III.1.Introduction

In recent years, Field Programmable Gate Arrays (FPGAs) have become key components in implementing high performance Digital Signal Processing (DSP) systems, especially in the areas of digital communications, networking, video, and imaging. The logic fabric of today's FPGAs consists not only of look-up tables, registers, multiplexers, distributed and block memory, but also dedicated circuitry for fast adders, multipliers, and I/O processing [18]The memory bandwidth of a modern FPGA far exceeds that of a microprocessor or DSP processor running at clock rates two to ten times that of the FPGA. Coupled with a capability for implementing highly parallel arithmetic architectures, this makes the FPGA ideally suited for creating high-performance custom data path processors for tasks such as digital filtering, fast Fourier transforms, and forward error correction.

System Generator is a software tool for modeling and designing FPGA-based DSP systems in Simulink. The tool presents a high level abstract view of a DSP system, yet nevertheless automatically maps the system to a faithful hardware implementation. What is most significant is that System Generator provides these services without substantially compromising either the quality of the abstract view or the performance of the hardware implementation.

In this chapter the details of the controller design and its implementation are presented. In particular, the main design issues are analyzed, and a brief description of the process model identification is provide, as well. Then, each implementation step is reported.

And explain a method for the design and implementation of multiplierless digital PID controller based on (FPGA) devices.

III.2. System Generator

Xilinx System Generator provides a set of Simulink blocks (models) for several hardware operations that could be implemented on various Xilinx FPGAs. These blocks can be used to simulate the functionality of the hardware system using Simulink environment. The nature of most DSP applications requires floating point format for data representation. While this is easy to implement on several computer systems running high level modeling software such as Simulink, it is more challenging in the hardware world due to the complexity of the implementation of floating point arithmetic. These challenges increase with portable DSP systems where more restricting constraints are applied to the system design. For these reasons Xilinx System Generator uses fixed point form at to represent all numerical values in the system. System generator provides some blocks to transform data provided from the software side of the simulation environment (in our case it is Simulink) and the hardware side (System Generator blocks)[19]. This is an important concept to understand during the design process using Xilinx System Generator.

The System Generator is able to generate an FPGA implementation consisting of RTL VHDL and Xilinx Smart-IP Cores from a Simulink subsystem built from the Xilinx Blockset. The overall design, including test environment, may consist of arbitrary Simulink blocks. However, the portion of a Simulink model to be implemented in an FPGA must be built exclusively of Xilinx blocks, with the exception of subsystems denoted as black boxes.

III.3.Objectives

This chapter will demonstrate the PID controller And process of creating a simple DSP system using Xilinx System Generator 12.1 The System Generator runs within the Simulink simulation environment which is part of MATLAB mathematical package.

AsimpleDSP system will be simulated using Simulink and system GeneratorCo-simulation integrates Simulink simulation capabilities with a hardware implementation to verify the functionality of the system.

The following steps are described in this chapter

- Starting System Generator with MATLAB.
- Creating a DSP system using Simulink and System Generator.
- Simulating the DSP system using Simulink.

• Preparing System Generator for (Spartan-6) Board.

• Performing Hardware/Software Simulation for the DSP system.

III.4.Starting System Generator

To Start Xilinx System Generator, select *Start → All Programs → Development → Xilinx → ISE Design Suite 12.1 → System Generator→ Xilinx System Generator 12.1*

The Simulink library browser shows a list of all the different Toolboxes installed within MATLAB. Xilinx System Generator components will appear under three categories

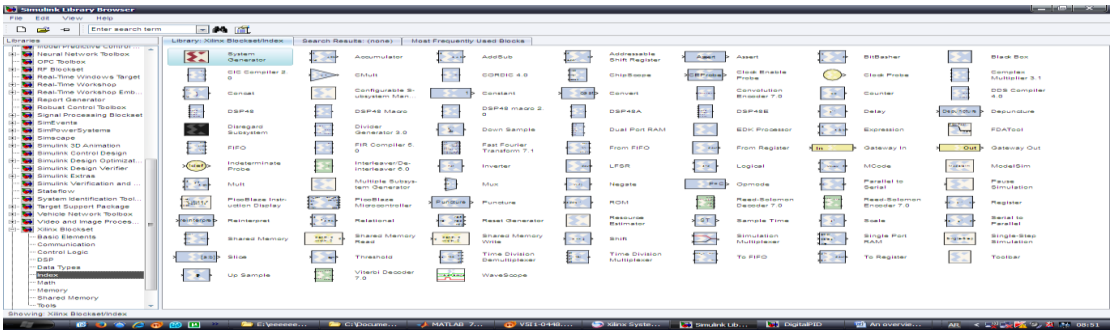
1. Xilinx Blockset.
2. Xilinx Reference Blockset.
3. Xilinx XtremeDSP Kit.

III.5.What is a Xilinx Block?

The Xilinx Blockset is a Simulink library, accessible from the Simulink library browser. It consists of building blocks that can be instantiated within a Simulink model and like other Simulink blocksets, blocks can be combined to form subsystems and arbitrary hierarchies. The Xilinx Gateway blocks (from the Xilinx Blockset's MATLAB I/O library) are used to interface between the Xilinx Blockset fixed point data type and other Simulink blocks[18]**.**

III.6.Some Xilinx DSP Block Set

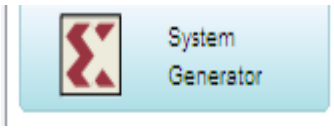
Over 90 DSP building blocks are provided in the Xilinx DSP blockset for Simulink. These blocks include the common DSP building blocks such as adders, multipliers and registers Also included are a set of complex DSP building blocks such as forward error correction blocks, FFTs, filters and memories. These blocks leverage the Xilinx IP core generators to deliver optimized results for the selected device**[**20**].**



FigureIII.1:block set system Generator

III.6.1.System Generator

The System Generator is a special Xilinx block that invokes the tool's code generation software.



FigureIII.2: System Generator

By placing the System Generator token on our Simulink project sheet we can generate HDL and Xilinx LogiCOREs for all the Xilinx blocks on that sheet and on any sheets beneath it in the hierarchy. the system Generator block parameters dialog box allows we to tailor our Simulink simulation and code generation[18].

III.6.2.Addressable Shift Register

The Xilinx Addressable Shift Register block is a variable-length shift register (or delay chain). This block differs from the Xilinx Delay block in that the amount of latency experienced by data from input to block output is variable and depends on the address value.

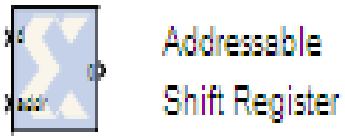


Figure III.3: Addressable Shift Register

Data presented to the block will traverse the entire delay chain. The output of the block is not necessarily the output of the last register in the chain, however. Instead, the output of the block is taken from the register pointed to by the address presented on the addr port.

III.6.3.The Xilinx Black Box

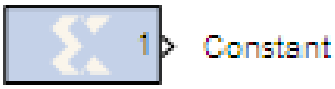
The Xilinx Black Box token enables you to instantiate our own specialized functions in our model, and subsequently into a generated design. Like the System Generator token, the Black Box token can be placed in any Simulink subsystem, identifying the subsystem as a black box. If we choose to include functionality in our Simulink model that does not exist in the current blockset, any Simulink subsystem can be treated as a black box. We may want to build a model out of non-Xilinx blocks for an HDL representation of functionality that you want to turn into a Simulink model.

Figure III.4:black box

To create a black box in the System Generator, we must supply both a Simulink model and a corresponding HDL file.

III.6.4.Constant

The Xilinx Constant block generates a constant. This block is similar to the Simulink constant block, but can be used to drive the inputs on Xilinx blocks.



FigureIII.5: Constant

III.6.5.Register

The Xilinx Register block models a D flip flop-based register, having latency of one sample period.

FigureIII.6: Register

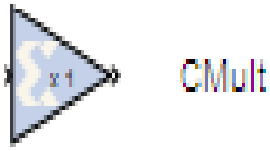
III.6.6.AddSub

The Xilinx AddSub block implements an Adder, Subtractor. The operation can be fixed (Add or Subtract) or changed dynamically under control of the sub mode signal

FigureIII.7: AddSub

III.6.7.CMult block

The Xilinx CMult block implements a gain operator, with output equal to the product of its input by a constant value. This value can be a MATLAB expression that evaluates to a constant.



FigureIII.8:CMult

III.6.8.MATLAB I/O

The MATLAB I/O section includes Xilinx Gateway blocks, the Enabled Subsystem gateway, blocks to report quantization error, and display blocks.

III.6.9.Gateway Blocks

The Xilinx Gateway blocks have several functions:

• Convert data from double precision floating point to the System Generator fixed point type and vice versa during Simulink simulation.

• Define I/O ports for the top level of the HDL design generated by System Generator. A Gateway In block defines a top level input port, and a Gateway Out block defines a top level output port.

• Define testbench stimuli and predicted output files when the System Generator Create Testbench option is selected. In this case, during HDL code generation, Simulink simulation values are logged as logic vectors into a data file for each top level port defined by a Gateway block. An HDL component is inserted in the top level testbench for each top level port which, during HDL simulation, reads the values from the file and compares them to the expected results.

• The name specified for the Gateway In or Gateway Out block is passed on as the port name on the top level VHDL entity.

FigureIII.9: Gateway(In,Out)

III.6.10.Display block

This is the Simulink Display block, linked into the Xilinx Blockset's MATLAB I/O section as a convenience. It is presented as output to the Sample Time display (described next).

Figure III.10: Display

III.6.11.Wave scope

This block in **FigureIII.10**. Is listed in the Xilinx blockset libraries tools And index.

The System Generator have scope block provides a powerful and easy to use wave form viewer for analysis and debugging System Generator design.

The viewer allows to observe the time changing values of any wires in the design after the conclusion of simulation the signals may be formatted in a logic or analog format and may be viewed in binary, hex or decimal radices .

III.7.Designing PID controller in the System Generator

At first the model of PID controller is depicted in Simulink environment for tuning the parameters and evaluation output results of controller. After verifying the performance, the block diagram of controller is drawn again using equivalent Xilinx blocks that are supported in Simulink environment. In these blocks the size of quantization of inputs is determined by Gateway in blocks. Also the output size of each computational block including of AddSub and Multblocks is determined. After evaluating the results, the HDL code can be generated using system generator block. In this block, we determine the type of FPGA device, the type of generated code (Verilog or VHDL), the FPGA clock period and other desired features. After this step, we continue the process using ISE software.

The creation of a DSP design begins with a mathematical description of the operations needed and concludes with a hardware realization of the algorithm. The hardware implementation is rarely faithful to the original functional description instead it is faithful enough. The challenge is to make the hardware area and speed efficient while still producing acceptable results.

In a typical design flow --a flow supported by System Generator-- the following steps occur:

1. Describe the algorithm in mathematical terms.
2. Realize the algorithm in the design environment, initially using double precision.
3. Translate the design into efficient hardware.

III.8.Describe the algorithm in mathematical terms

III.8.1.Dscretization PID Controller

We know The PID algorithm consists of three basic modes, the Proportional mode, the Integral and the Derivative modes.

The general form of PID controller given in most of the text book is the standard form.

$$u(t) = K_p.e(t) + K_i \int e(t).dt + K_d \frac{de(t)}{dt} \text{.....EqIII.1.}$$

The three types of control are combined together to form a PID controller with the transfer function in the S domain:

$$U(s) = K_p + \frac{K_i}{s} + K_d s \text{.....EqIII.2.}$$

The bilinear (Tustin) transform approximation is a simple and easy to use method in order to relate the S- and Z- domains. It is in the form:

$$S = \frac{T}{2} \quad \frac{(Z+1)}{(Z-1)}$$

where T is the sampling period of the discrete system

Applying Bilinear Transform on PID controller by **EqIII.2** To transfer **EqIII.2** in the Z domain

$$U(z) = K_p + \frac{K_i}{\frac{T}{2} \quad \frac{(Z+1)}{(Z-1)}} + K_d \frac{T}{2} \quad \frac{(Z+1)}{(Z-1)} \text{.....EqIII.3.}$$

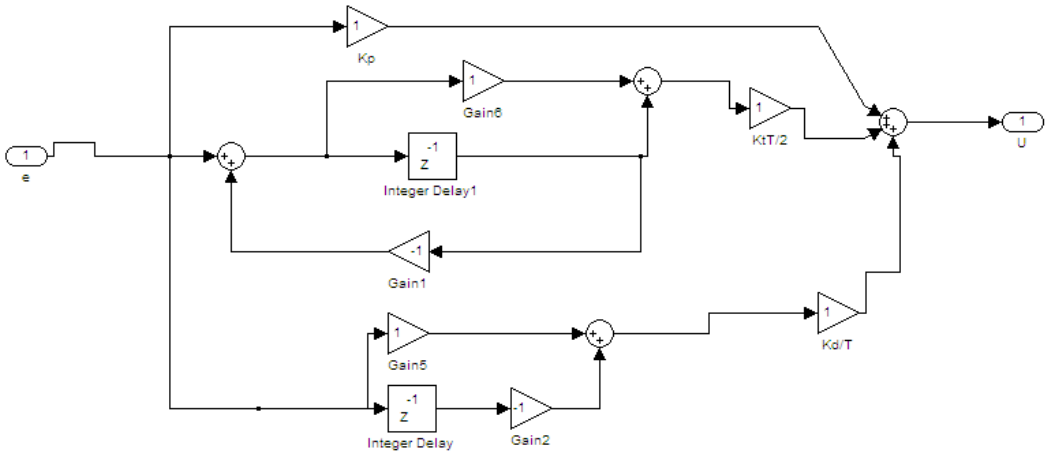
$$U(z) = K_p + \frac{K_i T\,(z+1)}{2(z-1)} + \frac{K_d}{T} \quad \frac{(Z-1)}{Z} \text{.....EqIII.4.}$$

$$U(z) = \frac{K_p(z^2 - z) + \frac{K_i T}{2}(z^2 + z) + \frac{K_d}{T}(z^2 - 2z + 1)}{(z^2 - z)} \text{.....EqIII.5.}$$

After multiplying numerator and denominator of **EqIII.5** by z^{-2}we find

$$U(z) = \frac{(K_p + \frac{K_i T}{2} + \frac{K_d}{T}) + (-K_p + \frac{K_i T}{2} - 2\frac{K_d}{T})z^{-1} + (\frac{K_d}{T})z^{-2}}{(1 - z^{-1})} \quad \text{.....EqIII.6.}$$

We do **EqIII.6** in the Direct formI structure of Digital PID controller **FigureIII.11** is shown this



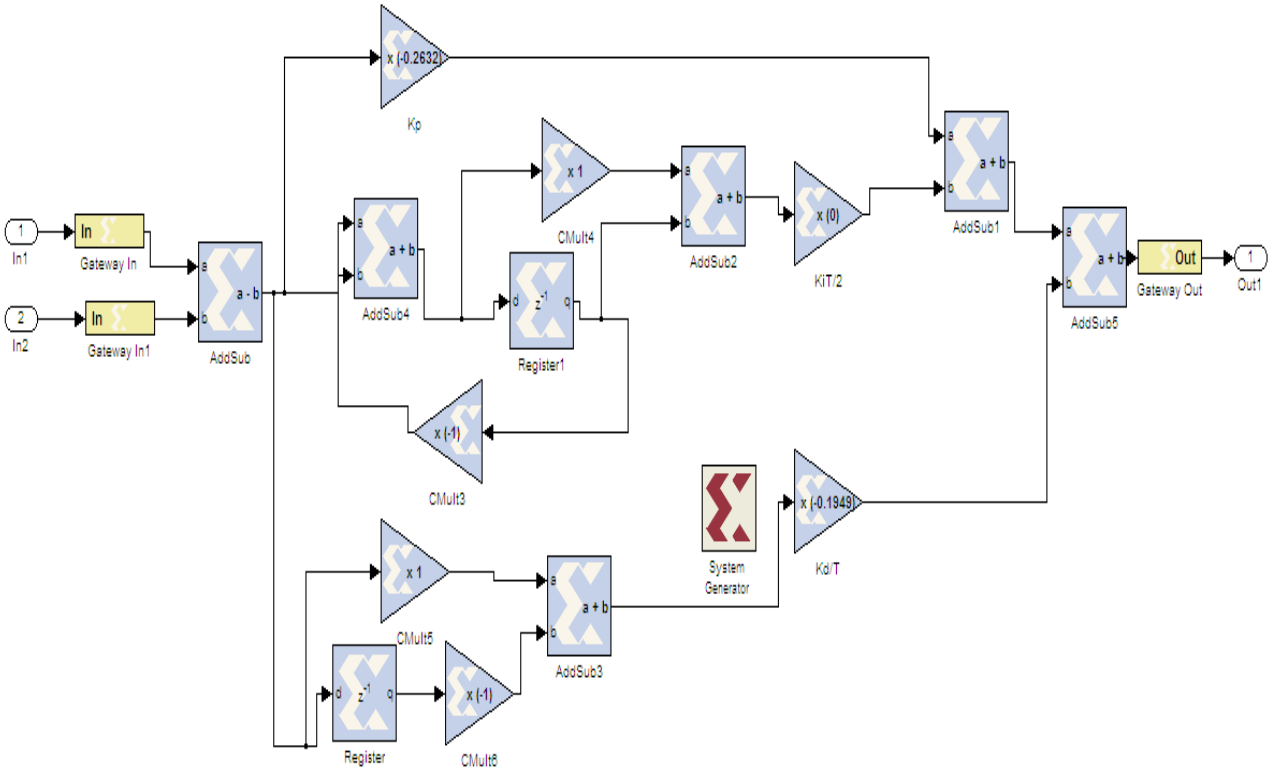
FigureIII.11: The Direct Form I structure

III.9.Designing of The Controllers by using Xilinx blocks

The controllers designed(Direct formI structure) using Simulink basic blocks are again designed using Xilinx blocks in Simulink.

The **FigureIII.12**.shows the Digital PID controller depicted using Xilinx blocks. Insert the System Generator block in the designed model to generate the HDL code.

The steps how to design the Digital PID controller using Xilinx blocks find step by step in AppendixA



FigurIII.12: Digital PID controller using Xilinx system Generator

We do the Digital PID controller in the Subsystem Matlab

III.10.Discret state space representation

We have state space representation of our model in the chapter I in the continuous time we want to transfer this state space in the Discret time just using a Matlabcode or instruction of discretization

So we have

$$\dot{x} = \begin{bmatrix} 0 & -131.33 & 0 & 0 \\ 0 & -0.0081 & 0.00000401 & 0 \\ 0.048 & 0 & -0.231 & 1 \\ 0 & 0 & 0 & -0.5 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.5 \end{bmatrix} u \quad \text{.....EqIII.7}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} x$$

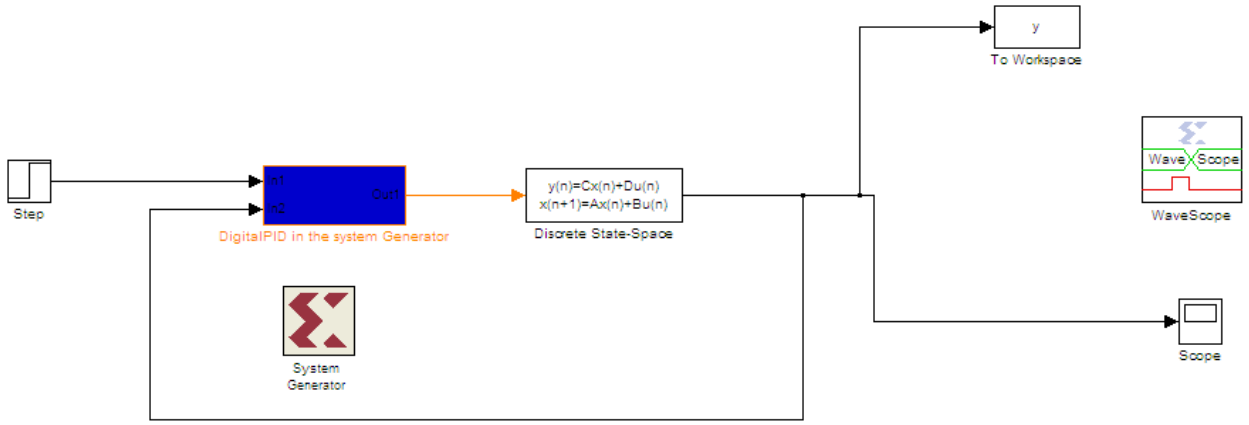
Code Matlab for the discrete the system

```
clear
A=[0 -131.33 0 0;0 -0.0081 0.00000401 0;0.048 0 -0.23 1; 0 0 0 -0.5];
B=[0;0;0;0.5];
C=[1 0 0 0];
D=0;
```

```
Ts=0.001;
sys = ss(A, B, C, D);
sysd = c2d(sys,Ts)
```

$$x(n+1)=\begin{bmatrix}0.999999999999996 & -0.131329468114936 & -0.000000000263296 & -0.000000000000088\\ 0.000000000000096 & 0.999991900032801 & 0.000000004009523 & 0.000000000002005\\ 0.000047994480423 & -0.000003151669857 & 0.999770026447968 & 0.000999635069641\\ 0 & 0 & 0 & 0.999500124979169\end{bmatrix}x(n)+1e\text{-}3.*\begin{bmatrix}-0.000000000000011\\ 0.000000000000334\\ 0.000249939175372\\ 0.499875020830729\end{bmatrix}u(n).....EqIII.8.$$

$$y(n)=\begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}x(n+1)$$



FigureIII.13: Building the Hardware Model

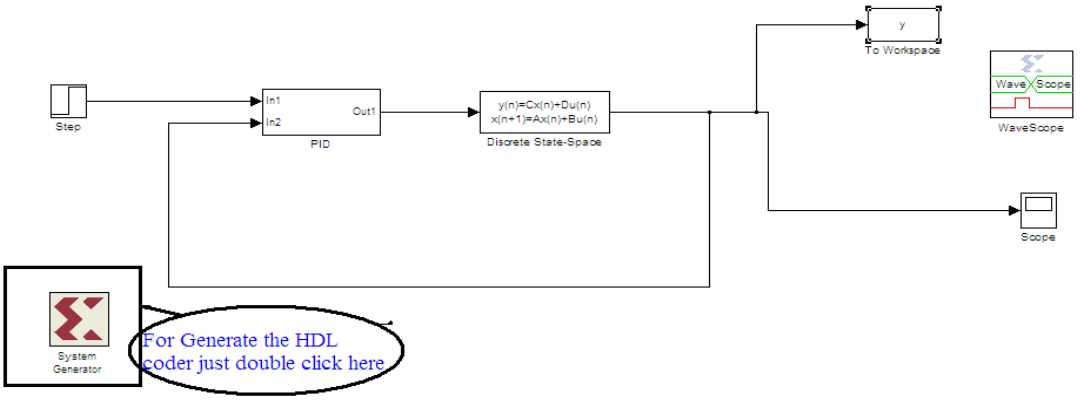
III.11.Building the Hardware Model

III.11.1.Generatio the HDL coder

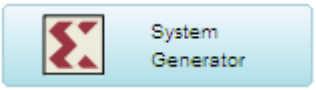
After we have finished modeling the system, we are ready to generate VHDL and cores for a Xilinx FPGA. We do this with the System Generator token from the Xilinx blockset.

III.11.2.System Generator block

• Xilinx System Generator Block: The first block to be used in any System Generator model is the System Generator Block. This block can be found in the following Simulink category: Xilinx Blockset→ Basic Elements→ System Generator. Place the System Generator block in the new model window as shown in **Figure III.14**



FigureIII. 14: Generation of HDL coder



III.11.3.Block Parameters Dialog Box

The block parameters dialog box can be invoked by double-clicking the block System Generator in our Simulink model.

Generate the VHDL code for the design, using the System Generator block with the following target information

- Compilation: HDL Netlist
- Part: Spartan3 xc3s200-4ft256
- Synthesis Tool: XST
- Target Directory: c:\xup\dsp_flow\labs\lab3\ise
- Create Testbench: Checked

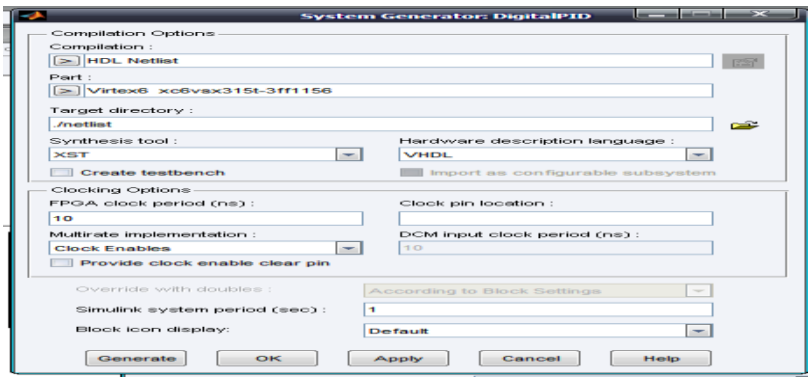


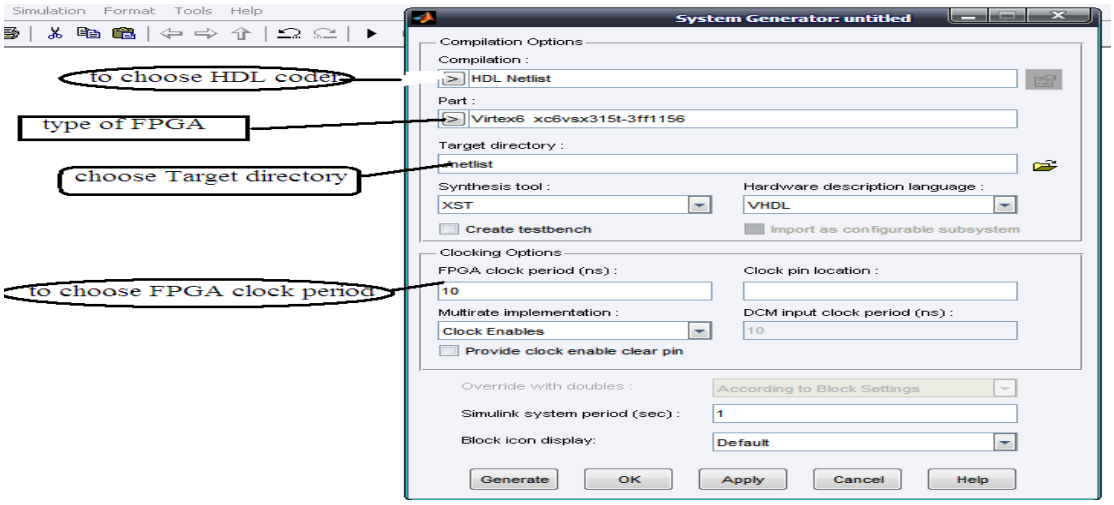
Figure III.15: System Generator block parameters dialog box

Browse to the current directory (c:\documents\Matlab\ netlist) as the Target Directory. Type in \ise at the end of the directory path to create a new sub-directory that will store the ISE project files generated from System Generator.

Specify HDL Netlist for Compilation and set the device related fields as:

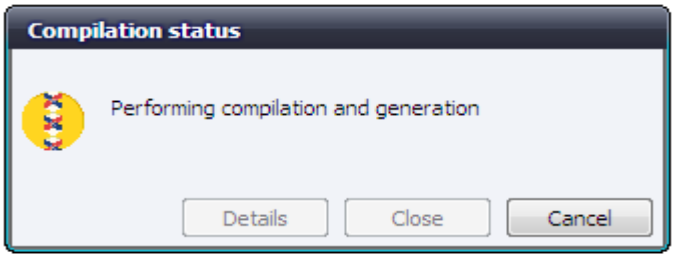
- Compilation: HDL Netlist
- Part: Spartan3 xc3s200-4ft256
- Synthesis Tool: XST

- The Apply button will save our selections and leave the window visible.

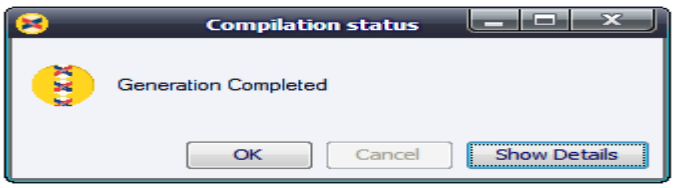


FigureIII.16: selection of HDl coder and browser

To Generate the HDL coder we click to Generate button



FigureIII.17.:Building the Design Netlist



FigureIII.18: Netlist building Complete

To generate VHDL and cores, -click on OK in the dialog box of Compilation status The OK button will save our selections and close the window. Invoking the Generate button generates our VHDL and cores

III.12.Compilation Options

1• Compilation: Specifies the type of compilation result that should be produced when the code generator is invoked. for more details.

2.Part: Defines the FPGA part to be used.

3•Target directory: Defines where System Generator should write compilation results. Because System Generator and the FPGA physical design tools typically create many files, it is best to create a separate target directory, other than the directory containing our Simulink model files.

4•Synthesis tool: Specifies the tool to be used to synthesize the design.

5•Hardware Description Language: Specifies the HDL language to be used for compilation of the design. The possibilities are VHDL and Verilog.

6•Create testbench: This instructs System Generator to create a HDL testbench. Simulating the testbench in an HDL simulator compares Simulink simulation results with ones obtained from the compiled version of the design. To construct test vectors, System Generator simulates the design in Simulink, and saves the values seen at gateways. The top HDL file for the testbench is named <name>_testbench.vhd where <name> is a name derived from the portion of the design being tested.

III.13.Clocking Options

1•FPGA clock period(ns): Defines the period in nanoseconds(ns) of the system clock. The value need not be an integer. The period is passed to the Xilinx implementation tools through a constraints file, where it is used as the global PERIOD constraint. Multicycle paths are constrained to integer multiples of this value.

2•Clock pin location: Defines the pin location for the hardware clock. This information is passed to the Xilinx implementation tools through a constraints file. This option should not be specified if the System Generator design is to be included as part of a larger HDL design.

3.Files Produced by System Generator Code Generation: From the System Generator block parameters dialog, choose a target directory, choose the Virtex device, make sure the Create Testbenchbox is checked, and click the Generate button. we will see taskbars showing the System Generator, then the Xilinx CORE Generator running.

. Now (in Windows Explorer or another file browser) view the files that have been written to our target directory. We will see the following files (among others):

¤ integrate.vhd - the top level VHDL file for our project. There are additional VHDL files included when our design has more hierarchy.

¤ integrate_xlmult_core1 - files associated with the generated multiplier core, such as the behavioral simulation models and edit file.

¤ corework - subdirectory containing the CORE Generator log file.

¤ integrate.npl- project file for opening the design in Xilinx ISE 12.1i Project Navigator using the XST synthesis compiler and ModelSim simulator.

¤ integrate_testbench.vhd- the top level VHDL testbench file associated with the top level VHDL source file in the project.

¤ integrate_<gateways>.dat – stimulus files for inputs to testbenchesor predicted outputs of testbenches. The .data files are generated by Simulink simulation and saved for running in Xilinx testbenches to verify design behavior. In this example <gateways> refers to the names of the Xilinx gateway blocks which collect and save the data.

¤ integrate_synplicity.prj - a project file for running this design in Synplify (synthesis tools from Synplicity).

¤ integrate_leon.tcl - a project file for running this design in Leonardo Spectrum (synthesis tools from Exemplar).

For a complete description of all of the files produced during code generation, please see Appendix B

III.14.Advantages of Using a Hardware Description Language (HDL) to Design FPGA Devices

Using a Hardware Description Language (HDL) to design high-density FPGA deviceshas the following advantages:

1. Top-Down Approach for Large Projects

Designers use a Hardware Description Language (HDL) to create complex designs. Thetop-down approach to system design works w ell for large HDL projects that requiremany designers working together . After the design team determines the overall designplan, individual designers can work independently on separate code sections

2.Functional Simulation Early in the Design Flow

You can verify design functionality early in the design flow by simulating the HDLdescription. Testing your design decisions before the design is implemented at theRegister Transfer Level (RTL) or gate level allows you to make any necessary changesearly on.

3.Synthesis of Hardware Description Language (HDL) Code to Gates

Synthesizing our hardware description to target the FPGA device implementation:

• Decreases design time by allowing a higher-level design specification, rather thanspecifying the design from the FPGA device base elements.

• Reduces the errors that can occur during a manual translation of a hardwaredescription to a schematic design.

• Allows you to apply the automation techniques used by the synthesis tool (suchas machine encoding styles and automatic I/O insertion) during optimization tothe original Hardware Description Language (HDL) code. This results in greateroptimization and efficiency .

4.Early Testing of Various Design Implementations

Using a Hardware Description Language (HDL) allows you to test different designimplementations early in the design flow . Use the synthesis tool to perform the logic synthesis and optimization into gates.

Xilinx® FPGA devices allow you to implement your design at your computer . Since thesynthesis time is short, you have more time to explore different architectural possibilitiesat the Register Transfer Lev el (RTL) You can reprogram Xilinx FPGA devices to testseveral design implementations.

5. Reuse of Register Transfer Level (R TL) Code

We can retarget Register Transfer Lev el (RTL) code to new FPGA devices withminimum recoding.

6. DesigningFPGA Devices With Hardware Description Language (HDL)

To effectively use an HDL, we must understand the:

• Syntax of the language.

• Synthesis and simulator tools.

• Architecture of our target device.

• Implementation tools.

III.15.Field Programmable Gate Array (FPGA)

A Field Programmable Gate Array (FPGA) is a programmable logic device that supports implementation of relatively large logic circuit. FPGAs are quite different from Simple Programmable Logic Devices (SPLDs) and Complex Programmable logic devices (CPLDs) because FPGAs don't contain AND or ORplanes. Instead, FPGAs provide logic blocks for implementation of required functions [22].

The general structure of an FPGA contain three main types of resources logic blocks, input/output blocks connecting to the pins of the package, and interconnection wires and switches.FPGA can be used to implement logic circuits of more than a few hundred thousand equivalent gates in size.

Each logic block in an FPGA typically has a small number of inputs and one output. A number of FPGA products are on the market, featuring different types of logic blocks. The most commonly used logic block is a Lookup Table (LUT), which contains storage cells that are used to implement a small logic function, each cell is capable of holding a single logic value, either 0 or 1. The stored value produced as the output of the storage cell. LUTs of various sizes may be created, where the size is defined by the number of inputs.

The software environments that represent the software package of FPGA have two main types. Firstly, Foundation series and secondly Navigator series.

The design of logic circuit could be done with above software by two ways, either schematically or Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL)[23].

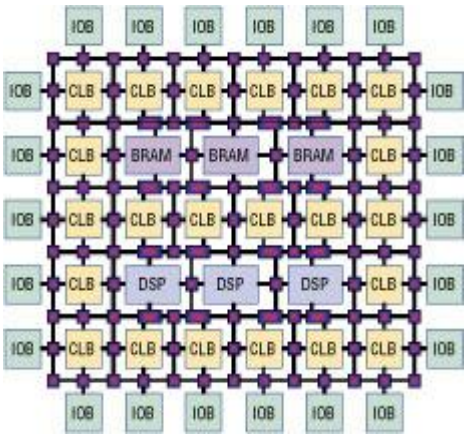


Figure III.19.: FPGA internal structure based on the Xilinx architecture

A typical FPGA logic block consists of a four-input lookup table (LUT) and flip-flop. Currently, FPGA consists of DSP blocks which have high-level functionality embedded into the silicon, high-speed IOBs, embedded memories and processors. It also contains Configurable logic blocks (CLBs) which comprised of multiple slices. A slice is a small set of building blocks. Moreover, modern FPGA consist of tens of thousands of CLBs and a programmable interconnected network in a rectangular grid ASICs (Application specific integrated circuits) is typically performed a single function throughout the lifetime of a chip while in FPGA, it can be reprogrammed in such a way that it can perform function in micro-seconds. Source code written in a hardware description language (HDL) such as Verilog and VHDL provides the functionality to perform tasks at run time. Synthesis process generates technology-mapped netlist (**FigureIII.19**). A map place and route process then fits the netlist to the actual FPGA in architecture. The process produces a bitstream which is used to reconfigure FPGA. Timing, postsynthesis, functional simulations and verifications methodologies can validate map place and route results.

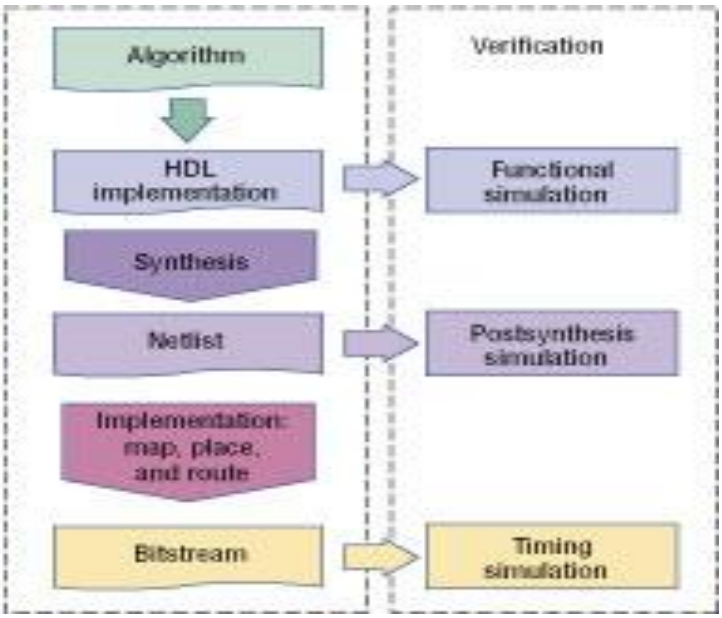


Figure III.20: Typical design flow of FPGA

FPGA supports the notion of reconfigurable computing and provides a facility of on-chip parallelism which can be mapped directly from the dataflow characteristics of an application's parallel algorithm. A recent emergence in high performance can be achieved by a hybrid approach to make a complex-system on a programmable chip. Examples are Virtex II Pro, Virtex-4 and Xilinx Devices. The most recent success of FPGAs in high-performance computing came under Tsubame cluster in Tokyo when FPGAs increased performance by additional 25%[23].

III.16.Progress in hardware system and programming software

With the emergence of technologies, many hardware systems have begun to resemble parallel computers. These systems are not designed for scalability because they consisted of a single board of one or more microprocessors connected to one or more FPGA devices. Recently, SRC-6 and SRC-7 have a parallel architecture in which used cross bar switch that can be piled for further scalability. Traditionally, high-performance computing vendors – specifically, Silicon Graphics Inc. (SGI), Cray and Linux Network have incorporated FPGAs in their parallel architectures [23].From software perspective, developers can create the hardware kernel by using hardware description languages such as VHDL and Verilog. SRC Computers allow other hardware description languages including Carte C, Carte Fortran, Impulse Accelerated Technologies' Impulse C, Mitrion C from Mitrionics, and Celoxica's Handel-C. Annapolis Micro Systems' CoreFire, Starbridge Systems' Viva, Xilinx System Generator and DSPlogic's reconfigurable computing toolbox are the high-level graphical programming development tools.

III.17.Advantages of FPGAs

There are number of advantages using FPGAs including speed, reduced energy power consumption. As in reconfigurable computing, hardware circuit is optimized with the application so that the power consumption will tend to be much lower than that for a general-purpose processor. FPGAs have other advantages which comprised of reduction in size, component count (and hence cost), improved time-to-market and improved flexibility and extendibility. These advantages are especially important for embedded applications[24].

III.18.Limitations of FPGAs

There are number of challenges in implementing reconfigurable computing. described three such challenges.

1. Structure of reconfigurable fabric
2. Interfaces between the fabric, processor(s)
3. Memory must be efficient

There is another challenge regarding the development of computer-aided design and compilation tools that map an application to a reconfigurable computing. The problem is related to know about which part of the application is mapped to the fabric and which should be mapped to the processor .

Few limitations of FPGAs in high performance computing should be addressed. These issues include the need of programming tools that address the overall architecture, profiling and debugging tools for parallel and reconfigurable performance. Furthermore, application-portability issues should be required to explore[24].

III.19.Who makes FPGAs?

There are (at least) 5 companies making FPGAs in the world. The first two (Xilinx and Altera) hold the bulk of the market.

- Xilinx is the biggest name in the FPGA world. It tends to be the density and technology leader.
- Altera is the second FPGA heavyweight, also a well-known name.
- Lattice, Actel, Quicklogic are much smaller and are the "specialty shops".

Xilinx 

Xilinx has traditionally been the silicon technology leader. Xilinx general philosophy is to provide all the features possible, at the cost of extra complexity.

- Biggest and most flexible (feature-full) devices.
- Complex architecture, powerful devices.

Altera

Altera philosophy is to provide the features that most people want while keeping their devices easy to use.

- Lean and efficient devices architecture.
- Powerful devices.,Lattice, Actel and Quicklogic

These companies have specialty products.

- Lattice, better known for its CPLDs, have also an "instant-on" FPGA family.
- Actel and QuickLogic have antifuse (programmable-only-once) products.

III.20.FPGA application design techniques

The application design techniques to enable substantial FPGA acceleration. These methods are as follows:

1. Use an algorithm optimal for FPGAs.
2. Use a computing mode appropriate for FPGAs.
3. Use appropriate FPGA structures.
4. Living withAmdahl’slaw.

5. Hide latency of independent functions.
6. Use rate techniques to remove bottlenecks.
7. Take advantage of FPGA-specific hardware.
8. Use appropriatearithmeticprecision.
9. Use appropriatearithmetic mode.
10. Minimize use of high-cost arithmetic operations.
11. Create families of applications, not point solutions.
12. Scale application for maximal use of FPGA hardware.

III.21.Copacobana Architecture

It consists of three basic blocks including one controller module, up to 20 FPGA modules and backbone which is used for providing interconnection between controller and FPGA modules (**Figure III.21**)[25].

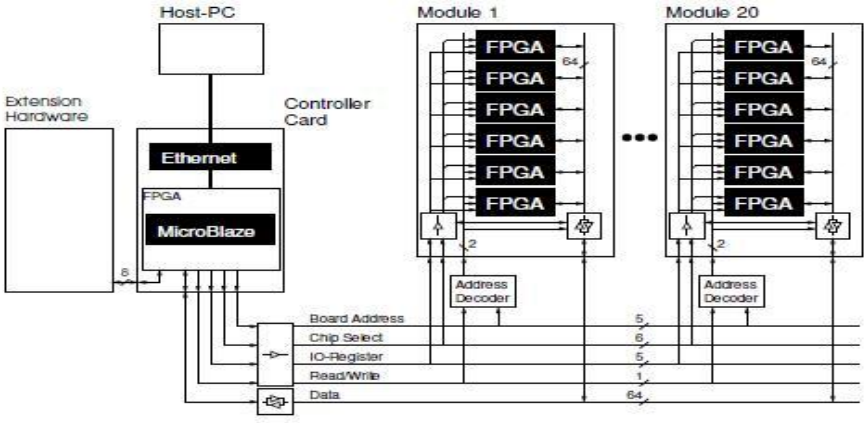


Figure III.21: Architecture of Copacobana

The FPGAs are directly connected to a common 64-bit data bus on board of the FPGA module which is interfaced to the backplane data bus via transceivers with 3-state outputs. While disconnected from the bus, the FPGAs can communicate locally via the internal 64-bit bus on the DIMM module. The DIMM format allows for a very compact component layout, which is important to closely connect the modules by a bus. Every FPGA module is assigned a unique hardware address, which is accomplished by Generic Array Logic (GAL) attached to every DIMM socket. Hence, all FPGA cores can have the same configuration and all FPGA modules can have the same layout. They can easily be replaced in case of a defect.

III.21.Conclusion

In this chapter describes two workflows. In the first, the design is created with blocks from both Simulink and System Generator for DSP. In the second, a Xilinx System Generator for DSP Black Box is generated from a MATLAB HDL design, to design FPGA .

We built a Digital PID controller by the System Generator Tools. And Generate a HDL coder, Talking about this step by step, after finding the results in the scope wave ,looking this results in the chapter IV.