Compiler, and the correct function of a FIR setup may involve modifying also the Sample Rate of its input and form the main System Generator option, nevertheless these are the main steps to test a basic filter.


FIR Compiler 5 0 (Xilinx FIR Compiler 5.0)

Filter Coefficients

Coefficient Vector : Num

Number of Coefficient Sets : 1

Filter Specification

Filter Type : Single_Rate

Rate Change Type : Integer

Interpolation Rate Value : 1

Decimation Rate Value : 1

Zero Pack Factor : 1

Number of Channels : 1

Hardware Oversampling Specification

Select format : Maximum_Possible

Sample period : 1

Hardware Oversampling Rate : 1

Figure 22. FIR Compiler configuration dialog.

2.4.2.13 – Interesting utilities and the Xilinx Toolbar

The Xilinx Toolbar has some complements useful when designing with Simulink blocks; it allows block sorting and port termination of unused ports. Port termination is required when using entities whose ports are not used completely allowing to compile the design. Whenever the terminators are used as inputs, sometimes it is necessary to fix some the terminations setting the correct data types.

2.4.2.14 – Xilinx special Simulink data types

There are some special data types used in Xilinx System Generator Simulink designs, this special data types have to be taken in account when programming m-code functions in this environment.

Xilinx fixed-point data types:

xlUnsigned (<word length>, <binaty piont>)

xlSigned (<word lenth>, <binary point>)

Xilinx state machine variable:

xl_state (init, precision)

Init: is the initial value of the register

Precision: is a Xilinx fixed-point data type.

2.4.2.15 – Bus management and configuration

The Xilinx Simulink buses may be Boolean, signed and unsigned. Boolean buses are simple bit connections, where no further bus configuration is needed. In the other hand, for signed and unsigned buses it is necessary to configure the binary point value, which defines the number of fractional bits that the bus represents.

As an example- in Figure 23, a fixed precision 16 bits bus with the binary point value set to 14 is shown. In order to represent integer values in a Xilinx bus the binary point must be set to 0.

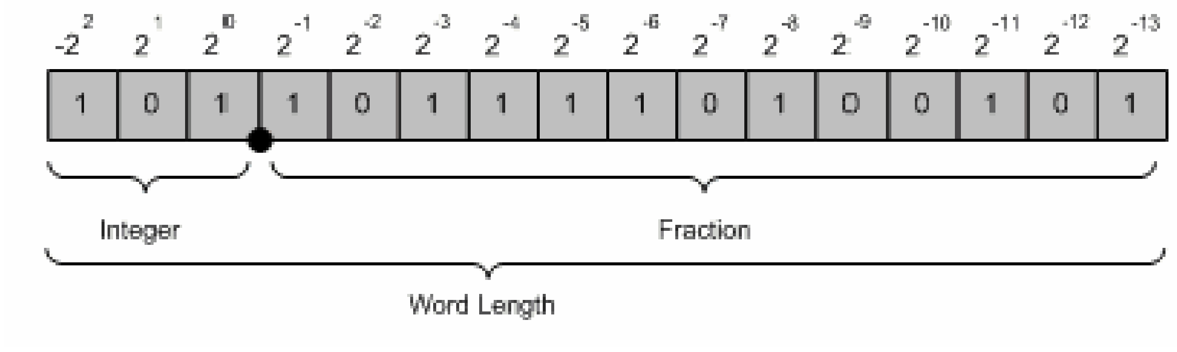


Figure 23. Xilinx bus distribution and nomenclature.

Xilinx System generator has a collection of blocks to aid interconnection between different bus sizes and data types. They are necessary in most designs, due to the automatic handling of inputs and outputs of the blocks and the fact that some blocks have restrictions in the data types and bus widths of their inputs.

The most commonly used blocks to perform bus conversion operations are:

Scale block: It is used to configure the way bus is scaled and the resulting data format.

Slice block: The Slice block is used to get a range of bits from the input bus and to assign them a data type for the exit.

Concat block: The Concat block is the opposite of the slice block; it is used to mix two buses into one, allowing some basic configuration of its output format.

Convert block: The convert block is one of the most useful blocks to adapt buses, in order to configure the convert block- the output format has to be set and the block adapts the data to match this format. Nevertheless, the default behaviour of the convert block is to shift the data to match the output format, in order to adapt the binary point of a bus, the Reinterpret block is recommended, since no data will be lost.

Reinterpret block: It is used to achieve similar results compared to the ones obtained with the Convert block; in contrast to the convert block, this type of conversion does not modify the data of the bus in any way, It just displaces de coma or changes the type without modificating the data itself.

BitBasher block: It is the most versatile of the bus casting blocks, it interprets Verilog code lines and allows more complex and exact conversion operations.

3 - Design

3.1 - General Design

The design finally implemented for this thesis aims to be a prove of concept of the design methodology presented in it, the System Generator design is a peripheral which can work alone or integrated in a SOC system using an EDK processor and the Microblaze bus interconnection.

This type of design was chosen due to the fact that it was the most versatile of all the available options, since it enables the System Generator design to be directly interconnected to the FPGA pins and it also provides a standard interface to allow a higher level of abstraction.

The function performed is a FFT, it is constructed with a FFT IP and a FSM which manages the control signals to make it work transparently from other modules as a FIFO, Figure 24. This structure can be used as a proof of concept for other signal processing designs with a higher degree of complexity.

Attached to the System Generator Design there is a Simulink simulation environment which helps to view the behaviour of the system in conjunction with the Simulink blocks enabling fast error correction. Nevertheless, it has been necessary to simulate the different parts of the design with ISim.

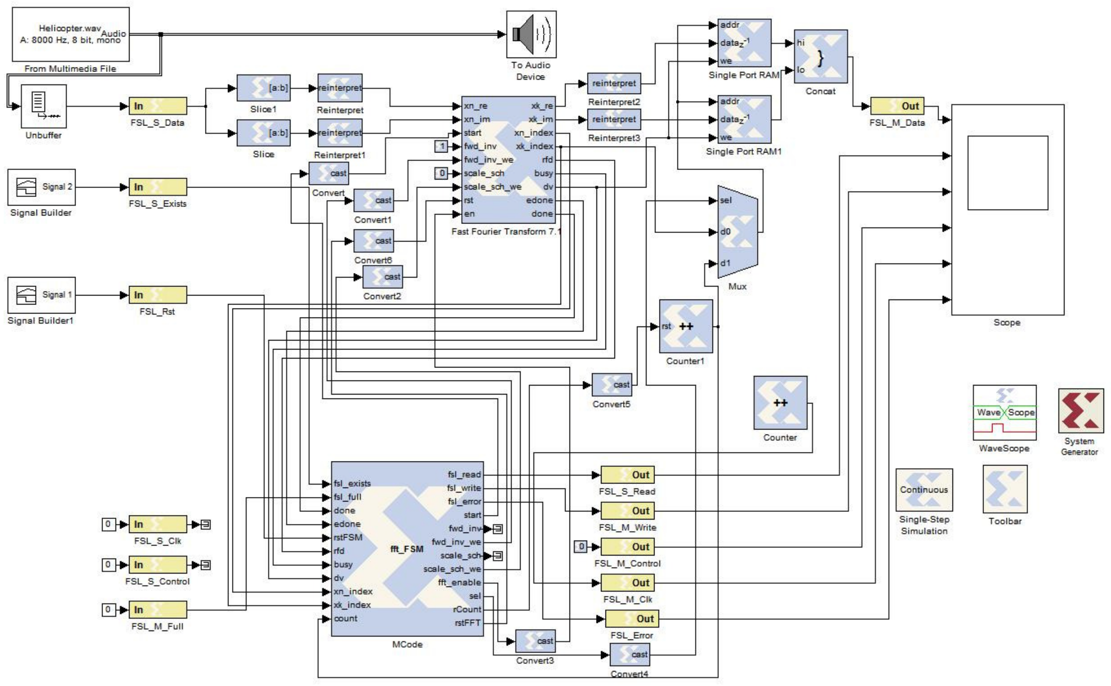


Figure 24. System Generator FFT peripheral design.

This peripheral incorporates the necessary logic to work with a FIFO interface; this has been achieved placing a demultiplexer memory at the output of the FFT since the data resulting from the IP is received disordered. The FSM manages all the necessary steps and signals to interface it with a FIFO. The descrambling section of the design can be seen in Figure 25; the FSM manages the selection bit of the multiplexer in the two steps that the data resulting from the FFT has to do, first kn_index is used to store the data resulting from the FFT to two 16 bits and 512 position RAMs; this data has been scaled by the reinterpret blocks to fit in the 16 bit real and imaginary output format. After the data is written in order to the RAMs, the select bit is changed by the FSM to use the counter to write the data sequentially to the FIFO.

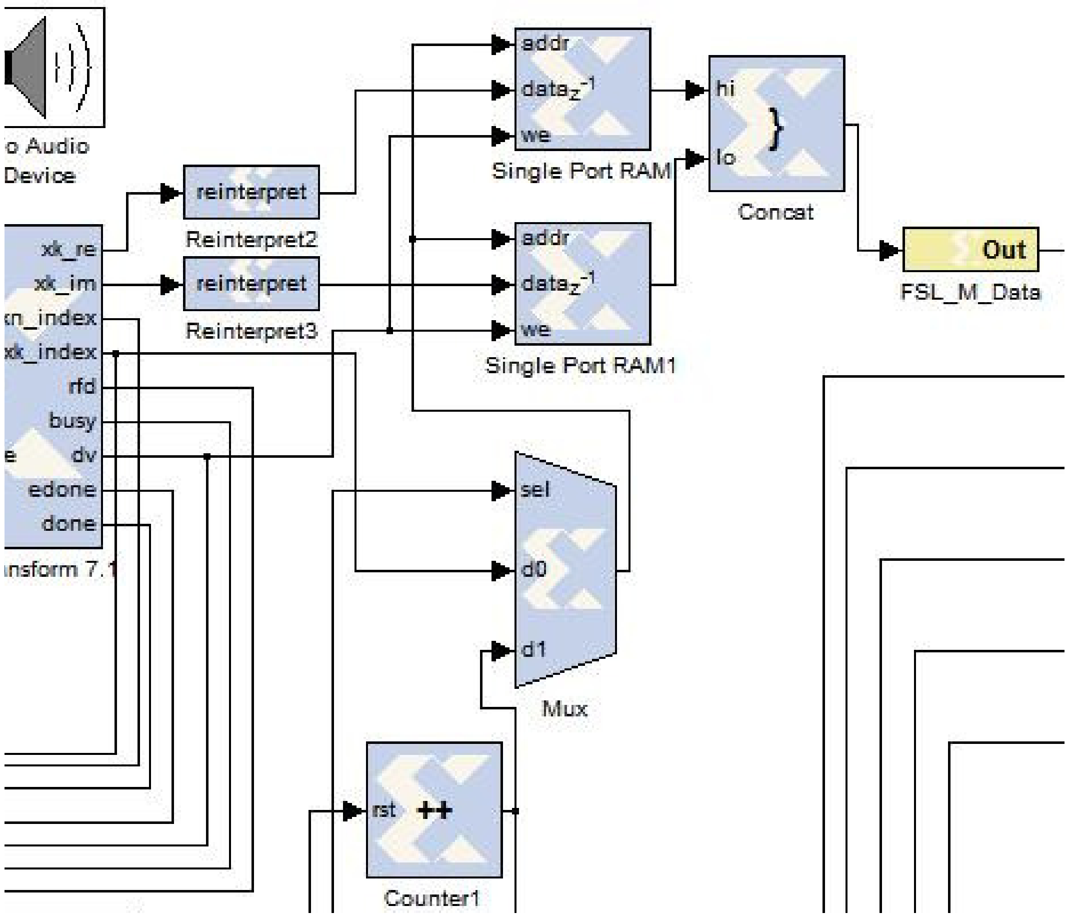


Figure 25. Descrambling section of the System Generator peripheral design.

The FFT Symbol can be seen in Figure 26, it is a radix-4 IP from the Xilinx IP repository. The most important signals are the start signal- which starts the FFT loading and unloading data process and FFT calculation; the rfd signal- which marks the cycles where the FFT is loading data; the dv signal- which identifies the cycles where the FFT is unloading data; and finally the done and edone which point out the cycle where the FFT finishes the calculation process and more data can be loaded.

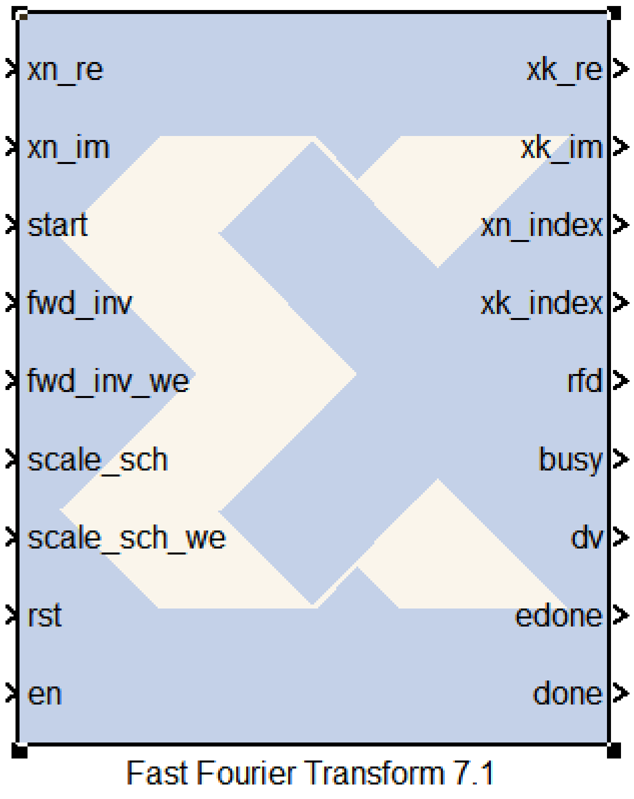


Figure 26. FFT block symbol from Xilinx System Generator.

The behaviour of the FFT has been deduced from Xilinx documentation **[19]** and from simulations, since some fields are outdated and the exact behaviour is not described in the documents or help of the FFT.

The FIFO interface is modelled after an FSL bus, which is supported by many Xilinx IPs, including the Microblaze microprocessor. This has been done this way since the FSL interface resembles very much a standard FIFO interface and it allows the interconnection with other peripherals transparently. This is achieved by renaming some of the pins that would have equally been used if the interface was a common FIFO and joining the 16 bits of the real and imaginary parts of the data into a 32 bits bus which is the FSL data bus.

3.2 – FSM implementation with Matlab

FSM design in Xilinx System generator is quite easy, since it can be done with a Matlab switch statement, in a way similar to other programming languages such as C++, Java or VisuaBasic.

A basic example of this type of design can be seen in Figure 27.

```
persistent state, state = xl_state(0,{xlUnsigned, 5, 0});

        switch state

                case 0

                        if din == 1
                                state = 1;
                        else
                                state = 0;
                        end

                case 1
                        matched = 0;
                        state = 0;
                otherwise
                        state = 0;
        END
```

Figure 27. Basic Matlab FSM example.

To design a FSM, it is necessary to include a special variable to allocate the current state, this is done with the *persistent state, state = xl_state(0,{xlUnsigned, 5, 0});* sentence; where state is the name of the variable, persistent is a Matlab keyword that enables the variable to hold its previous value between simulation steps and *xl_state(0,{xlUnsigned,5,0})* is a Xilinx defined variable of type unsigned and size 5 which acts as registered signal which will hold the current state. Nevertheless, it is quite easier to develop a FSM with this method. It is still necessary to define default values for the variables as it would be done in VHDL since inferred latches are not allowed.

A common issue that still have to be taken in account is that, despite the fact that abstract design may come very handy and seem easy, glitches generated by state transitions may still affect the general behaviour of the FPGA. Therefore, extensive simulation has to be done to observe signal propagation and stability. And in some cases the use of registers in the form: *var,var = xl_state...* or instantiated using System Generator blocks, may still be necessary.

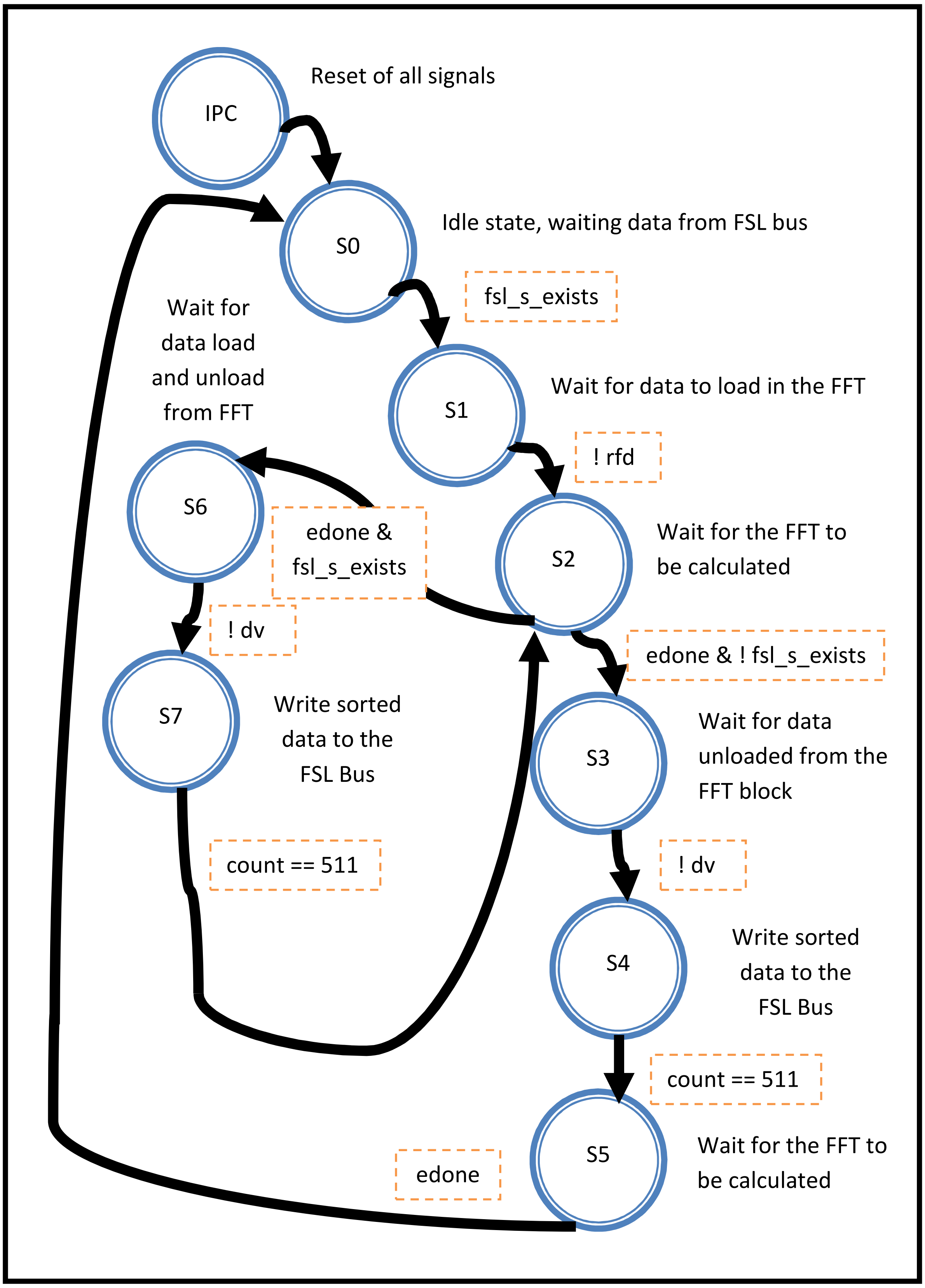

IPC
Reset of all signals
S0
Idle state, waiting data from FSL bus
fsl_s_exists
S1
Wait for data to load in the FFT
! rfd
S2
Wait for the FFT to be calculated
edone & ! fsl_s_exists
S3
Wait for data unloaded from the FFT block
! dv
S4
Write sorted data to the FSL Bus
count == 511
S5
Wait for the FFT to be calculated
edone
Wait for data load and unload from FFT
S6
edone & fsl_s_exists
! dv
S7
Write sorted data to the FSL Bus
count == 511

Figure 28. FFT peripheral FSM simplified representation.

In Figure 28 a simplified representation of the FSM used to control the FFT peripheral is shown, the FSM starts a transformation cycle whenever it detects data from the slave FSL bus, checking the fsl_s_exists signal. Once the data is detected it sets the start signal of the FFT to true for a cycle and starts unloading the data from the FSL bus and loading it to the FFT internal memory. Once this loading process is finished, the FFT block automatically starts the calculation of the transformation rising the edone and done signals to 1 when it finishes. At this state S2, the underlying characteristics of the radix-4 FFT bloc which starts simultaneously the loading and unloading of data and a new calculation cycle whenever the start cycle is set to 1 makes necessary to check if there is really new data waiting to be transformed, if this is the case fsl_s_exists will have a true value.

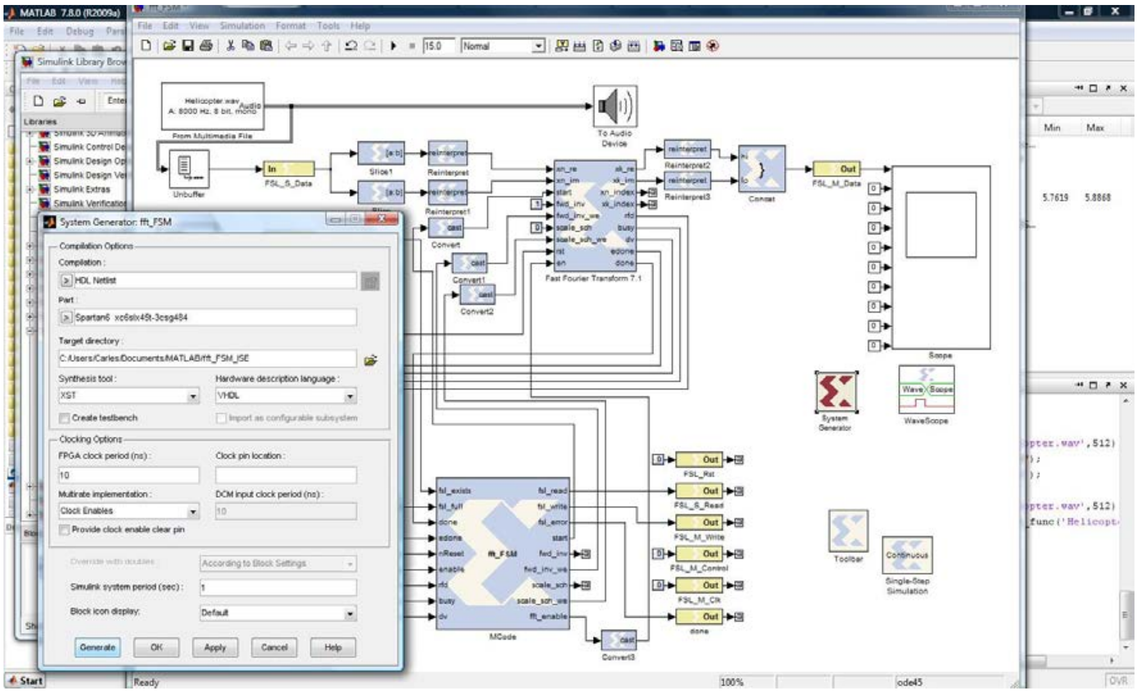
In the case that there is no data waiting to be processed the S3 state, the simple unloading path, will be chosen; it consists in waiting for the FFT to automatically write the resulting transformation data which come scrambled from the FFT to two intermediate RAMs, then a counter will be used to write it ordered to the FSL bus which the peripheral is master of and the flow returns to the idle state S0.

In the other case, when there is data waiting to be transformed the loading and unloading steps are made altogether; after the unloading process is done, the data is written into the FSL bus as it would be in the S4 state and, since all this process is finished before the calculation of the FFT is done, the execution flow returns to S2 to wait for a new FFT calculation to finish.

In the real FSM some of these steps require to be separated in smaller ones due to the peripheral dataflow characteristics, this implementation issues have been skipped since they make the understanding of the FSM more complex and is not meaningful for this study.

3.3 – Exporting procedure for Xilinx ISE

Once the design in System Generator is done, the System Generator block can be used to export de design to Xilinx ISE, once the System Generator Dialog is open; it is possible to configure all de relevant import options needed, Figure 29.


System Generator: fft_FSM

Compilation Options

Compilation : HDL Netlist

Part : Spartan6 xc6slx45t-3csg484

Target directory : C:/Users/Carles/Documents/MATLAB/fft_FSM_ISE

Synthesis tool : XST

Hardware description language : VHDL

Create testbench

Import as configurable subsystem

Clocking Options

FPGA clock period (ns) : 10

Clock pin location :

Multirate implementation : Clock Enables

DCM input clock period (ns) : 10

Provide clock enable clear pin

Override with doubles : According to Block Settings

Simulink system period (sec) : 1

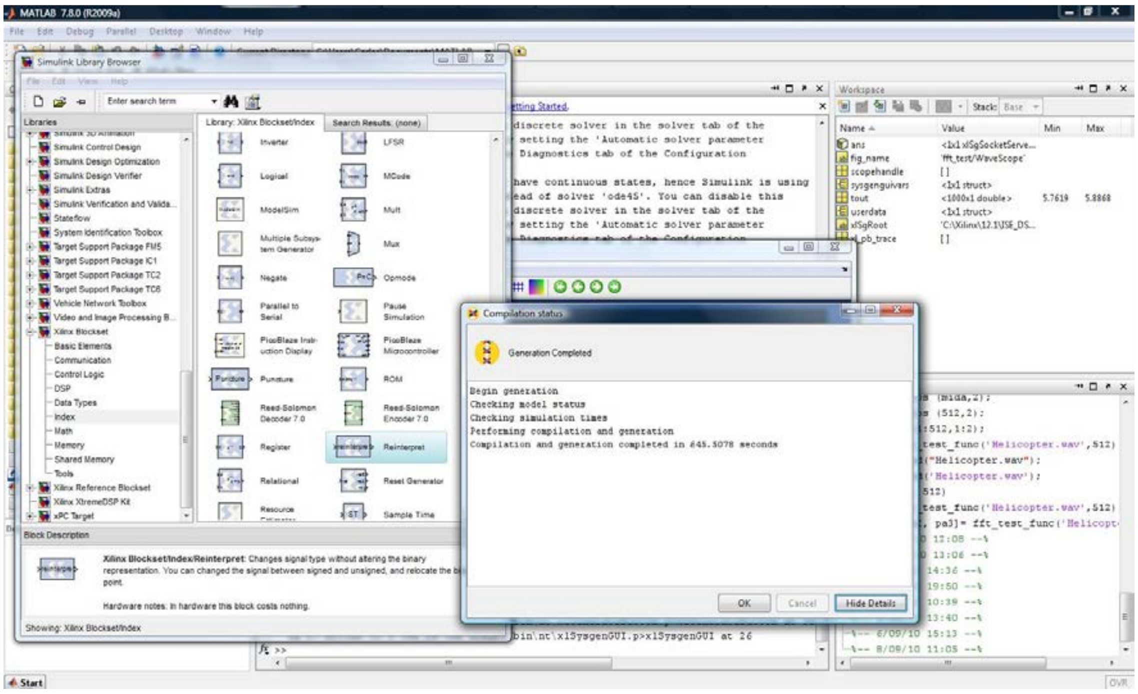
Block icon display: Default

Figure 29. System Generator Export Dialog.

It is convenient to choose that System Generator generates a VHDL netlist, since this type of exported files can be edited afterwards with Xilinx ISE. To correctly import the design to Xilinx ISE the correct FPGA has to be set and the language to represent the hardware, in this case a Spartan 6 and VHDL.

In some cases, it may be also desirable that System Generator exports the current Simulink test environment as a testbench since it can be useful to compare the simulation results and evaluate its correctness with the ISE simulator. Nevertheless, whenever that is done a bug may arise forbidding to simulate any other testbench generated with Xilinx ISE, that bug can be solved erasing the ISim folder located at the project root, which will allow to choose and simulate any of the available testbenches of the project.

When the information of the configuration dialog is properly set, the generation process can be started. As in the Simulink simulation process, the design is revised in this step and any design errors will arise. After a successful generation, a window like the one seen in Figure 30 will appear.



```
Begin generation
Checking model status
Checking simulation times
Performing compilation and generation
Compilation and generation completed in 645.5078 seconds
```

Figure 30. System Generator generation log after a successful generation.

Once the design generation has finished, a project for Xilinx ISE is stored in the target folder. There are three main ways to use the newly generated project in conjunction with other designs or tools:

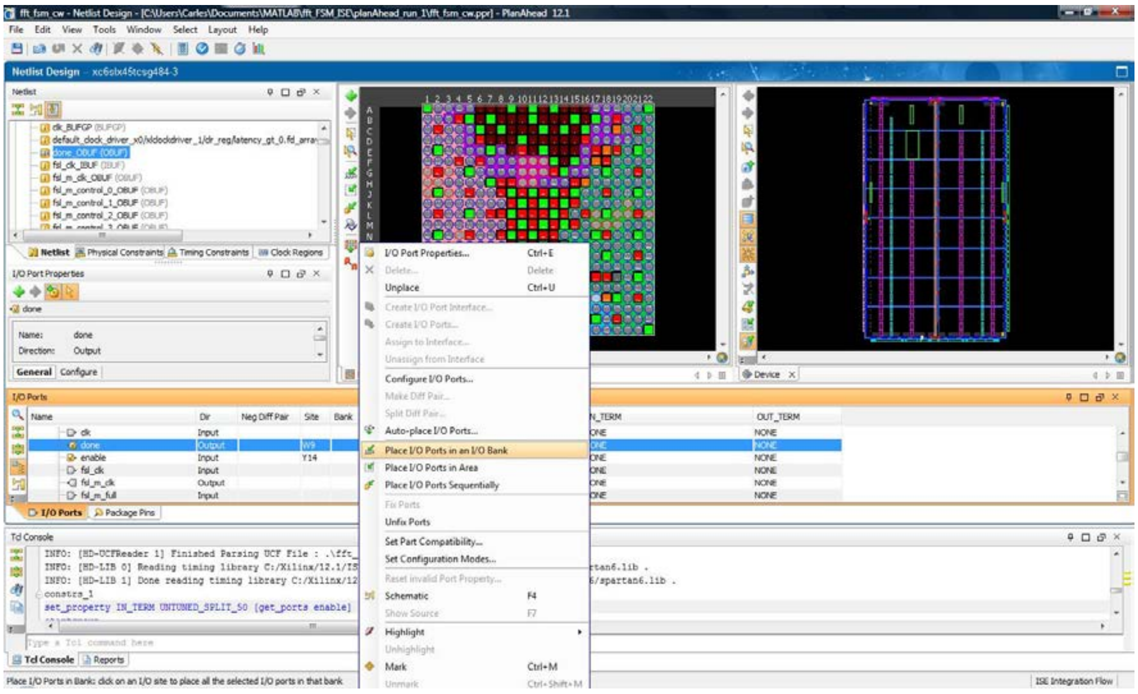
- The generated project can be opened with Xilinx ISE directly to perform tests and simulations, assign the pins to with Plan Ahead and finally generate a FPGA programming file of the design as it was designed with System Generator.
- The System Generator exported files may also be imported into an existing ISE project using the generated VHDL files- including them to another project and instantiating the entities as components.
- Finally, the most advanced way to integrate various design methodologies into an advanced SOC design with one or more microprocessors- is to generate an IP of the ISE project with the Xilinx Platform Studio to integrate the design as a peripheral; this type of solution can be seen in the next section.

Despite the design methodology chosen, it may be always useful to open the System Generator generated project with Xilinx ISE to test the design behaviour individually, performing the required test with the Simulation tool.

In the case that any modification is required it is not recommended to edit the System Generator generated files since they are very complex, for that case the second design methodology would be advised.

Finally, to generate the programming file with Xilinx ISE- Plan Ahead has to be used to set the input and output pins, since there is no way to adjust the pins with System

Genretor. To manually set each pin to an output the option of the right click menu place I/O Ports in a I/O bank has to be selected and then select the desired I/O pin, as seen in Figure 31.


fft_fsm_cw - Netlist Design - [C:\Users\Carles\Documents\MATLAB\fft_FSM_ISE\planAhead_run_1\fft_fsm_cw.ppr] - PlanAhead 12.1

Netlist Design - xc6slx45tcsg484-3

Name: done
Direction: Output

Name	Dir	Neg Diff Pair	Site	Bank	N_TERM	OUT_TERM
clk	Input				ONE	NONE
done	Output		W9		ONE	NONE
enable	Input		Y14		ONE	NONE
fsl_clk	Input				ONE	NONE
fsl_m_clk	Output				ONE	NONE
fsl_m_full	Input				ONE	NONE

```
INFO: [HD-UCFReader 1] Finished Parsing UCF File : .\fft_
INFO: [HD-LIB 0] Reading timing library C:/Xilinx/12.1/IS                                       rtan6.lib .
INFO: [HD-LIB 1] Done reading timing library C:/Xilinx/12                                        6/spartan6.lib .
constrs_1
set_property IN_TERM UNTUNED_SPLIT_50 [get_ports enable]
```

Place I/O Ports in Bank: click on an I/O site to place all the selected I/O ports in that bank

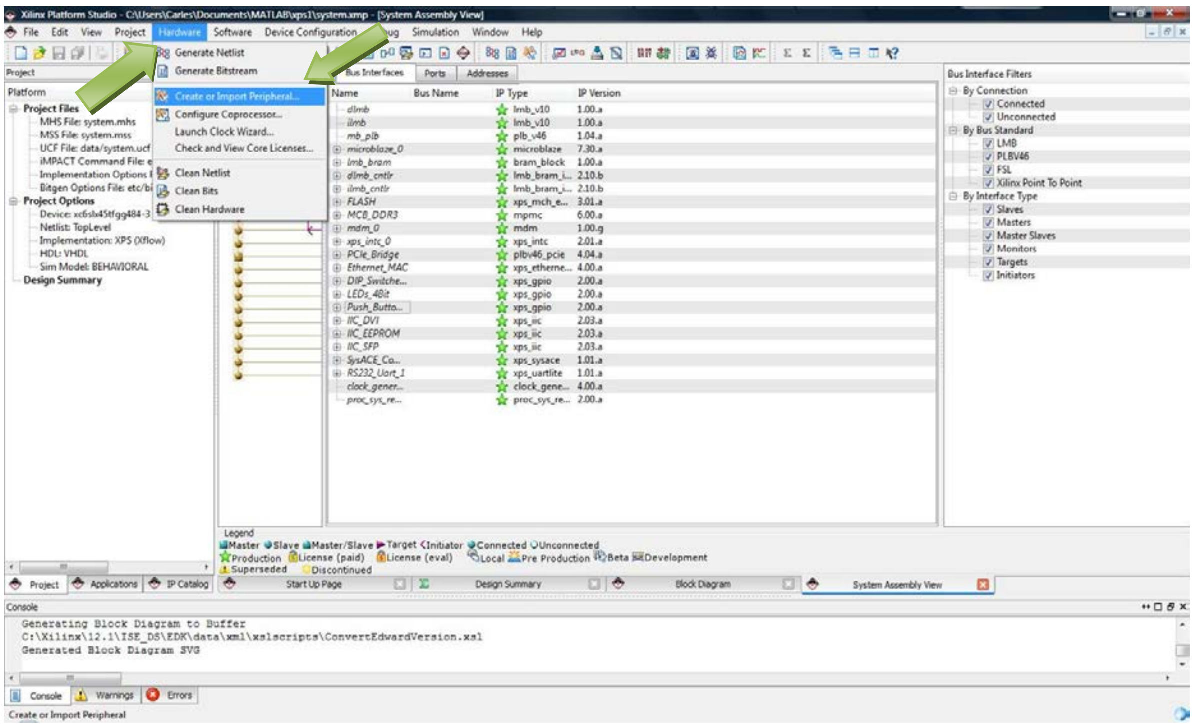
Figure 31. Plan Ahead pinout selection.

3.4 – Exporting procedure for Xilinx XPS

It is also possible to import the designs made with Xilinx System Generator in Matlab to the Xilinx Platform Studio, this allows access to the peripherals made with system generator from a microprocessor, enhancing the capabilities of the overall design allowing the creation of complete SOC solutions. Furthermore, if a default interconnection interface supported by the microprocessor is implemented in the module designed with Simulink- the access from the microprocessor can be done in a transparent form.

For this project, a default wizard generated project is used, XPS allows to create projects with one or multiple EDK Processors for the standard test boards built by Xilinx or other test boars built by 3rd party partners with the Base System Builder. Those standard XPS projects automatically configure all the peripherals of those boards simplifying the design process and supplying an error free base environment to test the main capabilities of the board **[17]**.

The first step, which can be seen in Figure 32, is to import the design into the Xilinx Platform Studio is to create a user defined IP, this can be performed by opening the hardware tab and choosing the Add or Import Peripheral option.

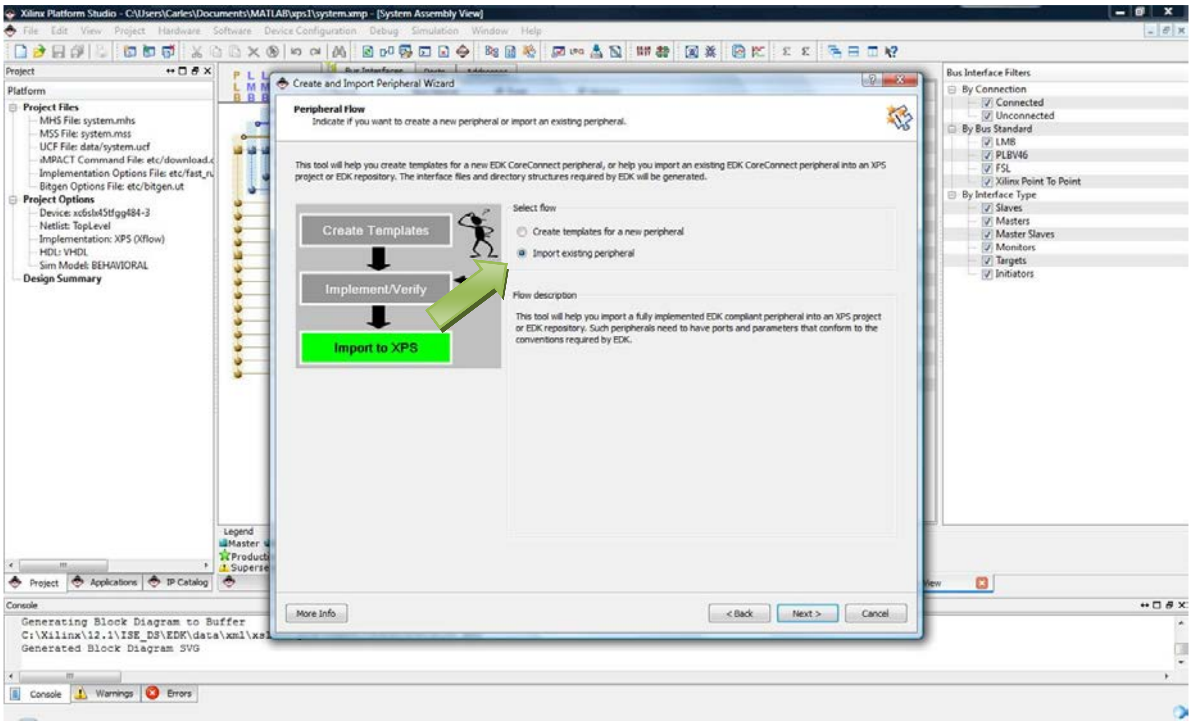


Name	Bus Name	IP Type	IP Version
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.04.a
microblaze_0		microblaze	7.30.a
lmb_bram		bram_block	1.00.a
dlmb_cntlr		lmb_bram_i...	2.10.b
ilmb_cntlr		lmb_bram_i...	2.10.b
FLASH		xps_mch_e...	3.01.a
MCB_DDR3		mpmc	6.00.a
mdm_0		mdm	1.00.g
xps_intc_0		xps_intc	2.01.a
PCIe_Bridge		plbv46_pcie	4.04.a
Ethernet_MAC		xps_etherne...	4.00.a
DIP_Switche...		xps_gpio	2.00.a
LEDs_4Bit		xps_gpio	2.00.a
Push_Butto...		xps_gpio	2.00.a
IIC_DVI		xps_iic	2.03.a
IIC_EEPROM		xps_iic	2.03.a
IIC_SFP		xps_iic	2.03.a
SysACE_Co...		xps_sysace	1.01.a
RS232_Uart_1		xps_uartlite	1.01.a
clock_gener...		clock_gene...	4.00.a
proc_sys_re...		proc_sys_re...	2.00.a

```
Generating Block Diagram to Buffer
C:\Xilinx\12.1\ISE_DS\EDK\data\xml\xslscripts\ConvertEdwardVersion.xsl
Generated Block Diagram SVG
```

Figure 32. XPS peripheral import.

The next step consists in choosing the Import Existing Peripheral option, as it can be seen in Figure 33.


Xilinx Platform Studio - C:\Users\Carles\Documents\MATLAB\xps1\system.xmp - [System Assembly View]

Create and Import Peripheral Wizard

Peripheral Flow
Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK CoreConnect peripheral, or help you import an existing EDK CoreConnect peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

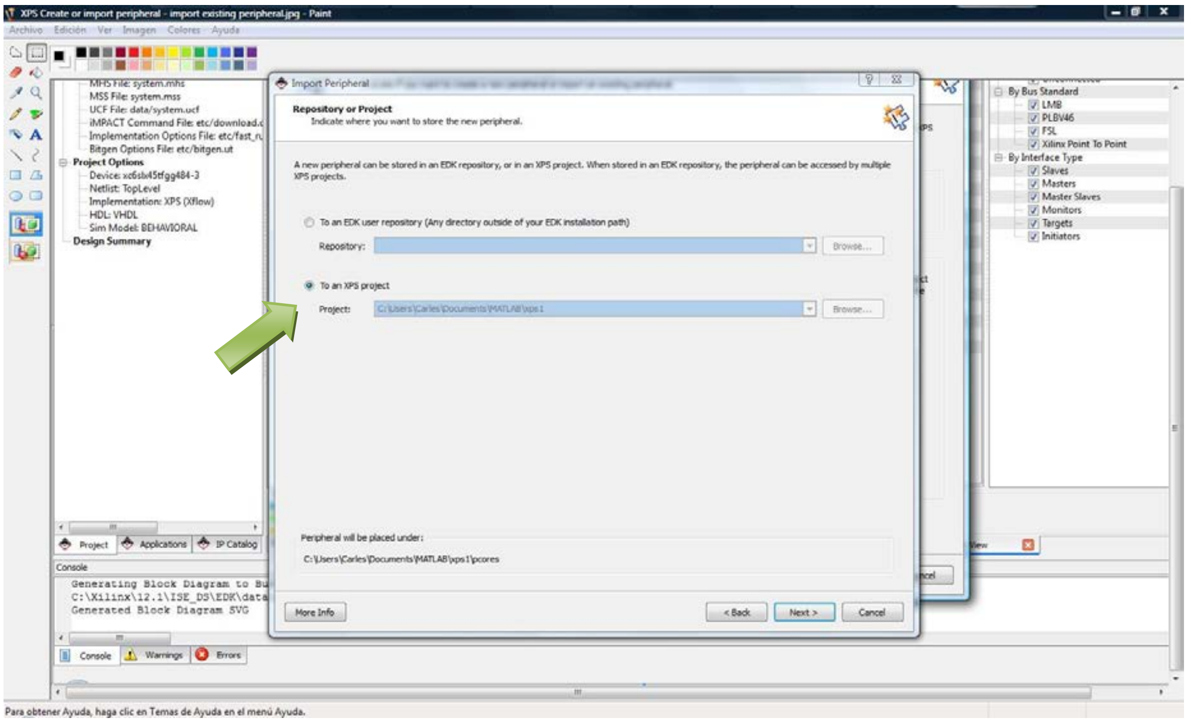
Select flow
- ( ) Create templates for a new peripheral
- (•) Import existing peripheral

Flow description
This tool will help you import a fully implemented EDK compliant peripheral into an XPS project or EDK repository. Such peripherals need to have ports and parameters that conform to the conventions required by EDK.

```
Generating Block Diagram to Buffer
C:\Xilinx\12.1\ISE_DS\EDK\data\xml\xsl
Generated Block Diagram SVG
```

Figure 33. XPS peripheral import- peripheral flow selection.

In the following step, the project where the new User defined IP must be created must be chosen, in this case the current project is used, as seen in Figure 34.


XPS Create or import peripheral - import existing peripheral.jpg - Paint

Import Peripheral

Repository or Project
Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

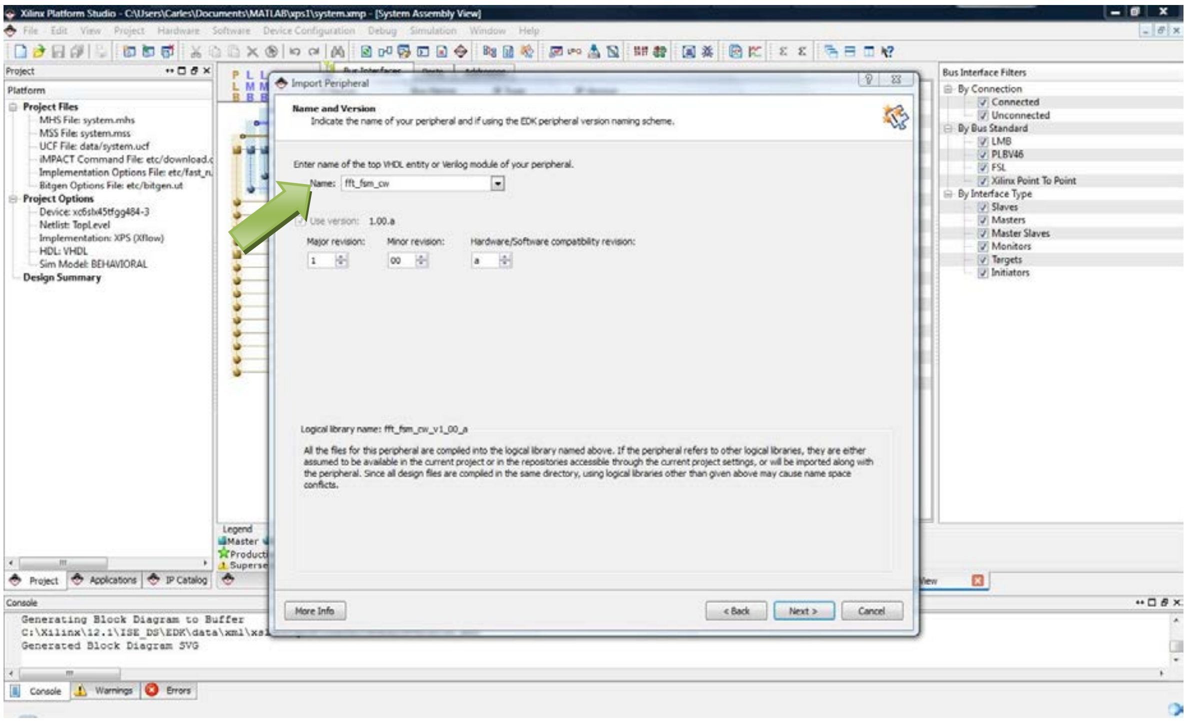
- ( ) To an EDK user repository (Any directory outside of your EDK installation path)
  Repository:
- (•) To an XPS project
  Project: C:\Users\Carles\Documents\MATLAB\xps1

Peripheral will be placed under:
C:\Users\Carles\Documents\MATLAB\xps1\pcores

Para obtener Ayuda, haga clic en Temas de Ayuda en el menú Ayuda.

Figure 34. XPS peripheral import- source type selection.

The next step consists in identifying the top level entity of the imported design, which must be previously known and it also allows indentifying the resulting IP with a revision number, as it can be seen in Figure 35.


Import Peripheral

Name and Version
Indicate the name of your peripheral and if using the EDK peripheral version naming scheme.

Enter name of the top VHDL entity or Verilog module of your peripheral.

Name: fft_fsm_cw

[✓] Use version: 1.00.a

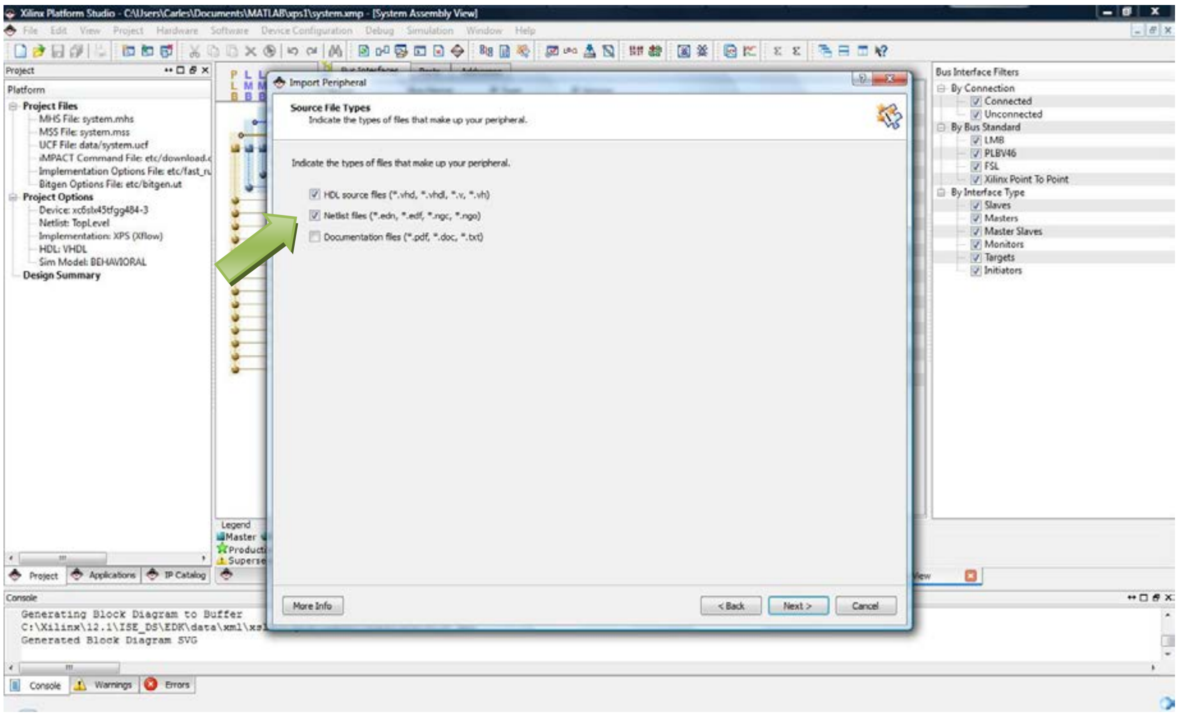
Major revision:	Minor revision:	Hardware/Software compatibility revision:
1	00	a

Logical library name: fft_fsm_cw_v1_00_a

All the files for this peripheral are compiled into the logical library named above. If the peripheral refers to other logical libraries, they are either assumed to be available in the current project or in the repositories accessible through the current project settings, or will be imported along with the peripheral. Since all design files are compiled in the same directory, using logical libraries other than given above may cause name space conflicts.

```
Generating Block Diagram to Buffer
C:\Xilinx\12.1\ISE_DS\EDK\data\xml\xsl
Generated Block Diagram SVG
```

Figure 35. XPS peripheral import- top level entity name selection.


Import Peripheral

Source File Types
Indicate the types of files that make up your peripheral.

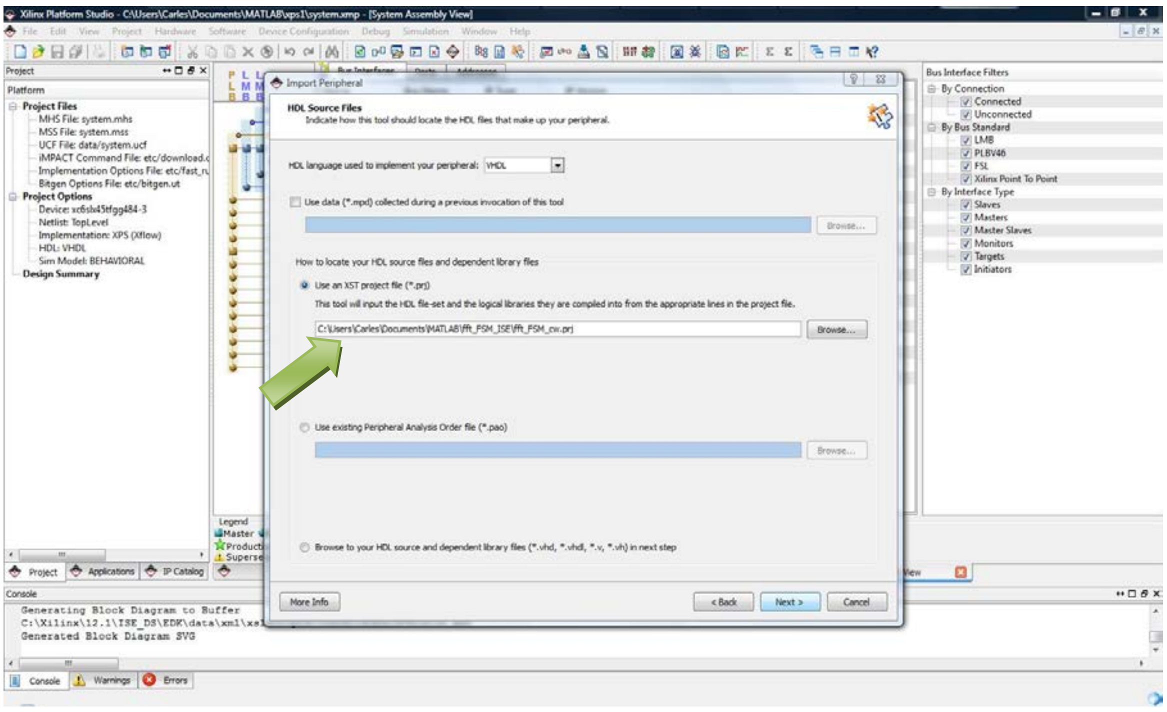
Indicate the types of files that make up your peripheral.

- [✓] HDL source files (*.vhd, *.vhdl, *.v, *.vh)
- [✓] Netlist files (*.edn, *.edf, *.ngc, *.ngo)
- [ ] Documentation files (*.pdf, *.doc, *.txt)

```
Generating Block Diagram to Buffer
C:\Xilinx\12.1\ISE_DS\EDK\data\xml\xsl
Generated Block Diagram SVG
```

Figure 36. XPS peripheral import- import files selection.

In the following step, shown in Figure 36, the type of files to import must be chosen, it is always a good advice to include the HDL and Netlist files. In the next step the ISE project from which the IP has to be created has to be selected with a file browser, as seen in Figure 37.



Xilinx Platform Studio - C:\Users\Carles\Documents\MATLAB\xps1\system.xmp - [System Assembly View]

Import Peripheral

HDL Source Files
Indicate how this tool should locate the HDL files that make up your peripheral.

HDL language used to implement your peripheral: VHDL

☐ Use data (*.mpd) collected during a previous invocation of this tool

How to locate your HDL source files and dependent library files

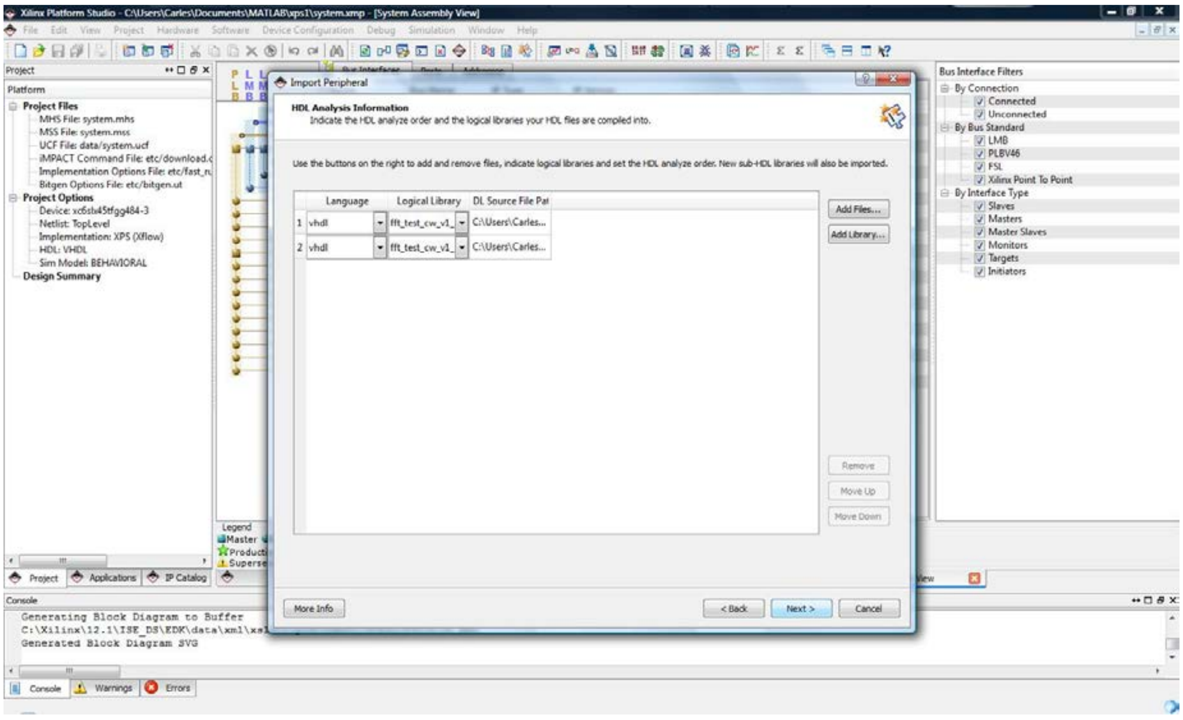
● Use an XST project file (*.prj)
This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines in the project file.
C:\Users\Carles\Documents\MATLAB\fft_FSM_ISE\fft_FSM_cw.prj

○ Use existing Peripheral Analysis Order file (*.pao)

○ Browse to your HDL source and dependent library files (*.vhd, *.vhdl, *.v, *.vh) in next step

```
Generating Block Diagram to Buffer
C:\Xilinx\12.1\ISE_DS\EDK\data\xml\xsl
Generated Block Diagram SVG
```

Figure 37. XPS peripheral import- ISE project location.



Xilinx Platform Studio - C:\Users\Carles\Documents\MATLAB\xps1\system.xmp - [System Assembly View]

Import Peripheral

HDL Analysis Information
Indicate the HDL analyze order and the logical libraries your HDL files are compiled into.

Use the buttons on the right to add and remove files, indicate logical libraries and set the HDL analyze order. New sub-HDL libraries will also be imported.

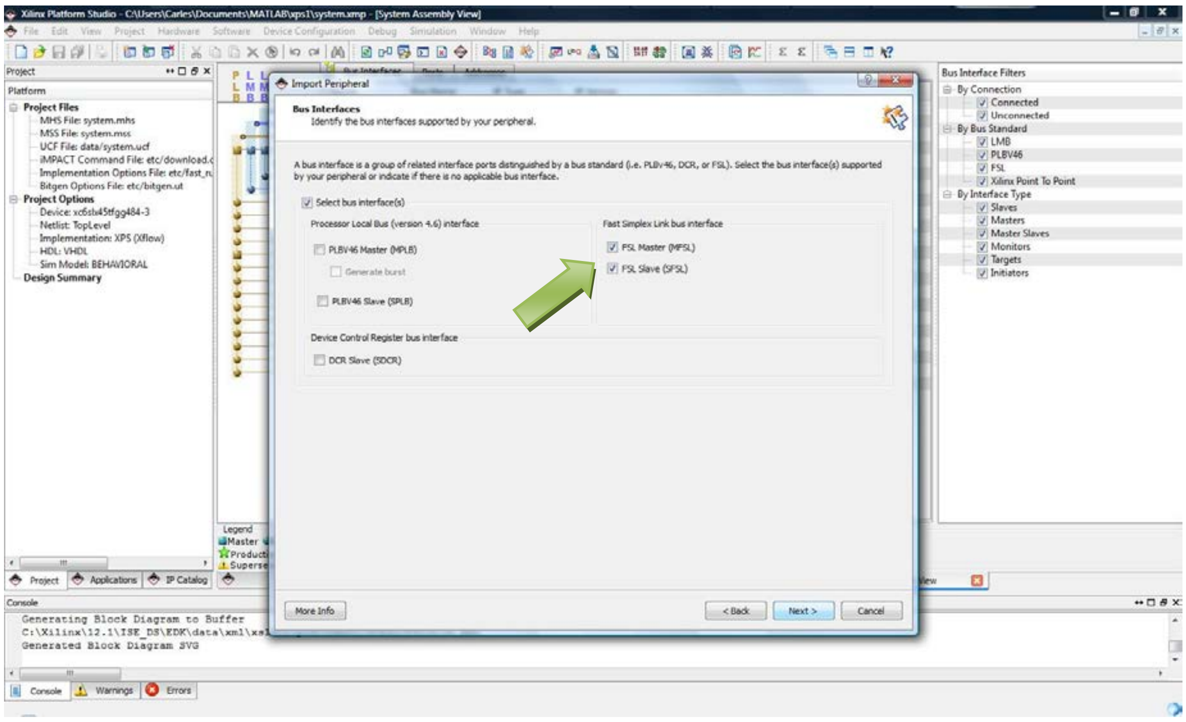
	Language	Logical Library	DL Source File Pat
1	vhdl	fft_test_cw_v1_	C:\Users\Carles...
2	vhdl	fft_test_cw_v1_	C:\Users\Carles...

```
Generating Block Diagram to Buffer
C:\Xilinx\12.1\ISE_DS\EDK\data\xml\xsl
Generated Block Diagram SVG
```

Figure 38. XPS peripheral import- VHDL sources selection.

For the following step, shown in Figure 38, in a project like the one used for this thesis the fields are fine and nothing has to be done.

Once arrived to the step where an interconnection between the peripheral and the EDK processor is required, as it can be seen in Figure 38, there are two main procedures preferred:



Import Peripheral

Bus Interfaces
Identify the bus interfaces supported by your peripheral.

A bus interface is a group of related interface ports distinguished by a bus standard (i.e. PLBv46, DCR, or FSL). Select the bus interface(s) supported by your peripheral or indicate if there is no applicable bus interface.

[✓] Select bus interface(s)

Processor Local Bus (version 4.6) interface
- [ ] PLBV46 Master (MPLB)
  - [ ] Generate burst
- [ ] PLBV46 Slave (SPLB)

Fast Simplex Link bus interface
- [✓] FSL Master (MFSL)
- [✓] FSL Slave (SFSL)

Device Control Register bus interface
- [ ] DCR Slave (SDCR)

```
Generating Block Diagram to Buffer
C:\Xilinx\12.1\ISE_DS\EDK\data\xml\xs
Generated Block Diagram SVG
```

Xilinx Platform Studio - C:\Users\Carles\Documents\MATLAB\xps1\system.xmp - [System Assembly View]
More Info
< Back
Next >
Cancel

Figure 39. XPS peripheral import- bus interface selection.

The first and most versatile one is performed using the PLB v46 bus standard, this bus is used by the peripherals imported by default and it is the main interconnection option used by Xilinx IPs and 3rd party IPs. This type of interface can be accessed transparently by the Xilinx microprocessors once an address space is assigned.

Despite that in some cases it is possible to implement, in a semiautomatic manner, an interface to the PLB bus using what is called a GPIO module provided by Xilinx, which is configurable and handles most of the interconnection issues; it could not be achieved to instantiate this module in this project and trying to implement the PLB bus interface by hand was too difficult- it has various transmission modes and, even for the slave side, many signals which the peripheral has to take care about **[16]**, as it can be seen in the Figure 40.

Since the scope of this project is not to achieve an advanced bus interconnection system for SOC peripherals, another simpler default interconnection method should be considered for simple modules.

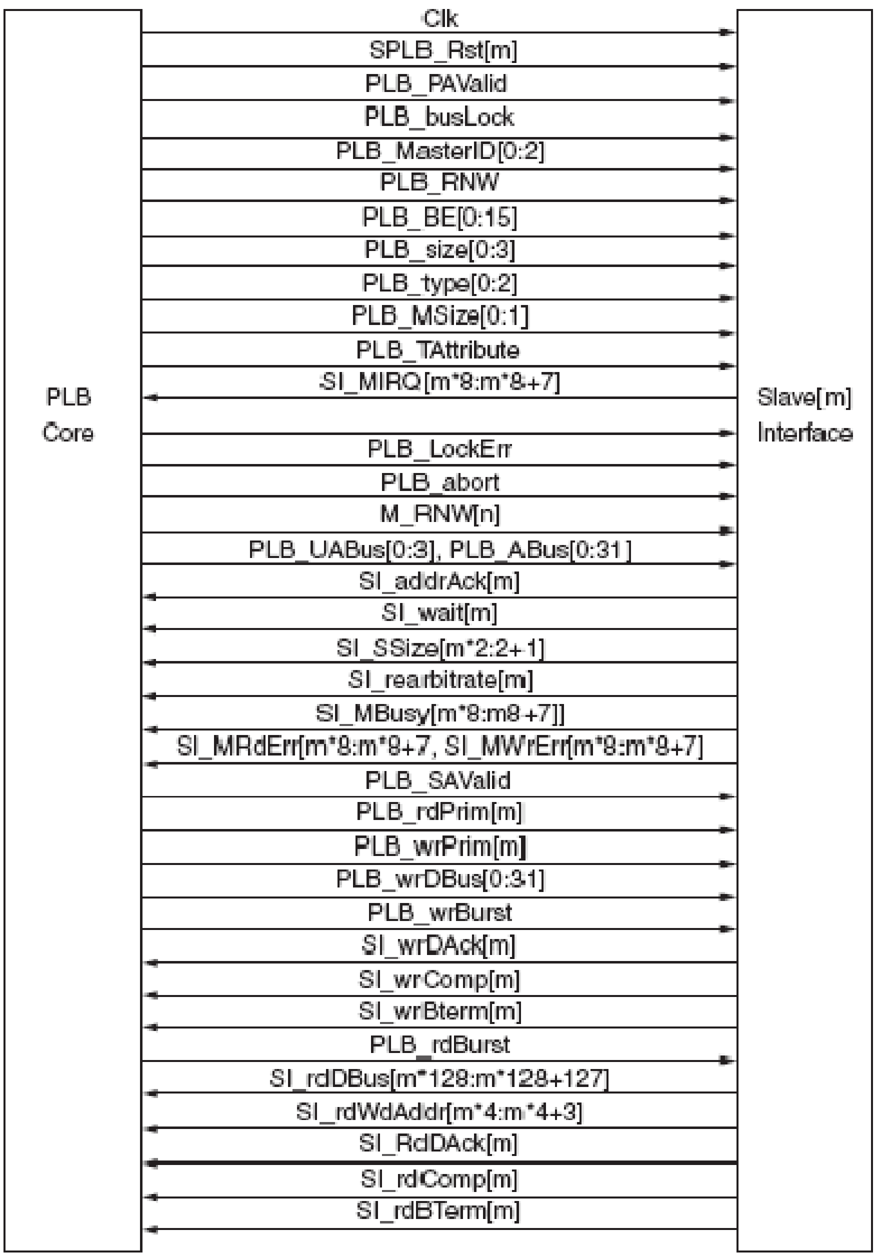


Figure 40. PLB v 46 bus slave interface.

The second preferred and much simpler interconnection mechanism is performed using the Fast Simplex Link V20 Bus. As illustrated in Figure 41, this bus is mainly a simplex FIFO buffer between a master, which writes to the FIFO, and a slave, which reads from the FIFO; therefore, to achieve a bidirectional communication channel two bus interfaces are required **[16]**. This bus is also easily accessible by a Microblaze microprocessor and its interface can be automatically imported during the Create Import peripheral process of the Xilinx Platform Studio environment if some design requirements are met.

The instructions for reading and writing from FSL bus can be blocking and non-blocking, and they have the following format **[1]**:

C Function Call	Assembly-Level Instruction	Description
Microblaze_bread_datafsl(val, id)	get	blocking data read
Microblaze_bwrite_datafsl(val, id)	put	blocking data write
Microblaze_nbread_datafsl(val, id)	nget	non-blocking data read
Microblaze_nbwrite_datafsl(val, id)	nput	non-blocking data write
Microblaze_bread_cntlfsl(val, id)	cget	blocking control read
Microblaze_bwrite_cntlfsl(val, id)	cput	blocking control write
Microblaze_nbread_cntlfsl(val, id)	ncget	non-blocking control read
Microblaze_nbwrite_cntlfsl(val, id)	ncput	non-blocking control write

Where val is the data to read/write which for the read operation it must be a variable and Id is the FSL port number starting from 0 set in the XPS bus interfaces tab.

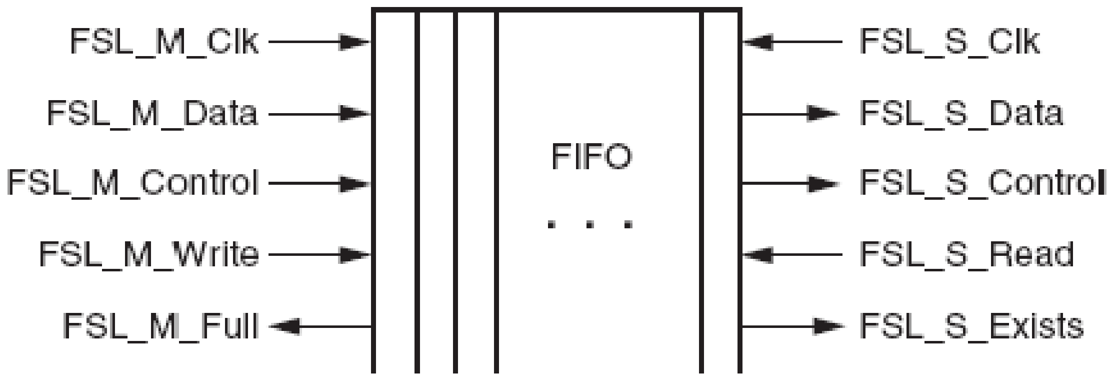


Figure 41. Fast Simplex Link(FSL) V20 Bus Block Diagram.

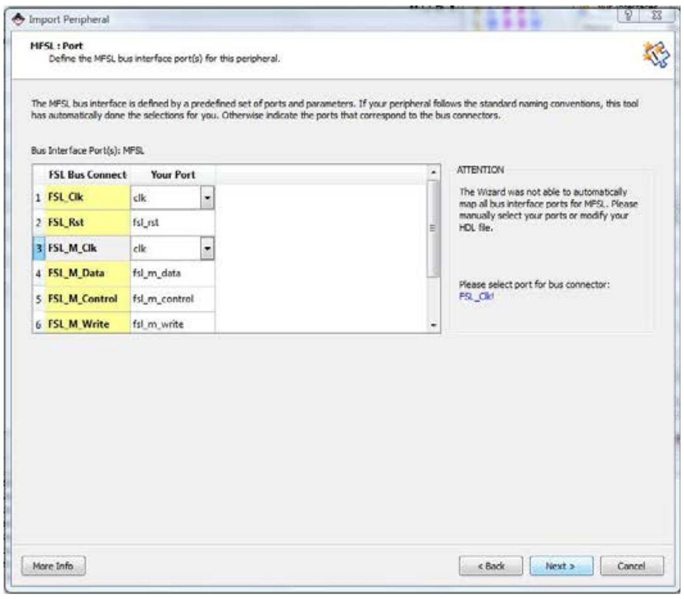
To be automatically imported, the bus signals of the top imported entity have to comply with the nomenclature shown in Figure 42.

Master side	Slave side
FSL_Clk	FSL_Clk
FSL_Rst	FSL_Rst
FSL_M_Clk	FSL_S_Clk
FSL_M_Data	FSL_S_Read
FSL_M_Control	FSL_S_Data
FSL_M_Write	FSL_S_Control
FSL_M_Full	FSL_S_Exists

Figure 42. FSLv20 nomenclature conventions.

The FSL bus allows synchronous and asynchronous communications; for the synchronous configuration the FSL_M_Clk, the FSL_S_Clk, the FSL_M_Control and the FSL_S_Control signals may be ignored leaving the interface to an even simplified form similar to a simple FIFO. Those parameters, the peripheral address and the FIFO size can be configured with the XPS platform **[16]**.

Once the FSL bus is chosen as the bus supported by the peripheral, the next two steps allow to make the pin associations for that bus, if the name tags and connections are correctly set in the peripheral- the connections will be almost made automatically. Nevertheless, for the case of Figure 43 and Figure 44 it was needed to correctly set the FSL_Clock signal, since the clock is not directly accessible in blocks made with System Generator and it could not be set in the IP; it must be associated with the clk signal.



Import Peripheral

MFSL : Port
Define the MFSL bus interface port(s) for this peripheral.

The MFSL bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool has automatically done the selections for you. Otherwise indicate the ports that correspond to the bus connectors.

Bus Interface Port(s): MFSL

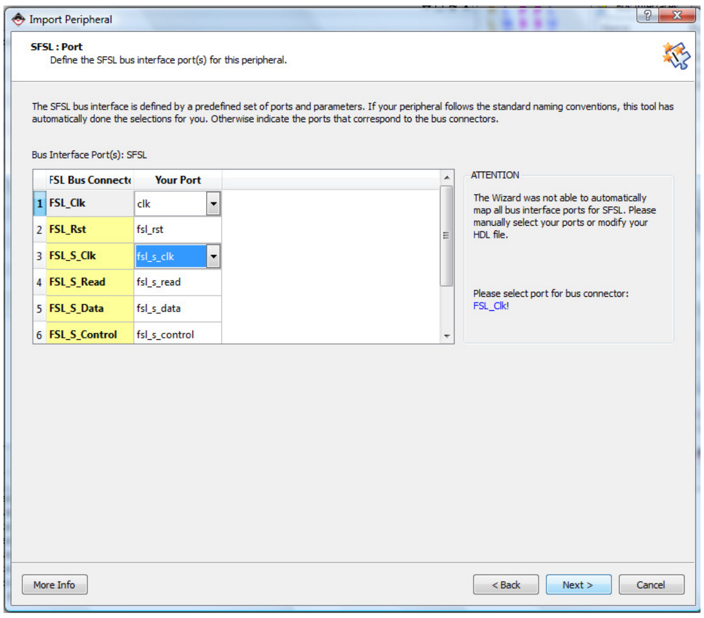
	FSL Bus Connect	Your Port
1	FSL_Clk	clk
2	FSL_Rst	fsl_rst
3	FSL_M_Clk	clk
4	FSL_M_Data	fsl_m_data
5	FSL_M_Control	fsl_m_control
6	FSL_M_Write	fsl_m_write

ATTENTION

The Wizard was not able to automatically map all bus interface ports for MFSL. Please manually select your ports or modify your HDL file.

Please select port for bus connector:
FSL_Clk!

Figure 43. XPS peripheral import- Master FSL pin configuration step.



Import Peripheral

SFSL : Port
Define the SFSL bus interface port(s) for this peripheral.

The SFSL bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool has automatically done the selections for you. Otherwise indicate the ports that correspond to the bus connectors.

Bus Interface Port(s): SFSL

	FSL Bus Connect	Your Port
1	FSL_Clk	clk
2	FSL_Rst	fsl_rst
3	FSL_S_Clk	fsl_s_clk
4	FSL_S_Read	fsl_s_read
5	FSL_S_Data	fsl_s_data
6	FSL_S_Control	fsl_s_control

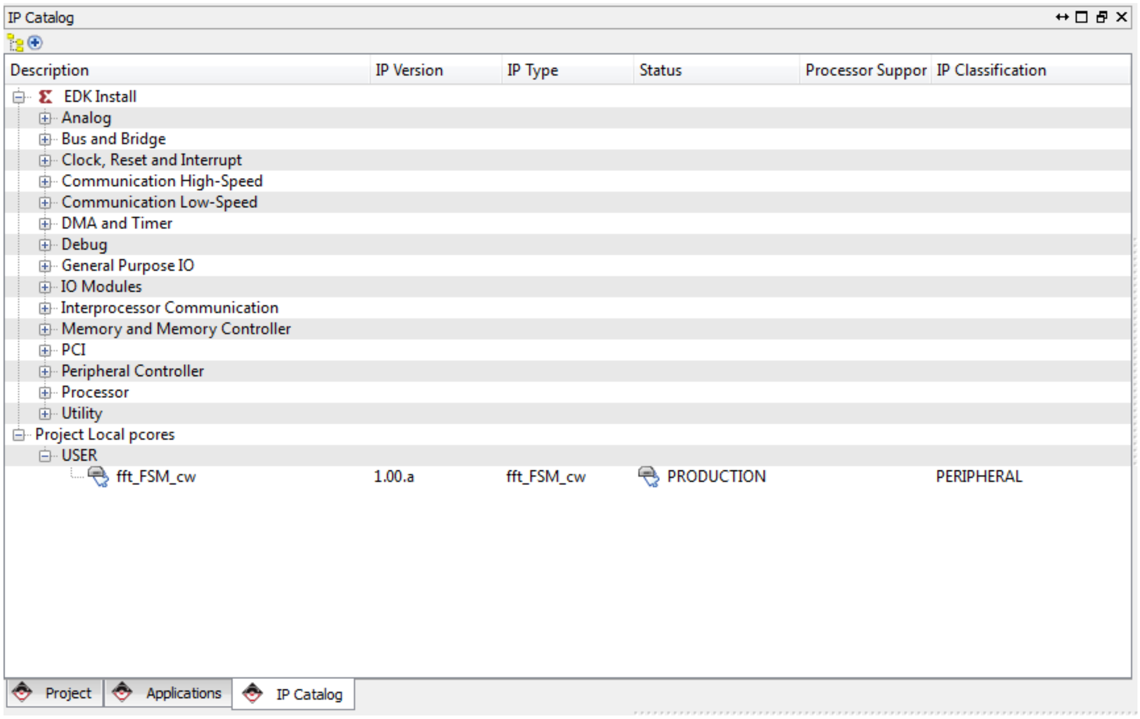
ATTENTION

The Wizard was not able to automatically map all bus interface ports for SFSL. Please manually select your ports or modify your HDL file.

Please select port for bus connector:
FSL_Clk!

Figure 44. XPS peripheral import- Slave FSL pin configuration step.

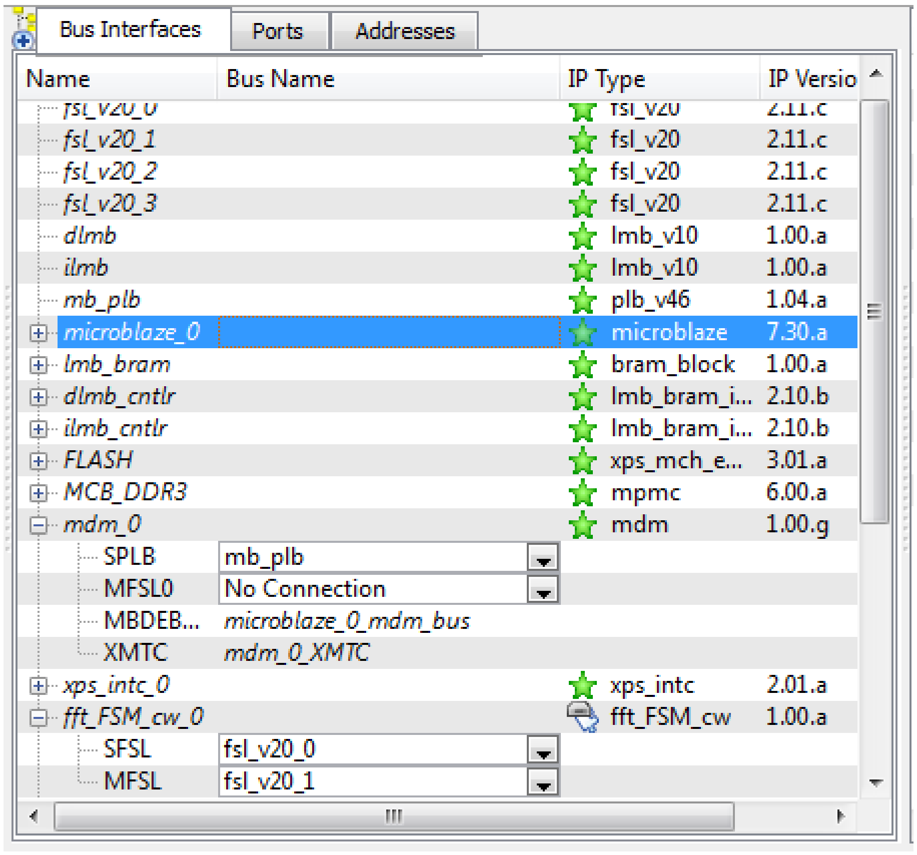
The remaining steps can be let with the default values proposed by the dialog until the they are finished, remaining to include all the required files.



IP Catalog

Description	IP Version	IP Type	Status	Processor Suppor	IP Classification
EDK Install					
Analog					
Bus and Bridge					
Clock, Reset and Interrupt					
Communication High-Speed					
Communication Low-Speed					
DMA and Timer					
Debug					
General Purpose IO					
IO Modules					
Interprocessor Communication					
Memory and Memory Controller					
PCI					
Peripheral Controller					
Processor					
Utility					
Project Local pcores					
USER					
fft_FSM_cw	1.00.a	fft_FSM_cw	PRODUCTION		PERIPHERAL

Figure 45. User defined IP available from the IP Catalog.



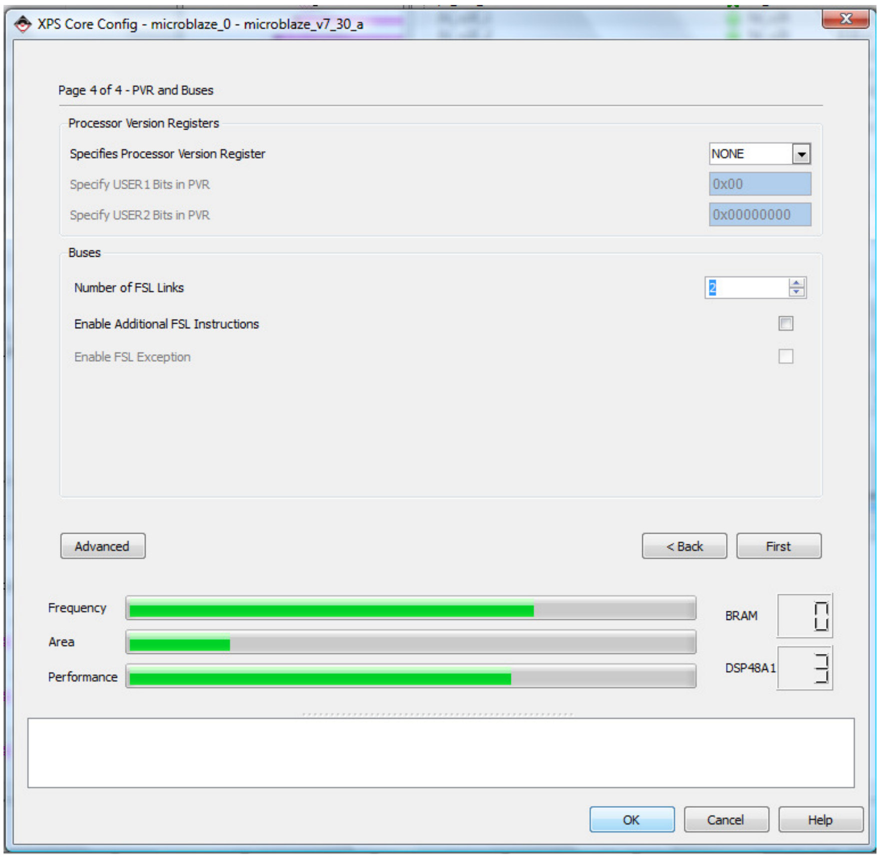
Name	Bus Name	IP Type	IP Versio
fsl_v20_0		fsl_v20	2.11.c
fsl_v20_1		fsl_v20	2.11.c
fsl_v20_2		fsl_v20	2.11.c
fsl_v20_3		fsl_v20	2.11.c
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.04.a
microblaze_0		microblaze	7.30.a
lmb_bram		bram_block	1.00.a
dlmb_cntlr		lmb_bram_i...	2.10.b
ilmb_cntlr		lmb_bram_i...	2.10.b
FLASH		xps_mch_e...	3.01.a
MCB_DDR3		mpmc	6.00.a
mdm_0		mdm	1.00.g
SPLB	mb_plb		
MFSL0	No Connection		
MBDEB...	microblaze_0_mdm_bus		
XMTC	mdm_0_XMTC		
xps_intc_0		xps_intc	2.01.a
fft_FSM_cw_0		fft_FSM_cw	1.00.a
SFSL	fsl_v20_0		
MFSL	fsl_v20_1		

Figure 46. Open Microblaze configuration Wizard.

Back to XPS main window, at the IP Catalog tab, it is possible to see the newly created IP core from the ISE project imported from the Simulink design done with System Generator, as it can be seen in Figure 45. To add the newly created IP to the design as a new peripheral it is just necessary to double-click over it.

The remaining step is to connect this peripheral to the Microblaze microprocessor trough FSL, in order to achieve the FSL interconnection two FSL busses have to be added trough the Micrblaze Configuration Wizard, right clocking over the Microblaze instance in the System Assembly view.

In the Microblaze Configuration Wizards, it is necessary to advance clicking next until finding the PVR and Buses dialog and adding the two required FSL buses to connect the peripheral (Figure 47).


XPS Core Config - microblaze_0 - microblaze_v7_30_a

Page 4 of 4 - PVR and Buses

Processor Version Registers

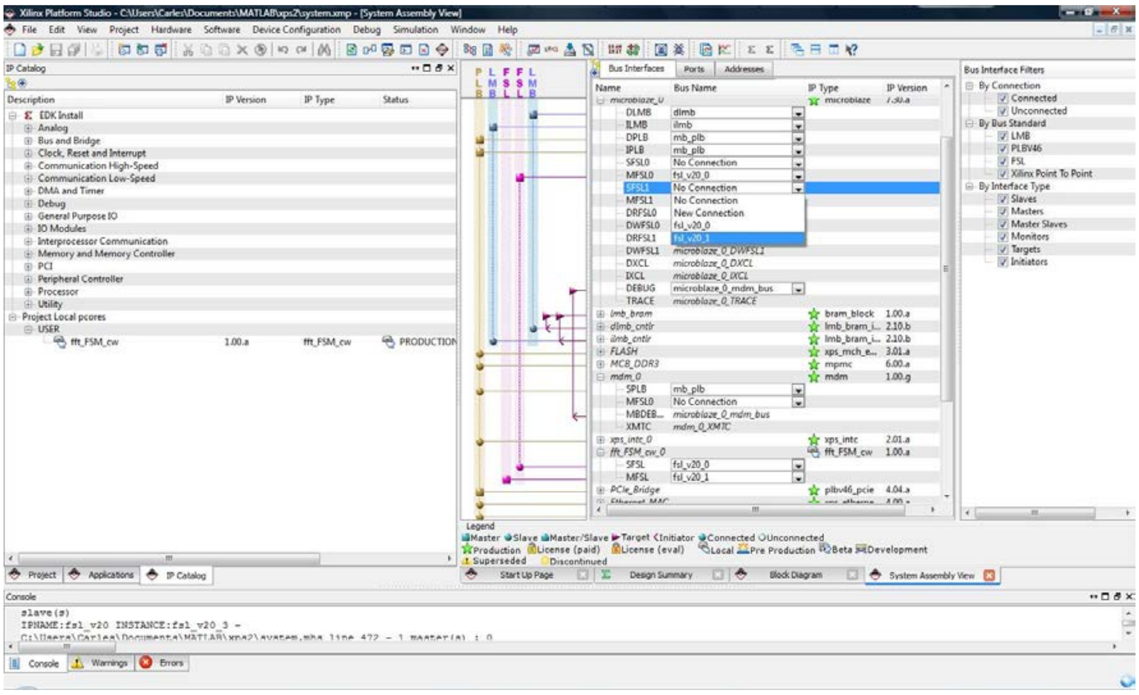
Specifies Processor Version Register	NONE
Specify USER1 Bits in PVR	0x00
Specify USER2 Bits in PVR	0x00000000

Buses

Number of FSL Links	2
Enable Additional FSL Instructions	☐
Enable FSL Exception	☐

Figure 47. Microblaze Configuration Wizard buses dialog.

The remaining step to connect the new peripheral is to physically connect in the System Assembly View a slave FSL connection from the Microblaze to a Master FSL connection of the peripheral and establish another connection with the remaining bus in the opposite communication direction, as it can be seen in Figure 48.



Name	Bus Name	IP Type	IP Version
microblaze_0		microblaze	7.30.a
DLMB	dlmb		
ILMB	ilmb		
DPLB	mb_plb		
IPLB	mb_plb		
SFSL0	No Connection		
MFSL0	fsl_v20_0		
SFSL1	No Connection		
MFSL1	No Connection		
DRFSL0	New Connection		
DWFSL0	fsl_v20_0		
DRFSL1	fsl_v20_1		
DWFSL1	microblaze_0_DWFSL1		
DXCL	microblaze_0_DXCL		
IXCL	microblaze_0_IXCL		
DEBUG	microblaze_0_mdm_bus		
TRACE	microblaze_0_TRACE		
lmb_bram		bram_block	1.00.a
dlmb_cntlr		lmb_bram_i...	2.10.b
ilmb_cntlr		lmb_bram_i...	2.10.b
FLASH		xps_mch_e...	3.01.a
MCB_DDR3		mpmc	6.00.a
mdm_0		mdm	1.00.g
SPLB	mb_plb		
MFSL0	No Connection		
MBDEB...	microblaze_0_mdm_bus		
XMTC	mdm_0_XMTC		
xps_intc_0		xps_intc	2.01.a
fft_FSM_cw_0		fft_FSM_cw	1.00.a
SFSL	fsl_v20_0		
MFSL	fsl_v20_1		
PCIe_Bridge		plbv46_pcie	4.04.a

```
slave(s)
IPNAME:fsl_v20 INSTANCE:fsl_v20_3 -
C:\Users\Carles\Documents\MATLAB\xps2\system.mhs line 472 - 1 master(s) : 0
```

Figure 48. XPS FSL Master/Slave port configuration between the Microblaze and peripheral.

Now to be able to use the newly created peripheral transparently with the test program of the SDK a final Export Step is required: Project -> Export Hardware Design to SDK. When the export process finishes the design can be open with the SDK and it can be programmed as an embedded system.

4 - Tests and Results

4.1 - Simulations and test benches

For each significant component used in this project, simulations and testbenches have been performed to assure its correct operation. In fact, some of them allowed the recycling of almost all the code since some of the signals where shared.

The simulation of the components separately allows to perform a much more precise design and avoiding interaction errors between the different modules while making them easier to find and fix. Nevertheless, tests over the complete peripheral where also made, since assuring the correct operation of all the blocks together with simulations is a crucial part of design process and some of those simulations can be seen in the following section, Figure 49 and Figure 50.

Given the fact that System Generator has a good graphic environment, specially designed to simulate math algorithms it plays also a role in the simulation process. Nevertheless, Matlab simulation environment is not well suited for working with the time scale of the FPGA signals. To help with that- Xilinx provides the WaveScope block, which allows visualizing the signals in a FPGA time scale. Nevertheless, the WaveScope block does not achieve the level of functionality available with Xilinx ISE simulation tools.

4.2 – FSM Test

To test the FSM, a special version of the peripheral has been implemented which allows the visualization of all the signals inside the peripheral; this facilitates an easier verification of the values of all signals during each step.

The FSM behaved as expected and there has not been any unexpected issue due to a bad synthesization of the m-code file, in fact the design of a m-code FSM in System Generator is very similar to designing it in VHDL, with the difference that the signal propagation is taken care of by System Generator. Nevertheless, transition glitches may appear in the same manner as they would in VHDL and in most cases a cast block is needed to adapt the output signal of the FSM to the incoming format of each destination block.

The tested FSM has 20 states and two execution branches, as described in section 3.2, to perform the test, a custom test bench has been done and a set of test data has been written to it form files; since the size of the FFT- 512 elements, makes it difficult to use written constants. In fact the test data has been exported from a Matlab signal to a file, then read by the testbench, processed by the peripheral and the result is written back to another file in a Matlab friendly format. This has allowed to verify the behaviour of the device with a mathematical tool such as Matlab to see if the results are as expected.

The testbench has the necessary procedures to synchronize the required input signals to manage the behaviour of the module and it mainly serves data to the peripheral in various cycle modes to test the different execution branches. As stated before, the FPGA behaves as expected.

To avoid issues if the Microblaze is not enough fast to load a new value of data each cycle, a protection mechanism has been implemented, if the fsl_s_exists changes to '0' during the loading process, the enable signal of the FFT is set to '0' halting its operation until the a new value arrives.

In Figure 49 a capture of the overall simulation process is shown, and as it can be seen, there are a total of 6 cycles in the simulation, 5 of simultaneous load and unloading of data and 1 of separate loading and unloading. The FFT loading data cycles are marked with a black arrow, the FFT unloading cycles are marked with a blue arrow, additionally to test the peripheral good behaviour even if not all the data is loaded continuously a starvation period has been introduced marked with the orange arrow, where it can be seen that the FFT resumes is operation after the data input resumes.

To get a detailed view of the steps taken to start the calculation of a new FFT, a plot of the simulation of the first 500 ns of operation of the peripheral can be seen at Figure

50. First of all the peripheral is reset by the fsl_rst signal marked with a black arrow. The FSM resets the other blocks of the peripheral setting the internal reset signals to ‘1’, marked with the dark blue arrow. After the FSL bus stops resetting the peripheral, the FSM initializes the FFT IP storing the forward/inverse and the scale parameters (light blue arrows) and waits in the idle state. Once data is received trough the bus, the fsl_s_exists signal changes its value to ‘1’; after one cycle, the start signal is set to one to enable the loading process of the FFT block (green arrow). Finally, in the next cycle the FFT starts accepting data and signals while setting the rfd signal to ‘1’, the FSM starts in this same cycle the unload process of the FSL FIFO setting the fsl_s_read signal to 1, which will pop a value from the FSL bus while the FFT stores it in the corresponding position.

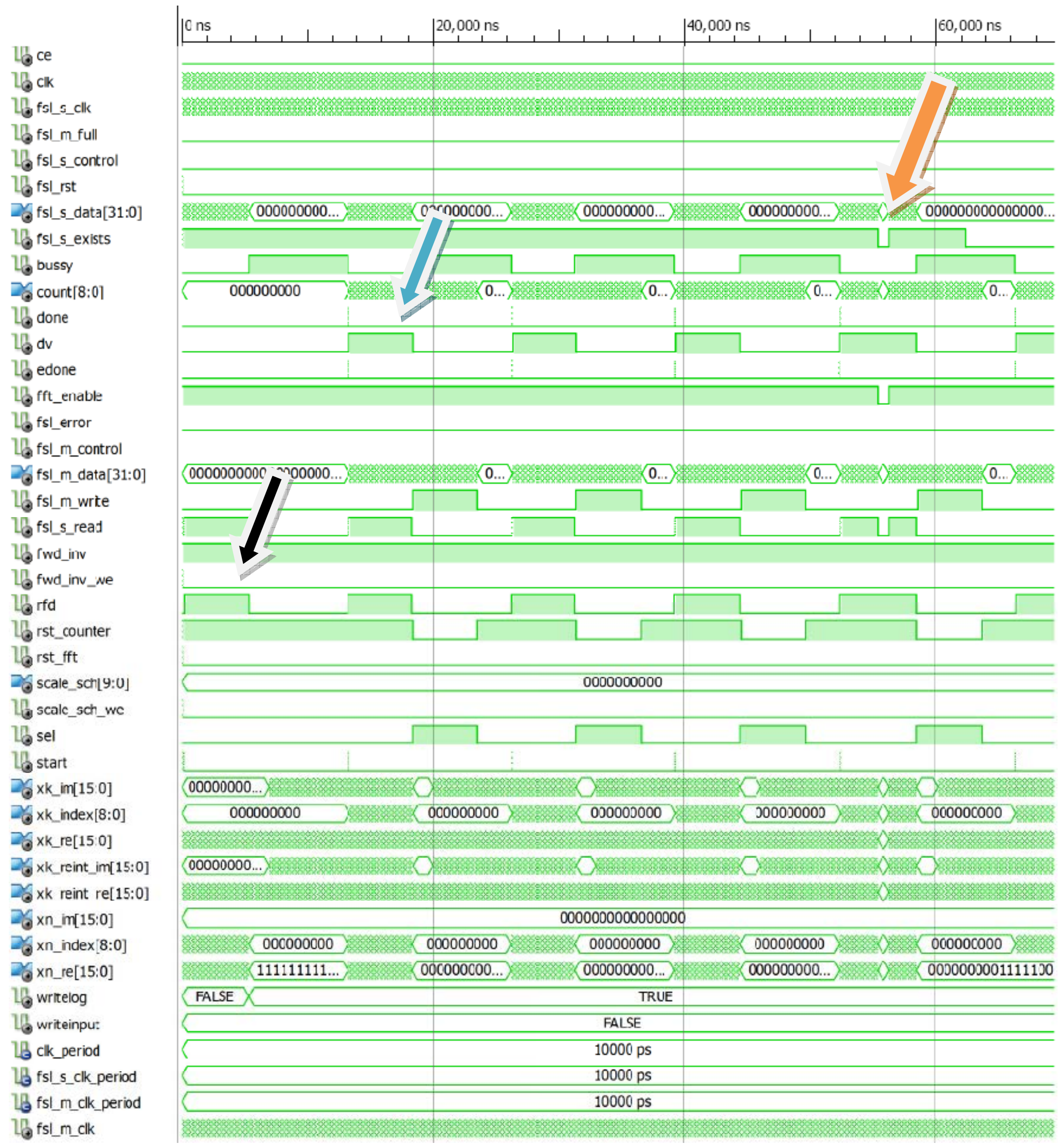


Figure 49. General simulation capture with all the cycles.

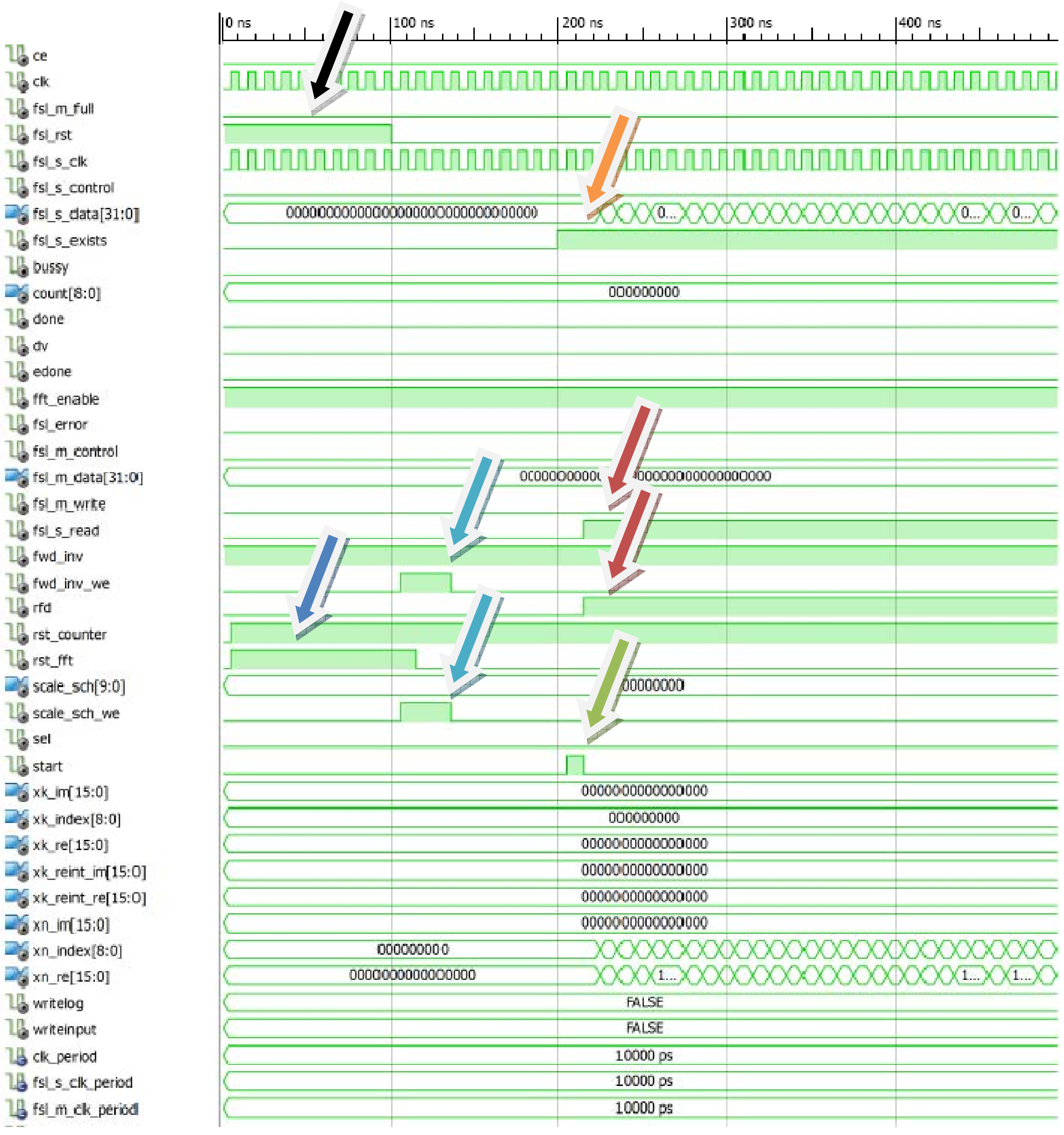


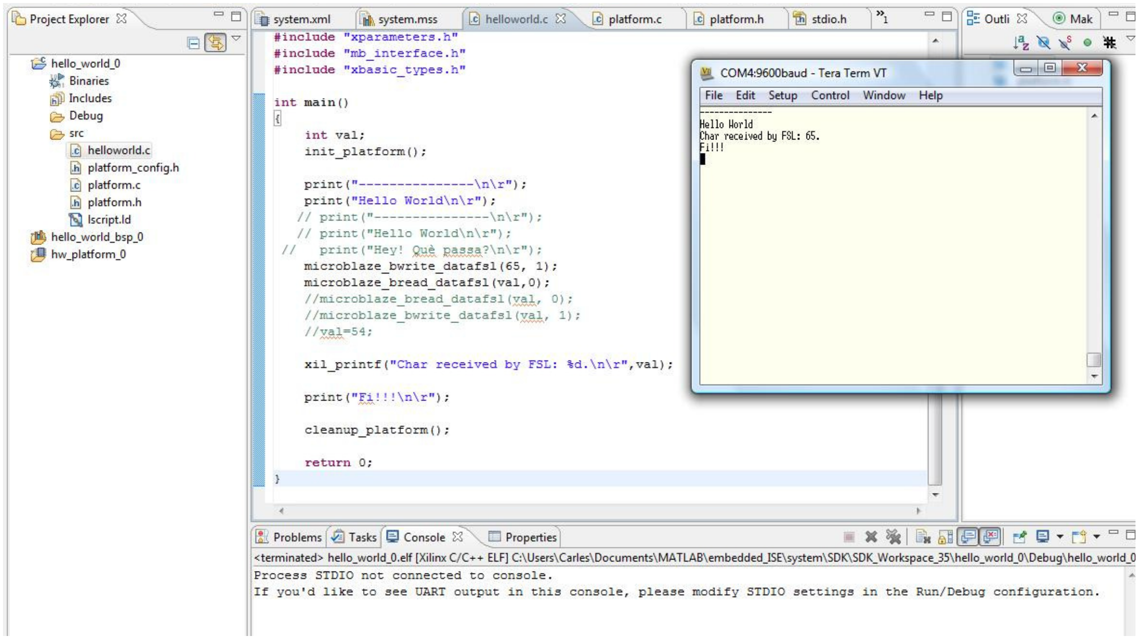
Figure 50. FFT peripheral FSM initial cycles simulation.

4.3 - Field experiments and tests

The SP605 board available at La Salle has been used to test the final designs and the findings done in the field of firmware development. Furthermore, basic tests and training for this specific board was done, since its capabilities where yet to be explored.

Regarding to the tests of the embedded system- a serial communications layer was established taking advantage of an UART instantiated with Xilinx Platform Studio. First the FPGA had been programmed with the system and after this step is performed the test programs could be loaded.

In Figure 51 a screenshot of the tests made in the system loaded in the SP605 board and the test program run with the resulting output read from the serial connection at its side is shown. This simple test which is a proof of concept of the design methodology developed during this project uses the FSL peripheral, developed with System Generator, which works as a loopback tester between a master FSL bus and a slave FSL bus, after being instantiated as a peripheral for the Microblaze it is used for the C program done by the Microblaze to test the loopback connection and the result is printed with the UART. In this example the value 65 is first written to the master FSL bus and then read with the slave FSL but proving the correctness of the setup.



```c
#include "xparameters.h"
#include "mb_interface.h"
#include "xbasic_types.h"

int main()
{
    int val;
    init_platform();

    print("---------------\n\r");
    print("Hello World\n\r");
   // print("---------------\n\r");
   // print("Hello World\n\r");
 //   print("Hey! Què passa?\n\r");
    microblaze_bwrite_datafsl(65, 1);
    microblaze_bread_datafsl(val,0);
    //microblaze_bread_datafsl(val, 0);
    //microblaze_bwrite_datafsl(val, 1);
    //val=54;

    xil_printf("Char received by FSL: %d.\n\r",val);

    print("Fi!!!\n\r");

    cleanup_platform();

    return 0;
}
```

```
---------------
Hello World
Char received by FSL: 65.
Fi!!!
```

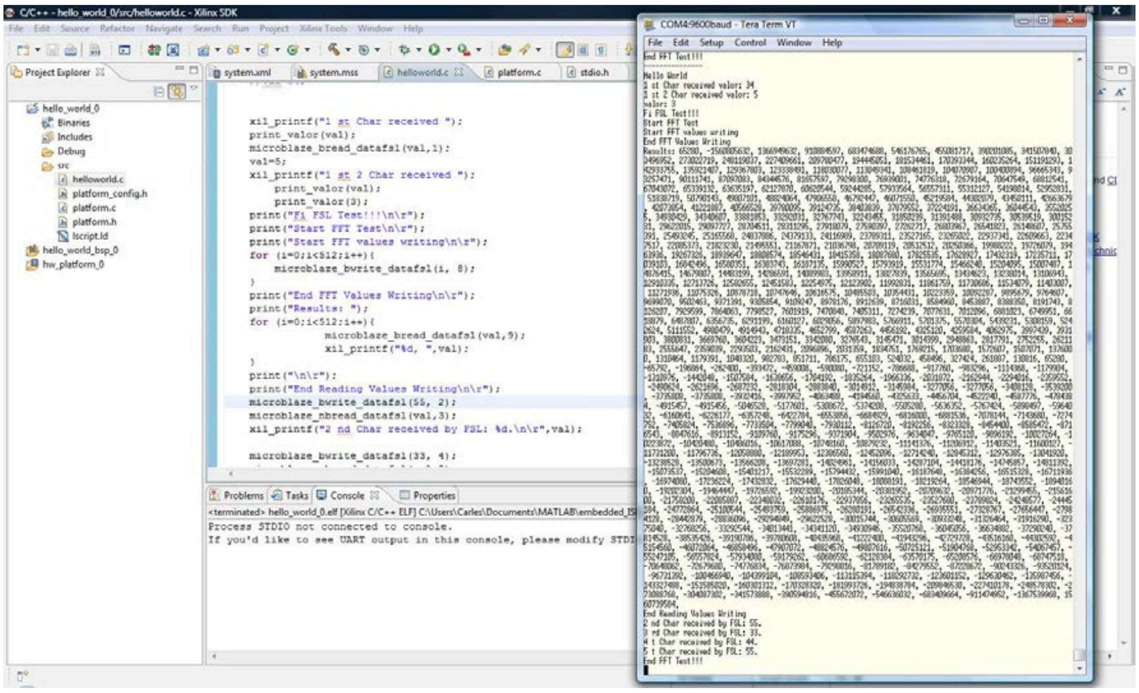
```
<terminated> hello_world_0.elf [Xilinx C/C++ ELF] C:\Users\Carles\Documents\MATLAB\embedded_ISE\system\SDK\SDK_Workspace_35\hello_world_0\Debug\hello_world_0
Process STDIO not connected to console.
If you'd like to see UART output in this console, please modify STDIO settings in the Run/Debug configuration.
```

Figure 51. Xilinx ISE screenshot of the FSL test program with the result read from the UART.

Despite the fact that testing the FSL peripheral yet validates the design methodology developed, it was intended to test a more useful block that may be used in communications systems to prove System Generator's reliability and convenience for creating those types of designs. Therefore, the FFT peripheral has been also tested having successfully performed transformations which is was the final objective of this thesis.

Despite the fact that the tests have been successfully accomplished, problems were encountered with the first designs due to the instantiation of the clocks of the peripherals, which made them unresponsive trough, the FSL bus. This was solved assuring that the clk signal of the peripherals was connected to one of the clocks of the FPGA.

To prove the stated before Figure 52 is provided where the output of the peripheral read by the Microblaze and printed through the UART.



```
xil_printf("1 st Char received ");
print_valor(val);
microblaze_bread_datafsl(val,1);
val=5;
xil_printf("1 st 2 Char received ");
    print_valor(val);
    print_valor(3);
print("Fi FSL Test!!!\n\r");
print("Start FFT Test\n\r");
print("Start FFT values writing\n\r");
for (i=0;i<512;i++){
    microblaze_bwrite_datafsl(i, 8);
}
print("End FFT Values Writing\n\r");
print("Results: ");
for (i=0;i<512;i++){
            microblaze_bread_datafsl(val,9);
            xil_printf("%d, ",val);
}
print("\n\r");
print("End Reading Values Writing\n\r");
microblaze_bwrite_datafsl(55, 2);
microblaze_nbread_datafsl(val,3);
xil_printf("2 nd Char received by FSL: %d.\n\r",val);

microblaze_bwrite_datafsl(33, 4);
```

```
<terminated> hello_world_0.elf [Xilinx C/C++ ELF] C:\Users\Carles\Documents\MATLAB\embedded_IS
Process STDIO not connected to console.
If you'd like to see UART output in this console, please modify STDI
```

```
End FFT Test!!!
---------------
Hello World
1 st Char received valor: 34
1 st 2 Char received valor: 5
valor: 3
Fi FSL Test!!!
Start FFT Test
Start FFT values writing
End FFT Values Writing
Results: 65280, -1560805632, 1366949632, 910884597, 683474688, 546176765, 455081717, 390201085, 341507840, [illegible]
60739584,
End Reading Values Writing
2 nd Char received by FSL: 55.
3 rd Char received by FSL: 33.
4 t Char received by FSL: 44.
5 t Char received by FSL: 55.
End FFT Test!!!
```

Figure 52. Tests with a SP605 of the FFT and FSL peripherals run by a C program from the Microblaze soft microprocessor.

In order to use the data output from the FSL bus it is necessary to split the 32 bits signal into two 16 bits integers and extend the signal bit, due to limitations of Matlab's bit treatment functions, this issue has to be solved in C on the Microblaze with the following code to extract each value, which is :

```
temp=val&0x0000FFFF;
if((temp&0x00008000)==0x00008000) temp|=0xFFFF0000;
printf("%d, ",temp);
temp=(val&0xFFFF0000)>>16;
if((temp&0x00008000)==0x00008000) temp|=0xFFFF0000;
printf("%d, ",temp);
```

In Figure 53, represented in Matlab the resulting continuous signal of performing a FFT to a delta with amplitude 10 at t=0 is represented.

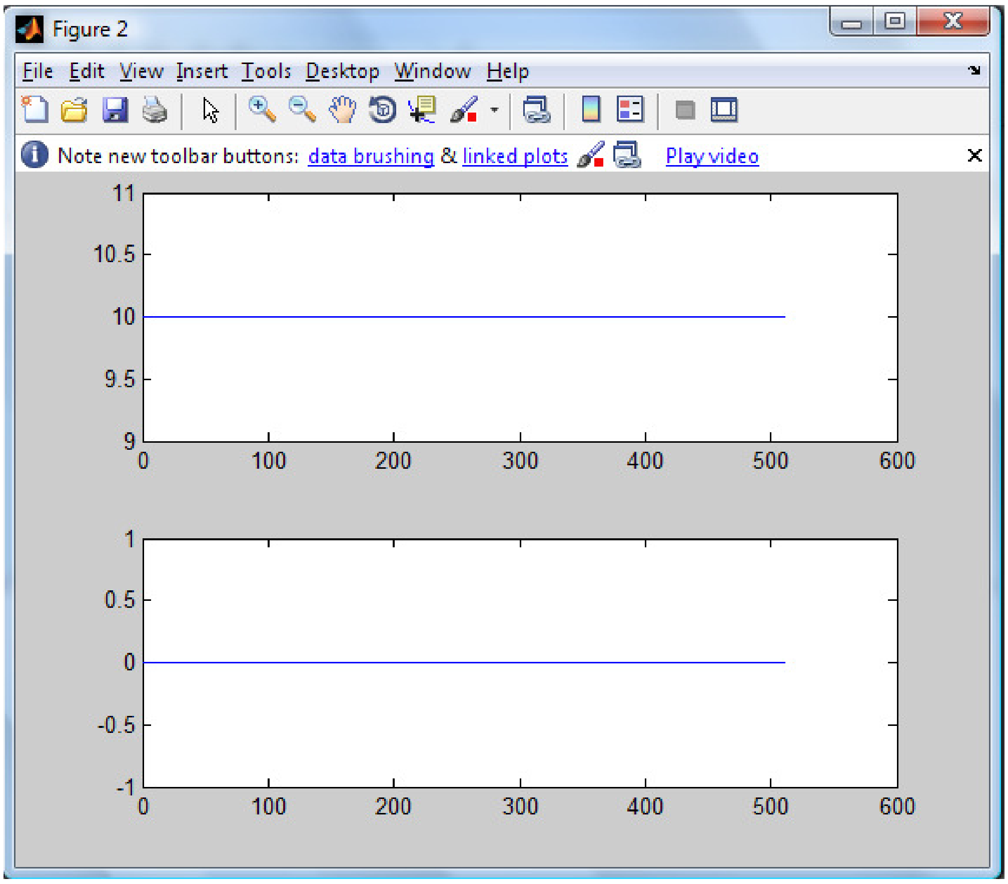


Figure 53. 10·δ(t) FFT calculated with the Microblaze peripheral.

The following figures represent a δ at n-1 input signal of two different amplitudes transformed, which allow to observe the behavior of the real instantiated device compared to the expected ideal result. In Figure 54 and Figure 55 there can be seen the result of the δ of amplitude 10. Since this is a low value, the limited precision of the FFT causes low power high frequency noise, which is traduced in small random oscillations around the expected values. On the contrary, with a high value as an input such as 1000- this small noise caused to the fixed coma arithmetic of the FFT becomes unnoticeable and as it can be seen in Figure 56 and Figure 57; the signals resulting from Matlab and the Microblaze look alike.

During the tests, no problems related to bad routing and delays have been observed, which implies that the designs made with System Generator are reliable and are not affected during the placement process. In addition, the FSL bus interconnection system has also proven to be reliable and a good option regarding to connecting peripherals of system on chip design.

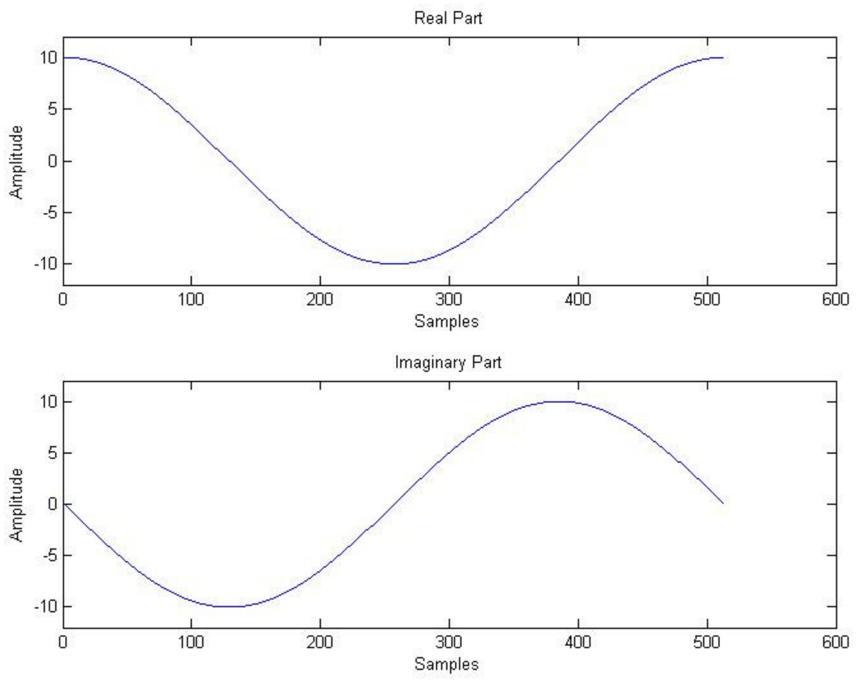


Figure 54. 10·δ(n-1) FFT calculated with the Microblaze peripheral.

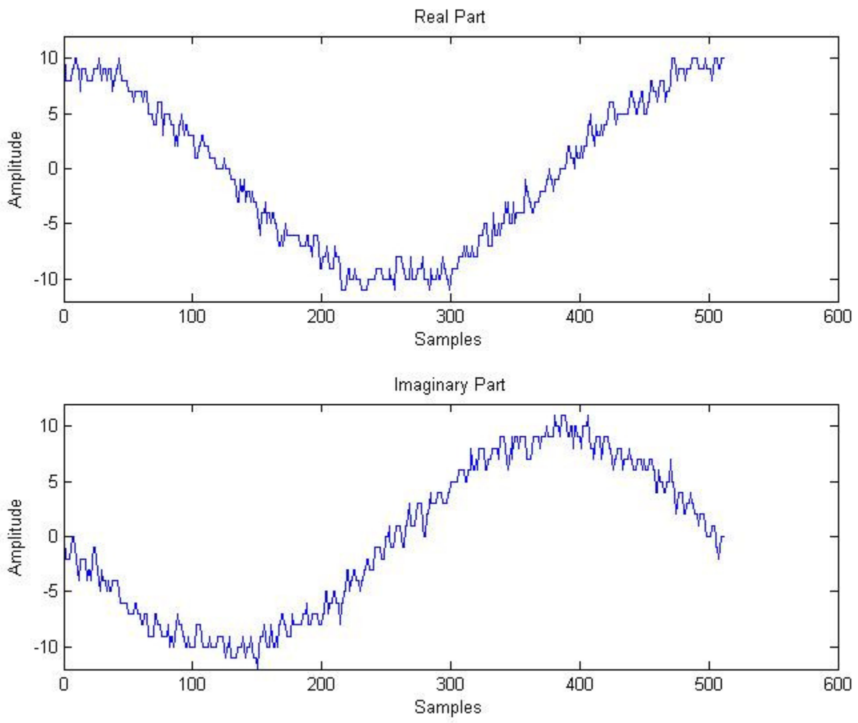
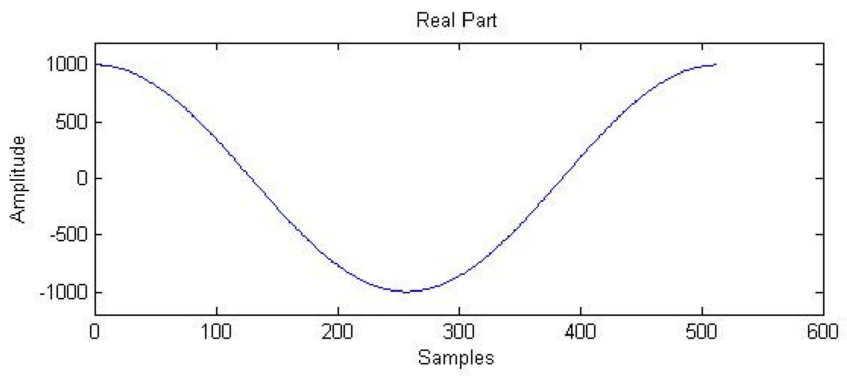


Figure 55. 10·δ(n-1) FFT calculated with the Microblaze peripheral.



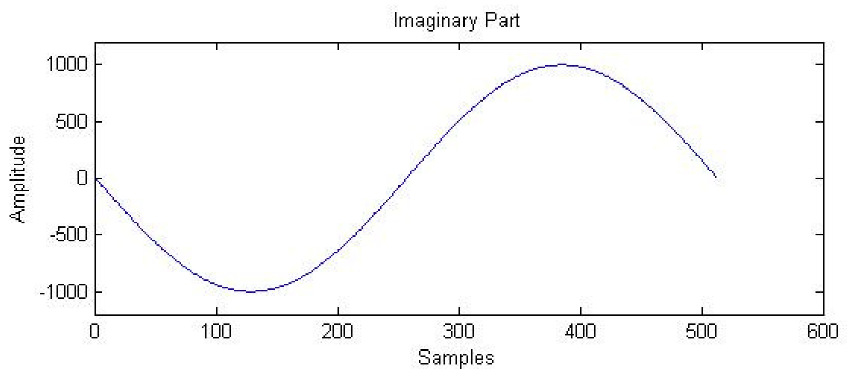
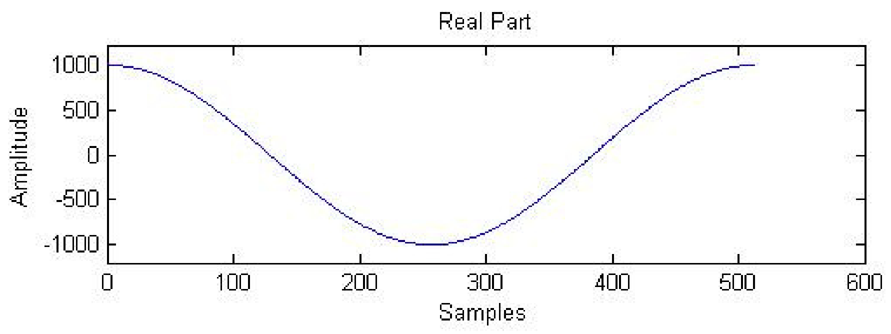


Figure 56. 100·δ(n-1) FFT calculated with Matlab.



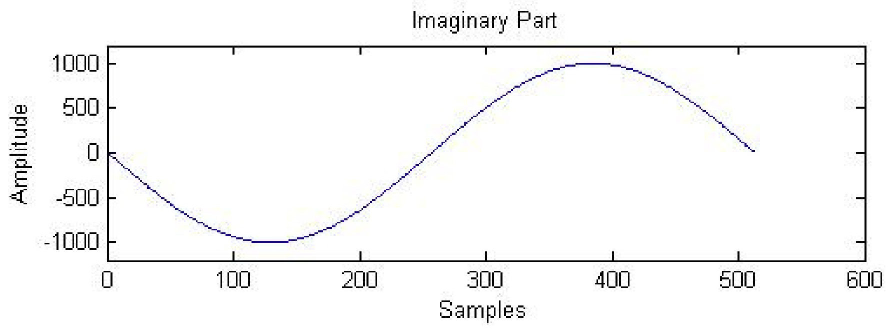


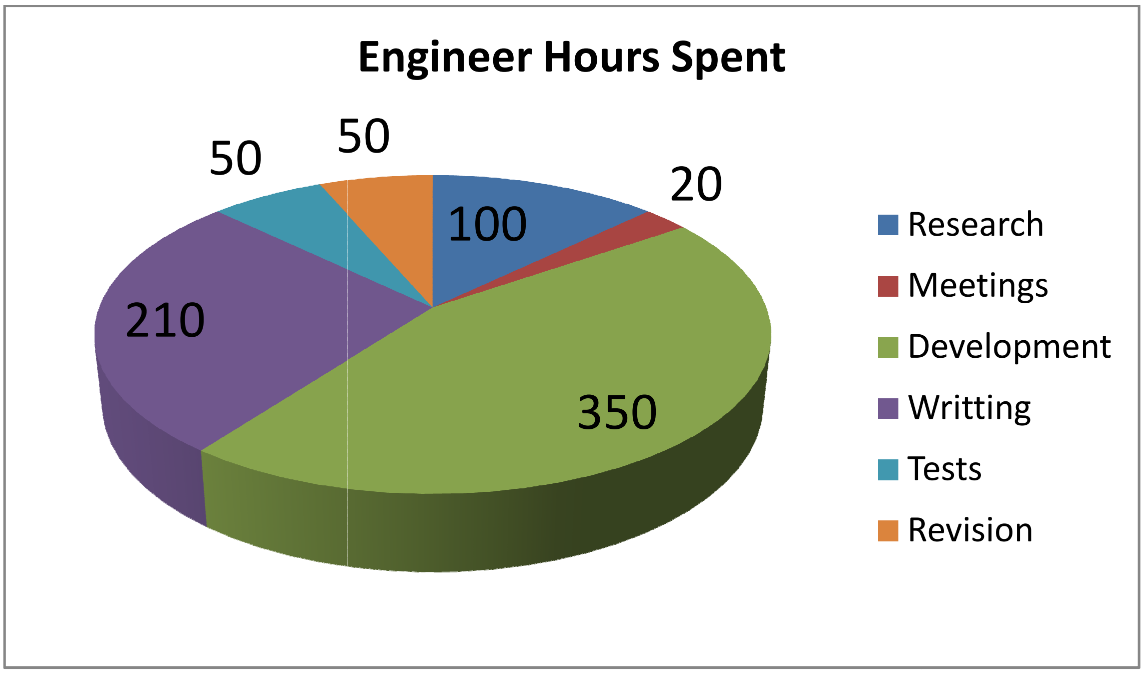
Figure 57. 100·δ(n-1) FFT calculated with Microblaze peripheral.

5 - Resources spent

For this project around 780 engineer hours have been spent among its development, coordination meetings, research, writing of the memory and revision.

Regarding to the physical resources used- 1 personal computer has been used, the Xilinx SP605 board and the necessary connection cables. The most relevant software packages used are: Matlab, Xilinx ISE and Microsoft Office.

No resources have been required to be specially acquired for this project, therefore it has been free of charge, however the number of already available resources and equipment used for its development is considerable. A graphic of the distribution of the engineer hours in tasks can be seen below.



6 - Conclusions

With this project we have reached objectives far beyond the initial ones, which where to explore and document Xilinx System Generator and its integration capabilities with Xilinx ISE. In addition to the initial objectives, we have successfully created instantiable designs with System Generator that can be used with Xilinx ISE to create a firmware alone or as a component of other entities. Furthermore, we have achieved to integrate these designs in the complete Xilinx SoC design flow, in the form of peripherals for a Microblaze microprocessor. This setup enables these peripherals to be directly connected to free pins of the FPGA, other modules with custom interconnection methods and to be accessed transparently as trough the Microblaze supported buses.

Regarding to the lab tests performed, a set of simulations and tests have been run on the various elements that have been created for this project to assure its proper operation; including the FFT alone, a modified version of the FFT which allows to see its internal signals, the FSM alone and the FSL interface. Additionally, tests with the SP605 board on the lab have been performed to verify the assumptions made during the theoric studies and simulations made for this project. During those tests, the full SoC solution was verified loading programs to the Microblaze microprocessor to perform actions to the System Generator generated peripherals. The correct operation of the C compiled program run by the Microblaze implies the correct interaction of all the design tools used to implement this solution, a part of the tests made to the different parts by separate.

The creation of an embedded system into a FPGA implemented with a soft processor including various design sources working altogether to create a mixed platform capable to execute compiled programs on a higher level of abstraction is a very complex duty not in reach for most engineers. In fact, this was not yet achieved at La Salle. Therefore, considering that the work done for this project is of a very high complexity- this project could be qualified as a great success.

Figures Index

Glossary

BSB: Base System Builder
CMOS: Complementary Metal Oxide Semiconductor
CPU: Central Processing Unit
DSP: Digital Signal Processing
EDK: Embedded Development Kit
EDAC: Error Detection And Correction
ELF: Executable and Linkable Format
FIFO: First Input First Output
FPGA: Field Programmable Gate Array
FSM: Finite State machine
GUI: Graphical User Interface
GPIO: General Purpose Input/Output
HDL: Hardware Description Language
I2c: Inter-Integrated Circuit protocol
IC: Integrated Circuit
IP: Intellectual Propriety
ISE: Integrated Software Environment
LED: Light Emitting Diode
PC: Personal Computer
PMT: PhotoMultiplier Tube
RAM: Random Access Memory
ROM: Read Only Memory
RS: Red Solomon
SDK: Software Development Kit
SOC: System On Chip
SRAM: Static Random Access Memory
VHDL: VHSIC (Very High Speed Integrated Circuits) Hardware Description Language
XPS: Xilinx Platform Studio

Bibliography

[1] Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel; Hans-Peter Rosinger; May 12, 2004; Xilinx

[2] Sklar, Bernard; Digital Communications: Fundamentals and Applications; 2nd Edition; Prentice Hall; 2001

[3] ISE Simulator (ISim) In-Depth Tutorial; April 27, 2009; Xilinx

[4] Hamming code definition; Wikipedia; http://en.wikipedia.org/wiki/Hamming_code

[5] http://en.wikipedia.org/wiki/Xilinx

[6] http://www.xilinx.com/company/about.htm

[7] http://www.ehow.com/about_5390865_introduction-xilinx.html

[8] http://en.wikipedia.org/wiki/Microblaze

[9] PicoBlaze 8-bit Embedded Microcontroller User Guide

[10] ISE Design Suite: Intellectual Property

[11] Part Family Details for: "Spartan 6 FPGAs • Xilinx"; http://www.supplyframe.com/content/part-family/xilinx/XC6SLX4|XC6SLX9|XC6SLX16|XC6SLX25|XC6SLX25T|XC6SLX45|XC6SLX45T|XC6SLX75|XC6SLX75T|XC6SLX100|XC6SLX100T|XC6SLX150|XC6SLX150T?id=1932196

[12] Spartan-6 FPGA Configurable Logic Block User Guide; UG384 (v1.1) February 23, 2010; Xilinx

[13] Xilinx Homepage; Xilinx Products & Services; http://www.xilinx.com/products/index.htm

[14] SP605 Hardware User Guide; UG526 (v1.3) June 16, 2010; Xilinx

[15] Processor Local Bus (PLB) v4.6 Product Specification; December 2, 2009; Xilinx

[16] LogiCORE IP Fast Simplex Link (FSL) V20 Bus Product Specification; April 19, 2010; Xilinx

[17] EDK Concepts, Tools, and Techniques. Guide to Effective Embedded System Design; Xilinx

[18] XPS General Purpose Input/Output (GPIO) (v2.00a); December 2, 2009; Xilinx

[19] LogiCORE IP Fast Fourier Transform v7.1; April 19, 2010; Xilinx