FPGA Based Hardware-in-the Loop Controller for Electric Drives

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Visweshwar Chandrasekaran

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Advisor: Prof. Ned Mohan

December, 2013

Acknowledgements

First and foremost, I thank my advisor Prof. Ned Mohan for giving me the opportunity to work on this project and for his guidance towards completing it to satisfaction. Prof. Mohan's interest and enthusiasm for the project has been the primary driving force, acting as a great source of motivation to constantly innovate while also maintaining a clear objective. I would also like to acknowledge his contribution towards developing the Electric Drives Laboratory, which is a great learning arena for electric machines and power electronics.

I thank Prof. Thomas Posbergh for his continued support through all legs of the project with his cheerful demeanor and shrewd observations.

I thank Prof. Sairaj Dhople and Prof. Rajesh Rajamani in their capacity as members of the examining committee and their support and enthusiasm in the project undertaken.

I thank Bryan Dalley for helping me kick start this project with his earlier contributions. I would also like to thank Santhosh Krishnamoorthi for his contributions and for constantly acting as a sound board for the many discussions that were part of the project.

Finally I owe it to my research group and to my peers for their immense support and invaluable guidance at various stages. They provided a very friendly atmosphere for research and often times provided great insight into many aspects of this thesis to which I am very grateful.

Dedication

To my parents, sister and teachers.

Abstract

Research and industrial implementation of Digital Signal Processor (DSP) based electric drive controllers continue to increase each year. DSP-based controllers are typically implemented on FPGA's or DSP microcontrollers by using proprietary development software. FPGA's have advantages over DSP microcontrollers for control due to their parallel processing capability and flexible architectures. An FPGA based motor control system was developed using the *Matlab Simulink Toolbox: Xilinx System Generator for DSP*. A customized library was developed that has many common blocks used in the development of drive models. An FPGA board based on Xilinx's Spartan 6 family was also developed which can be used with a PC for hardware-in-the-loop co-simulation. The controller can be operated in a DSP-based Electric Drives Laboratory that is currently using a dSPACE rapid prototyping system. This is hence a cost-effective replacement that still provides the full processing power of dSPACE controllers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Digital Controllers for Drive Applications

Power Electronics has become an emerging field with an increasing application towards control of motor drives that is used in many areas of the industry like automation, renewable energy, automotive, aerospace, medical etc. The many different applications pose different challenges to electric drive design such as precision motion control for factory automation or high efficiency drives in power generation applications. Drive systems is also a multi-disciplinary field incorporating areas such as machine theory, power electronics, control theory, real-time Digital Signal Processing (DSP) control, mechanical system modeling, sensors and utility interaction. There has been an increasing need for smart control of electric motors to provide a wide range of operation with enhanced operation that reduces the energy demand. The methods to design electric drives have been constantly improving. For the purposes of research and development there is a popular demand for rapid design and prototyping of drive systems to reduce the development time as well as provide the user more features to designing such systems[4][5][6]. The market has opened up for powerful yet inexpensive digital controllers like microcontrollers, DSPs, FPGAs etc. These controllers significantly reduce the physical design complexity by reducing the component count on hardware designs. The latest controllers also come with high speed architectures with many peripherals that are relevant to motor control applications.

Digital control systems consist of software and hardware components which are optimized to execute algorithms quickly and efficiently. The algorithms may involve moderate to complex mathematical calculations, logical and relational operations and communication protocols. There are also operations involving interpretation of motor

encoder information, conversion of analog signals into a digital format and vice versa. These algorithms can be compiled into an optimized code for a microcontroller which in turn connects with power electronic converters and measurement circuits to operate various motors. The digital control system can then be used to dictate the position, speed and torque of the motors based on a velocity and load profile.

1.2 Reinventing the Electric Drives Laboratory

Due to the diverse subject, application challenges, and sometimes abstract mathematical method of teaching electric drive design, Professor Ned Mohan of the University of Minnesota developed a practical method to teaching the subject based on his book *Electric Drives, An Integrative Approach* [1] and provided understanding of advanced topics on drives in [2]. The corresponding laboratory for the Electric Drives course, a DSP-based Electric Drives Laboratory [3], follows the same principles as the course. Students are educated on the principles of developing electric drives for controlling various DC, Induction and Permanent Magnet AC motors (PMAC). The lab setup uses four primary components to perform Electric Drive experiments for controlling motors including: a motor coupling system, Power Electronics Drive Board, *dSPACE Inc.* DS1104 R&D controller Board with CP 1104 I/O Board, and software packages Matlab Simulink and *dSPACE, Inc.* ControlDesk.

The existing *dSPACE, Inc.* DSP system is an advanced development system that abstracts much of the information on how the controllers are implemented in the laboratory by using the control-desk tool, integrated with Matlab Simulink, to generate the designed controllers in the C programming language. The program is downloaded to a Texas Instruments TMS320F240 DSP microcontroller on the DS1104 R&D controller board to implement the controller.

A new FPGA based rapid prototyping system has been developed as a low cost alternative which provides the processing power of dSPACE controllers, a popular tool for research and development of control solutions. It is targeted for development of electric drive and power electronic control implementations. The system is customized from a DSP based design tool called System Generator from Xilinx. It enables the use of Simulink's model-based design environment Simulink for FPGA design. Previous experience with Xilinx FPGAs or RTL design methodologies is not required when using System Generator. Designs are implemented in the DSP friendly Simulink modeling environment using a Xilinx blockset. All of the downstream FPGA implementation steps including synthesis and place and route are automatically performed to generate an FPGA programming file. A highly customized library of HDL blocks has been created that meets all the needs of the developer for the above said applications. A hardware platform has been created that has a Spartan 6 FPGA controller at the core with peripheral circuits such as ADC, PROM, quadrature encoder, digital I/O and PWM outputs. These controller interfaces with the present electric drives hardware circuit. Model simulation results are "bit and cycle accurate". For rapid prototyping and user interaction, hardware co-simulation is employed using a high speed JTAG link. Using the Xilinx blockset and the customized library the nine experiments in the Electric Drives Laboratory have been developed and tested. The system provides a similar level of user interaction and simplicity as that of dSPACE while substantially reducing the hardware costs associated with the latter.

1.2 Scope and organization of this thesis

This thesis presents a system for designing electric drives and power electronic converters with the ability to do the following:

- Powerful FPGA based controller which can perform high speed complex tasks like fixed and floating point arithmetic, logical and relational operations,

communication protocols etc.

- Simulation within Simulink for easy construction of algorithms.

- A highly customized toolbox for designing motor models and control loops.

- Hardware-in-the-Loop Co-simulation for improved usability for rapid development and as a powerful debugging tool for fresh designs.

- Highly customized FPGA control card that integrates with the existing hardware in the electric drives laboratory (EE 4703).

The thesis is organized as follows:

1. Chapter 1: Introduction

2. Chapter 2 presents a detailed description of the System Generator toolbox for Matlab/Simulink with details on the design procedure for the custom library toolbox that has been developed.

3. Chapter 3 introduces the application of the FPGA based control solution in building algorithms for open loop and closed loop control of DC and AC machines.

4. Chapter 4 details the design specifications of a custom developed FPGA control card for the electric drives lab.

5. Chapter 5 summarizes the development of the new system with a discussion on the possible applications and future work for this thesis.

Chapter 2

Model based design using System Generator

Xilinx, Inc. has developed a Matlab Simulink blockset, System Generator for DSP, to facilitate the development of DSP systems on Xilinx FPGA's[7]. The system enables Hardware-In-the-Loop (HIL) testing of the FPGA's with the added capability of co-simulating the system in real-time with the Simulink model. Although the Xilinx System Generator abstracts the hardware implementation of the DSP models in the FPGA, the blockset can be easily followed to understand the operation and implementation of the controller. Additionally, multiple Hardware Co-Simulation models exist for many prototyping boards or they can be easily set up for a custom made board containing a Xilinx FPGA. The ability to develop models for custom boards enables users to test systems on their own development boards or even on the actual board to be used in implementation which may also lower costs of development.

2.1 A Brief Introduction to FPGAs

A field programmable gate array (FPGA) is a general-purpose integrated circuit that is "programmed" by the designer rather than the device manufacturer. Unlike an application-specific integrated circuit (ASIC), which can perform a similar function in an electronic system, an FPGA can be reprogrammed, even after it has been deployed into a system. An FPGA is programmed by downloading a configuration program called a bitstream into static on-chip random-access memory. Much like the object code for a microprocessor, this bitstream is the product of compilation tools that translate the high-level abstractions produced by a designer into something equivalent but low-level and executable. Xilinx System Generator is a powerful tool for compiling an FPGA program from a high-level Simulink model based design.

An FPGA provides you with a two-dimensional array of configurable resources that can implement a wide range of arithmetic and logic functions. These resources include dedicated DSP blocks, multipliers, dual port memories, lookup tables (LUTs), registers, tristate buffers, multiplexers, and digital clock managers. In addition, Xilinx FPGAs contain sophisticated I/O mechanisms that can handle a wide range of bandwidth and voltage requirements.

FPGAs are high performance data processing devices. DSP performance is derived from the FPGA's ability to construct highly parallel architectures for processing data. In contrast with a microprocessor or DSP processor, where performance is tied to the clock rate at which the processor can run, FPGA performance is tied to the amount of parallelism that can be brought to bear in the algorithms that make up a signal processing system. A combination of increasingly high system clock rates (current system frequencies of 100-200 MHz are common today) and a highly-distributed memory architecture gives the system designer an ability to exploit parallelism in DSP (and other) applications. For example, the raw memory bandwidth of a large FPGA running at a clock rate of 150 MHz can be hundreds of terabytes per second.

There are many DSP applications (e.g., digital up/down converters) that can be implemented only in custom integrated circuits (ICs) or in an FPGA; a von Neumann processor lacks both the compute capability and the memory bandwidth required.
Advantages of using an FPGA include significantly lower non-recurring engineering costs than those associated with a custom IC (FPGAs are commercial off-the-shelf devices), shorter time to market, and the configurability of an FPGA, which allows a design to be modified, even after deployment in an end application.

When working in System Generator, it is important to keep in mind that an FPGA has many degrees of freedom in implementing signal processing functions. You have, for example, the freedom to define data path widths throughout your system and to employ

many individual arithmetic engines (accumulators, multipliers etc.), depending on system requirements. System Generator provides abstractions that allow you to design for an FPGA largely by thinking about the algorithm you want to implement. However, the more you know about the underlying FPGA, the more likely you are to exploit the unique capabilities an FPGA provides in achieving high performance.

2.2 Design Flows using System Generator

System Generator can be useful in many settings. Sometimes you may want to explore an algorithm without translating the design into hardware. Other times you might plan to use a System Generator design as part of something bigger. A third possibility is that a System Generator design is complete in its own right, and is to be used in FPGA hardware. This topic describes all three possibilities.

2.2.1 Algorithm Exploration

System Generator is particularly useful for algorithm exploration, design prototyping, and model analysis. When these are the goals, you can use the tool to flesh out an algorithm in order to get a feel for the design problems that are likely to be faced, and perhaps to estimate the cost and performance of an implementation in hardware. The work is preparatory, and there is little need to translate the design into hardware.

In this setting, you assemble key portions of the design without worrying about fine points or detailed implementation. Simulink blocks and MATLAB M-code provide stimuli for simulations, and for analyzing results. Resource estimation gives a rough idea of the cost of the design in hardware. Experiments using hardware generation can suggest the hardware speeds that are possible.

Once a promising approach has been identified, the design can be fleshed out. System

Generator allows refinements to be done in steps, so some portions of the design can be made ready for implementation in hardware, while others remain high-level and abstract. System Generator's facilities for hardware co-simulation are particularly useful when portions of a design are being refined.

2.2.2 Implementing Part of a Larger Design

Often System Generator is used to implement a portion of a larger design. For example, System Generator is a good setting in which to implement data paths and control, but is less well suited for sophisticated external interfaces that have strict timing requirements. In this case, it may be useful to implement parts of the design using System Generator, implement other parts outside, and then combine the parts into a working whole.

A typical approach to this flow is to create an HDL wrapper that represents the entire design, and to use the System Generator portion as a component. The non-System Generator portions of the design can also be components in the wrapper, or can be instantiated directly in the wrapper.

2.2.3 Implementing a Complete Design

Many times, everything needed for a design is available inside System Generator. For such a design, pressing the **Generate** button instructs System Generator to translate the design into HDL, and to write the files needed to process the HDL using downstream tools.

The files written include the following:

- HDL that implements the design itself;

- A clock wrapper that encloses the design. This clock wrapper produces the clock and clock enable signals that the design needs.

- A HDL testbench that encloses the clock wrapper. The testbench allows results from Simulink simulations to be compared against ones produced by a logic simulator.

- Project files and scripts that allow various synthesis tools, such as XST and Synplify Pro to operate on System Generator HDL

- Files that allow the System Generator HDL to be used as a project in Project Navigator [8].

2.3 System Generator Blocks

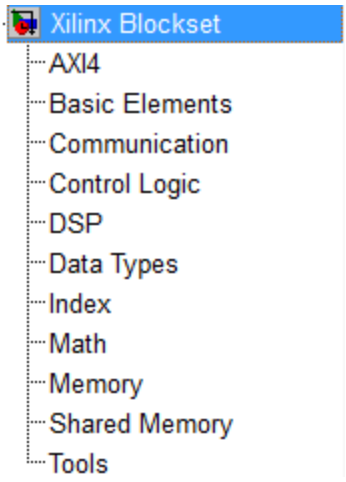
A *Simulink blockset* is a library of blocks that can be connected in the Simulink block editor to create functional models of a dynamical system. For system modeling, System Generator blocksets are used like other Simulink blocksets. The blocks provide abstractions of mathematical, logic, memory, and DSP functions that can be used to build sophisticated signal processing (and other) systems. There are also blocks that provide interfaces to other software tools (e.g., FDATool, ModelSim) as well as the System Generator code generation software. System Generator blocks are *bit-accurate* and *cycle-accurate*.

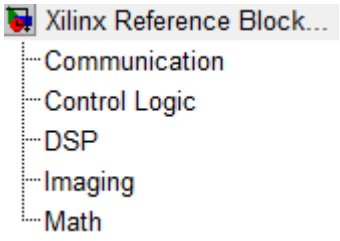
Bit Accuracy: Bit-accurate blocks produce values in Simulink that match corresponding values produced in hardware. The signal value and its accuracy is determined by the setting of various parameters in the blocks such as Signal Type, Number of Bits, and Binary Point as well as settings for Quantization and Overflow.

Cycle Accuracy: To say a simulation is cycle-accurate means that at the boundaries, corresponding values are produced at corresponding times. The boundaries of the design are the points at which System Generator gateway blocks exist. When a design is translated into hardware, Gateway In (respectively, Gateway Out) blocks become top-level input (resp., output) ports.

2.3.1 Xilinx Blockset and Xilinx Reference Blockset

The Xilinx Blockset is a family of libraries that contain basic System Generator blocks. Some blocks are low-level, providing access to device-specific hardware. Others are high-level, implementing (for example) signal processing and advanced communications algorithms. For convenience, blocks with broad applicability (e.g., the Gateway I/O blocks) are members of several libraries. Every block is contained in the Index library as shown in Fig. 2.1(a) and Fig. 2.1(b). Below is listed the various components of the Xilinx blockset and the Xilinx Reference Blockset.





The Xilinx Blockset has the following categories:

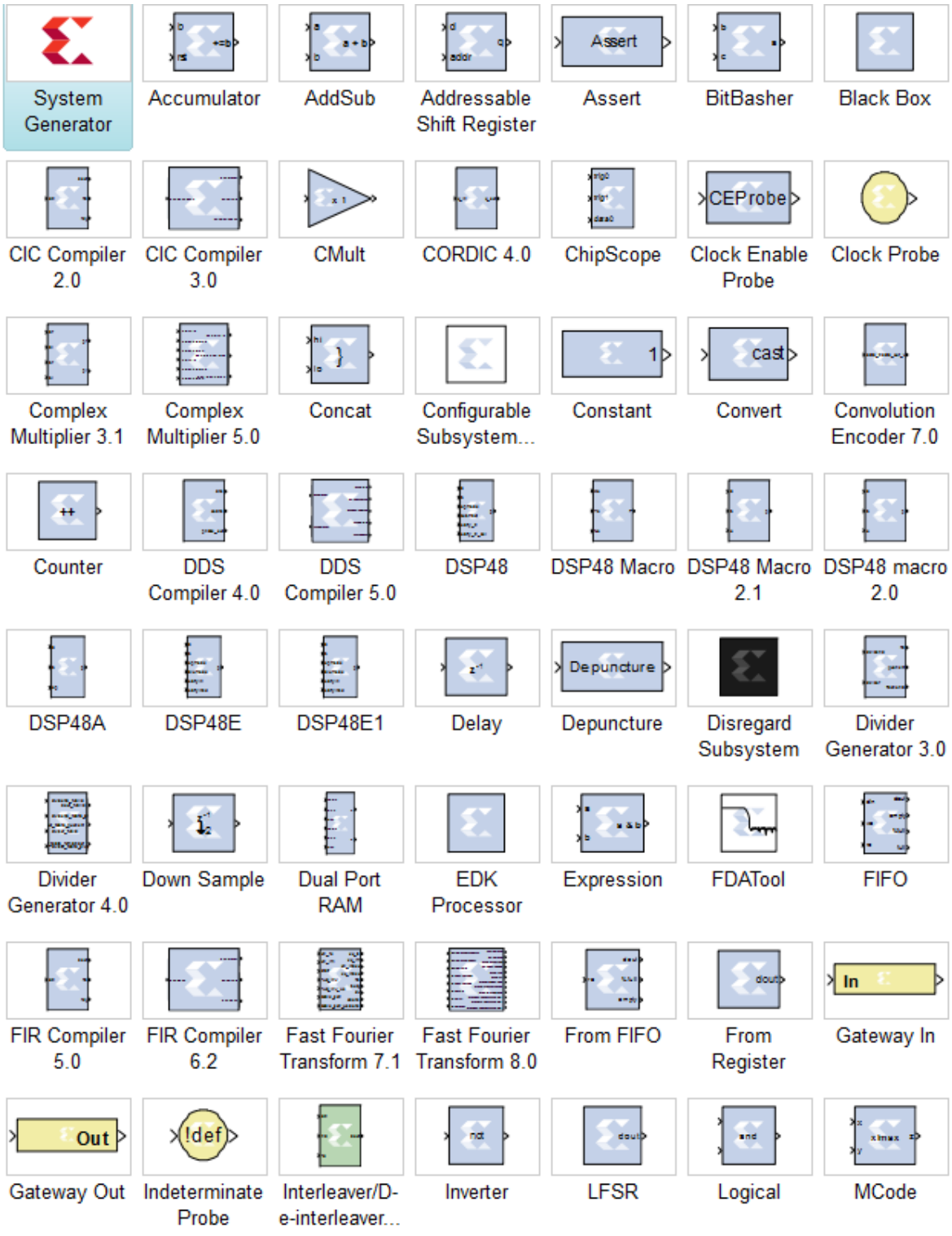
- AX14: Blocks with interfaces that conform to the AX^{TM}4 specification.
- Basic Elements: Standard elements for building blocks for digital logic.

- Communication: Forward error correction and modulator blocks, commonly used in digital communication systems.
- Control Logic: Blocks for control circuitry and state machines.
- DSP
- Data Types: Blocks that convert data types (includes gateways).
- Floating-Point: Blocks that support the Floating-Point data type.
- Index: Every block in the Xilinx Blockset.
- Math: Blocks that implement mathematical functions.
- Memory: Blocks that implement and access memories.
- Shared Memory: Blocks that implement and access Xilinx shared memories.
- Tools: "Utility" blocks, e.g., code generation (System Generator toke), resource estimation, HDL-co-simulation, etc.

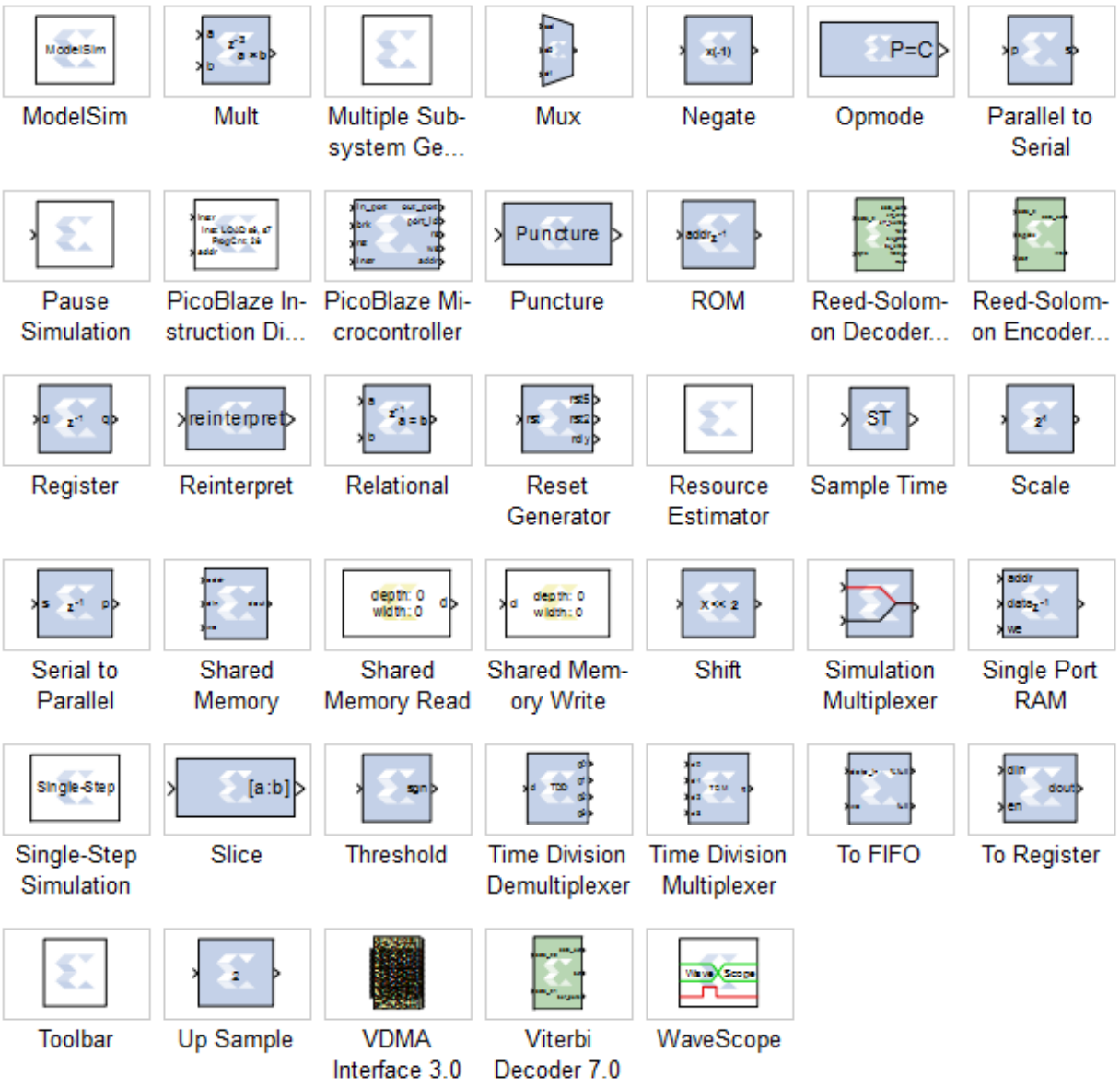
The Xilinx Reference Blockset contains composite System Generator blocks that implement a wide range of functions. Blocks in this blockset are organized by function into different libraries. The libraries are described below:

- Communication: Blocks commonly used in digital communication systems
- Control Logic: Logic blocks used for control circuitry and state machines
- DSP: Digital signal processing (DSP) blocks.
- Imaging: Image processing blocks
- Math: Blocks that implement mathematical functions.

Each block in this blockset is a composite, i.e., is implemented as a masked subsystem, with parameters that configure the block.


System Generator
Accumulator
AddSub
Addressable Shift Register
Assert
BitBasher
Black Box
CIC Compiler 2.0
CIC Compiler 3.0
CMult
CORDIC 4.0
ChipScope
CEProbe
Clock Enable Probe
Clock Probe
Complex Multiplier 3.1
Complex Multiplier 5.0
Concat
Configurable Subsystem...
Constant
cast
Convert
Convolution Encoder 7.0
Counter
DDS Compiler 4.0
DDS Compiler 5.0
DSP48
DSP48 Macro
DSP48 Macro 2.1
DSP48 macro 2.0
DSP48A
DSP48E
DSP48E1
Delay
Depuncture
Disregard Subsystem
Divider Generator 3.0
Divider Generator 4.0
Down Sample
Dual Port RAM
EDK Processor
Expression
FDATool
FIFO
FIR Compiler 5.0
FIR Compiler 6.2
Fast Fourier Transform 7.1
Fast Fourier Transform 8.0
From FIFO
From Register
In
Gateway In
Out
Gateway Out
!def
Indeterminate Probe
Interleaver/D-e-interleaver...
Inverter
LFSR
Logical
MCode

(a)



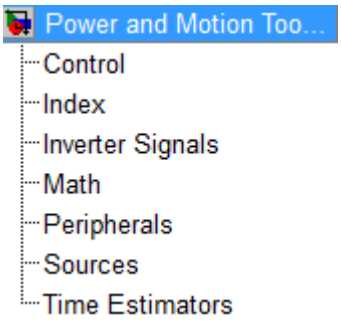
(b)

Fig. 2.1 (a): Xilinx Blockset Library; (b): Xilinx Blockset contd.

2.3.2 Power and Motion Toolbox

For the purpose of the Electric Drives lab the Power and Motion Toolbox has been created which has a complete set of blocks that can be used to create drives models. The blocks

have been created using the Xilinx blockset library using the concept of masked subsystems. The blocks are shown in Fig. 2.2 and are categorized as follows:



- Control: Blocks useful in closed loop designs
- Inverter Signals: Blocks used in operation of power hardware
- Math: Blocks that implement mathematical functions.
- Peripherals: Blocks that implement ADC and encoder communications.
- Sources: Blocks providing data sources
- Time Estimators: Blocks that calculate time between two values of a signal

Fig. 2.2: Power and Motion Toolbox

2.4 Signal Type, Precision and Polymorphism

In order to provide bit-accurate simulation of hardware, System Generator blocks operate on Boolean, floating-point, and arbitrary precision fixed-point values. By contrast, the fundamental scalar signal type in Simulink is double precision floating point. The connection between Xilinx blocks and non-Xilinx blocks is provided by *gateway blocks*. The *gateway in* converts a double precision signal into a Xilinx signal, and the *gateway out* converts a Xilinx signal into double precision. Simulink continuous time signals must be sampled by the Gateway In block.

Fig. 2.3 shows a simple connection example between Xilinx and Simulink blocks through the *gateway blocks.*

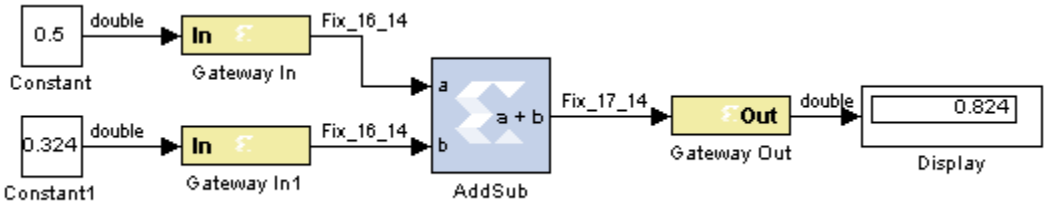


Fig. 2.3: Interconnection between Simulink and Xilinx blocks

Note: To determine the signal data type for each connection turn on Port Data Types under Format>Port/Signal Displays.

As seen in the figure the Simulink blocks have double precision floating point signals while the signal data type of the Xilinx blocks can be selected under the block parameters. In this example, the Gateway In block has an ouput as “Fix_16_14” which implies that the signal is of type fixed point with 16 total bits and the binary point at the 14^{th} bit. Hence there are 2 bits for integer precision and the rest 14 for fractional precision. The Gateway Out block reconverts Xilinx block data types to the Simulink standard double precision floating point type.

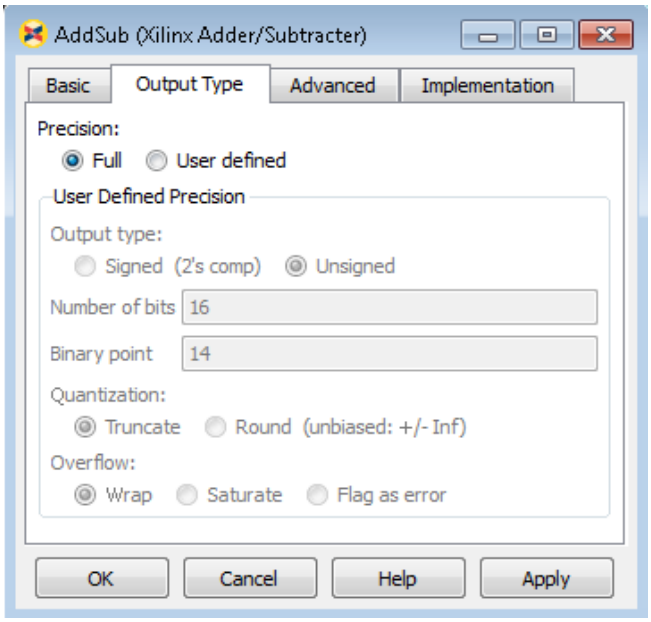


Fig. 2.4: AddSub Block Parameter Dialog Box

Most Xilinx blocks are polymorphic, i.e., they are able to deduce appropriate output types based on their input types. When *full precision* is specified for a block in its parameters dialog box, System Generator chooses the output type to ensure no precision is lost. Sign extension and zero padding occur automatically as necessary. In the example above, the AddSub block's output precision is set at "Full" as shown in Fig. 2.4. Hence the output type inherits the data type of the two inputs to the block and hence has a type "Fix_17_14".

User-specified precision is usually also available. This allows you to set the output type for a block and to specify how quantization and overflow should be handled. Quantization possibilities include unbiased rounding towards plus or minus infinity, depending on sign, or truncation. Overflow options include saturation, truncation, and reporting overflow as an error.

2.5 Enhancing the usability of Xilinx blocks

The Xilinx blockset is a powerful tool for building models for implementation in FPGAs. However there is a complexity in the parameter settings of certain blocks that make the designs prone to incorrect data propagation through certain operations in the model. Two such situations are discussed below.

2.5.1 Incorrect data due to user negligence

As discussed in the previous section, the Xilinx blocks have various parameters that need to be accurately set to prevent loss of data precision. To demonstrate a situation that produces an incorrect output because of wrong parameter settings a simple arithmetic operation with wrong parameter settings is shown in Fig. 2.5.

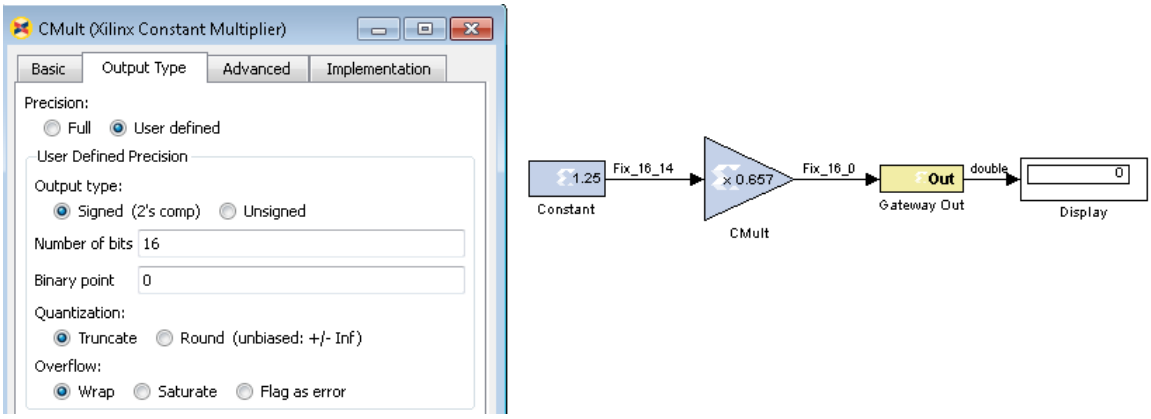


Fig. 2.5: Operation with incorrect parameters

The Binary point of the Output type of the “CMult” block has been accidently set to 0. This produces a wrong result of the product of the constant value 1.25 with 0.657 as shown in the figure. Such an error can cause bad outputs of arithmetic operations and hence proper care needs to be taken to set the data type settings correctly. The subsequent

sections will explain a custom optimization that has been undertaken to prevent such issues from occurring while performing simulations within System Generator.

2.5.2 Magnification of data widths

The user needs to carefully plan the allocation of bits to the data as well as keep in mind the factors such as quantization and overflow. Another issue is the magnification of data width with certain arithmetic operations such as multiplication. For example, the product of a Fix_16_14 with a Fix_16_14 is a Fix_32_28. This magnification is a fundamental issue of fixed point arithmetic and causes large signals in the model that the user is implementing as shown in Fig. 2.6.

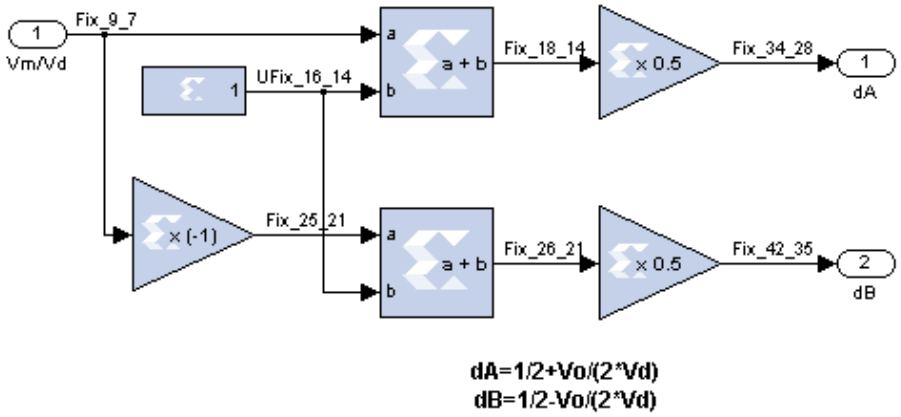


Fig. 2.6: Using default Xilinx blocks

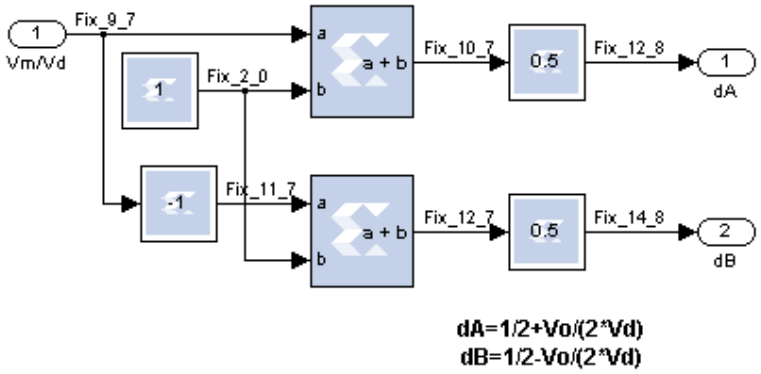


Fig. 2.7: Using Power and Motion Toolbox blocks

2.5.3 Optimizing data widths

The two issues discussed above could be critical in implementation of various arithmetic equations as incorrect data width setting would cause loss in data as well as reduced precision. To overcome these drawbacks of the Xilinx blockset a routine to calculate data width and precision has been developed.

Case study: The Constant block

As mentioned in section 2.3.2, a custom library called “Power and Motion Toolbox” has been specifically designed for drive applications. One particular component of the library is the Constant block. Fig. 2.8 shows the parameter windows of the default Xilinx blockset Constant and the Power and Motion Toolbox Constant.

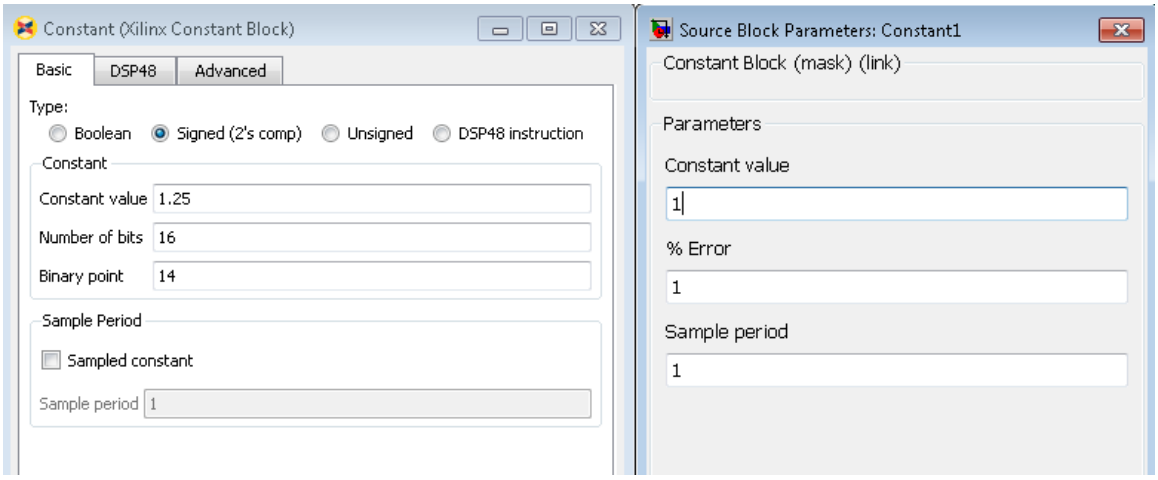


Fig. 2.8 Parameter dialog boxes of Xilinx and Power and Motion Toolboxes

The Power and Motion Toolbox Constant has fewer number of parameter fields as compared to the Xilinx blockset’s equivalent and it closely mirrors the Simulink Constant. The Number of bits and Binary point which were explicit fields in the former has been hidden from the user and is calculated within the mask of the block. The field “% Error” is responsible for the bit settings as it determines the accuracy of the value and

hence the number of bits needed to represent the constant value to within the percentage of error that the user specifies. The following code snippet shows how the data width is calculated considering the value of the constant and the percentage error of the output.

Code Snippet:

```
pos = 0;
neg = 1;
flagint = 1;
flagfrac = 1;
subflag = 1;
g = 1;

[intval fracval] = deal(abs(fix(constant)),abs(constant-
fix(constant)));

while (flagint == 1)
    if (2^pos-1 < abs(intval))
        pos = pos+1;
        flagint = 1;
    else
        flagint = 0;
    end
end
if (fracval == 0)
    neg = 0;
    flagfrac = 0;
    subflag = 0;
end
while (flagfrac == 1)

    if((1/2^neg)>fracval && neg<=20)
        neg = neg+1;
        flagfrac = 1;
    else
```

```
          flagfrac = 0;
    end
end
accum = 1/2^neg;
while (subflag == 1)

    error = (fracval - accum)/fracval*100;

    if(abs(error)>err && neg<=20)
        if (g == 1)
            neg = neg+1;
        end
        past = accum;
        accum = accum + 1/2^neg;
        if(accum>fracval)
            accum = past;
            neg = neg+1;
            g = 0;
        else
            subflag = 1;
            g = 1;
        end
    else
        subflag = 0;
    end
end

bitnum = pos+neg+1;
binpoint = neg;
```

The following flowchart explains the code snippet in more detail.

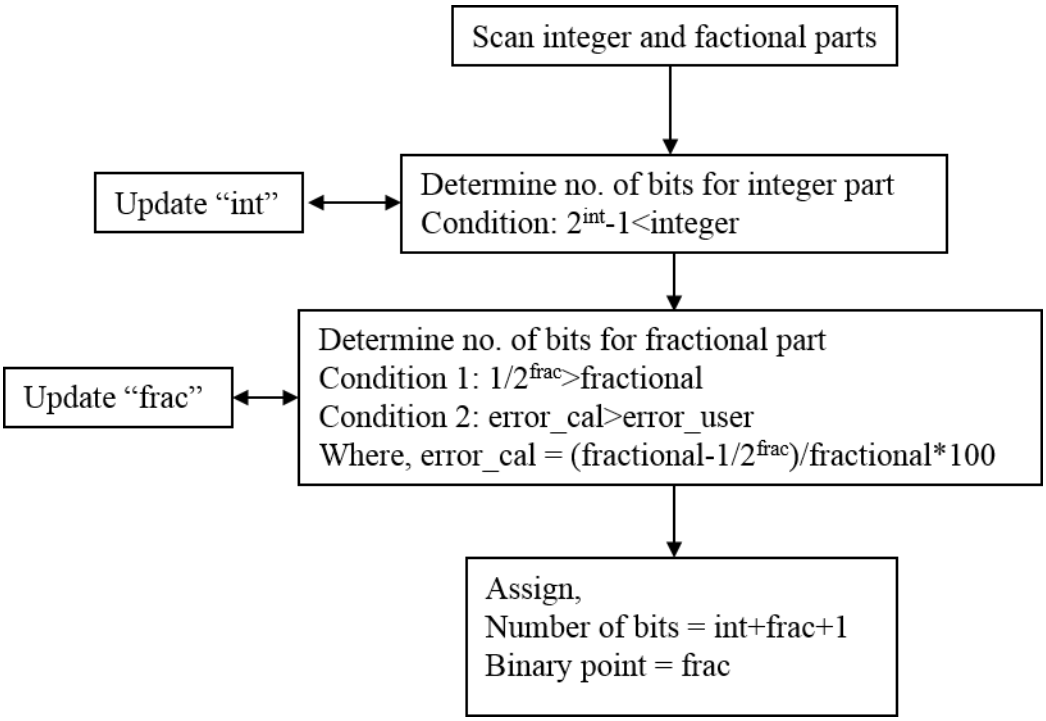


Fig. 2.9: Flowchart to calculate data width

2.6 Hardware-in-the-loop (HIL) co-simulation

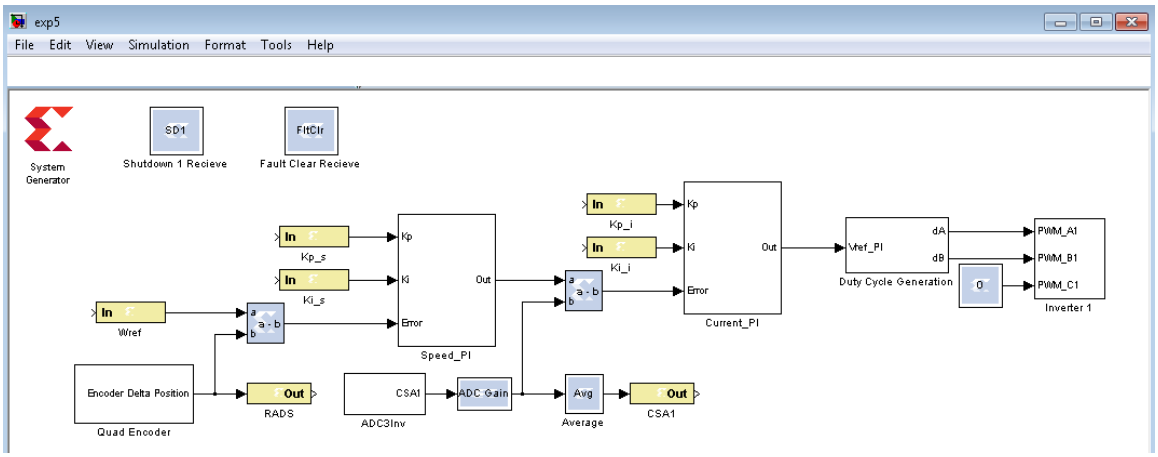
Simulations have become an integral part of the product development cycle, being used for testing the system behavior or performance. A typical simulation would consist of a set of inputs, a model of the system, control algorithms and a set of outputs, with the whole simulation code being executed on a computing platform. After the simulation, a prototype would be fabricated and tested. Any modifications to the design would require repeating of the whole cycle.

In a hardware-in-the-loop simulation, hardware components are included into the system's control loop and the simulation is executed in real time. Such a system is closer to the real product, and thus provides a better understanding of the system. Hardware-in-the-loop simulations have numerous benefits. The development cycle is reduced and the cost to innovate is lowered, as the traditional "simulate and prototype" loop is avoided by

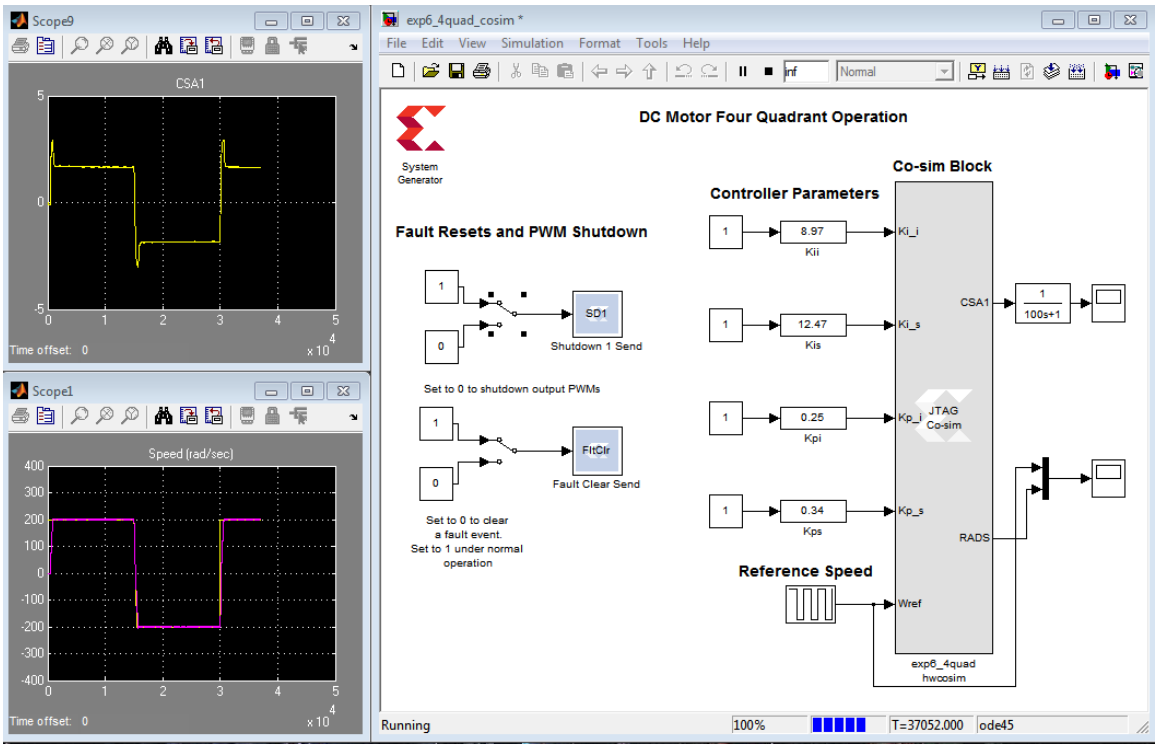
introducing “rapid prototyping”. Through this any modifications to the design can be quickly tested on hardware. This also improves the reliability and increases the efficiency of the testing system. It also helps in identifying design issues early in the development cycle.

This approach offers a lot of flexibility in the product development cycle, for example, the testing of control hardware need not wait for the availability of hardware prototype of the plant. It can proceed with a mathematical model of plant simulated with real control hardware. The approach also helps in avoiding potentially dangerous system level tests.

This thesis illustrates such a setup using System Generator for hardware-in-the-loop (HIL) co-simulations for power electronics and electric drives applications. Fig. 2.10(a) and Fig. 2.10(b) show an example of a HIL setup for the closed loop speed control of a DC motor. Fig. 10a shows the model of a cascaded PI loop with an outer speed control and an inner current control structure. The model is built from the library “Power and Motion Toolbox” which is introduced in section 2.3.2. The model is built and simulated within Simulink to understand the various stages of the cascaded structure. After a thorough analysis the model is then generated into a bitstream file that gets represented as a cosim blocks as shown in Fig. 2.10(b). The co-simulation block can then be run in real-time within Simulink which downloads the bitstream file into the FPGA’s random access memory and then can be used to pass data back and forth between the PC and the FPGA controller. In this example as can be seen from Fig. 10b the model is set up such that the controller parameters can be adjusted in real-time from the PC and the model implemented in the FPGA reflects any changes made to the parameters and thus changes the performance of the DC motor under operation. Similarly signals such as encoder speed feedback and current and voltage measurements from ADCs can be received in the PC for display to the user. This provides a powerful real-time graphical interaction between the user and the FPGA controller and can be used to visually understand various aspects of the control algorithm and make changes to it without turning off the drive.



(a)



(b)

Fig. 2.10 (a): Main Xilinx Model; (b) Co-simulation GUI environment

Chapter 3

Motor Control Implementation

3.1 Usage of Xilinx System Generator in the controller design

Matlab Simulink software package provides a powerful high level modeling environment for people who are involved in system modeling and simulations. Xilinx System Generator Tool developed for Matlab Simulink package is widely used for algorithm development and verification purposes in Digital Signal Processors (DSP) and Field Programmable Gate Arrays (FPGAs). System Generator Tool allows an abstraction level algorithm development while keeping the traditional Simulink blocksets, but at the same time automatically translating designs into hardware implementations that are faithful, synthesizable, and efficient.

Here in this chapter, a closed loop speed control of a DC motor and an open loop Volts/Hertz control of an induction machine driven by a Voltage Source Inverter (VSI) is analyzed by using a Matlab Simulink model. The control signals for the VSI in the related models are generated by the Xilinx FPGA chip. The Xilinx System Generator Tool provides such an interface, that is, a control algorithm developed by Xilinx System Generator Tool convenient to be used with traditional Simulink blocksets can be translated to the VHDL codes needed for the controller to be embedded in the FPGA chip.

The following section briefly introduces system modeling using the Xilinx System Generator Tool's Xilinx blockset along with the custom built Power and Motion Toolbox.

3.2 System modelling using the Xilinx System Generator

The formation of a FPGA design begins with a mathematical description of the operations needed for the controller as well as the communication protocols for ADCs, encoder and ends with the 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 efficient in terms of resource utilization and speed, while still producing acceptable results. As an example showing the issue of non-faithfulness of hardware realization as compared to the simulation, a simple case of multiplication of two constant numbers is compared between blocks of Simulink and that of Power and Motion Toolbox and is shown in Fig. 3.1.

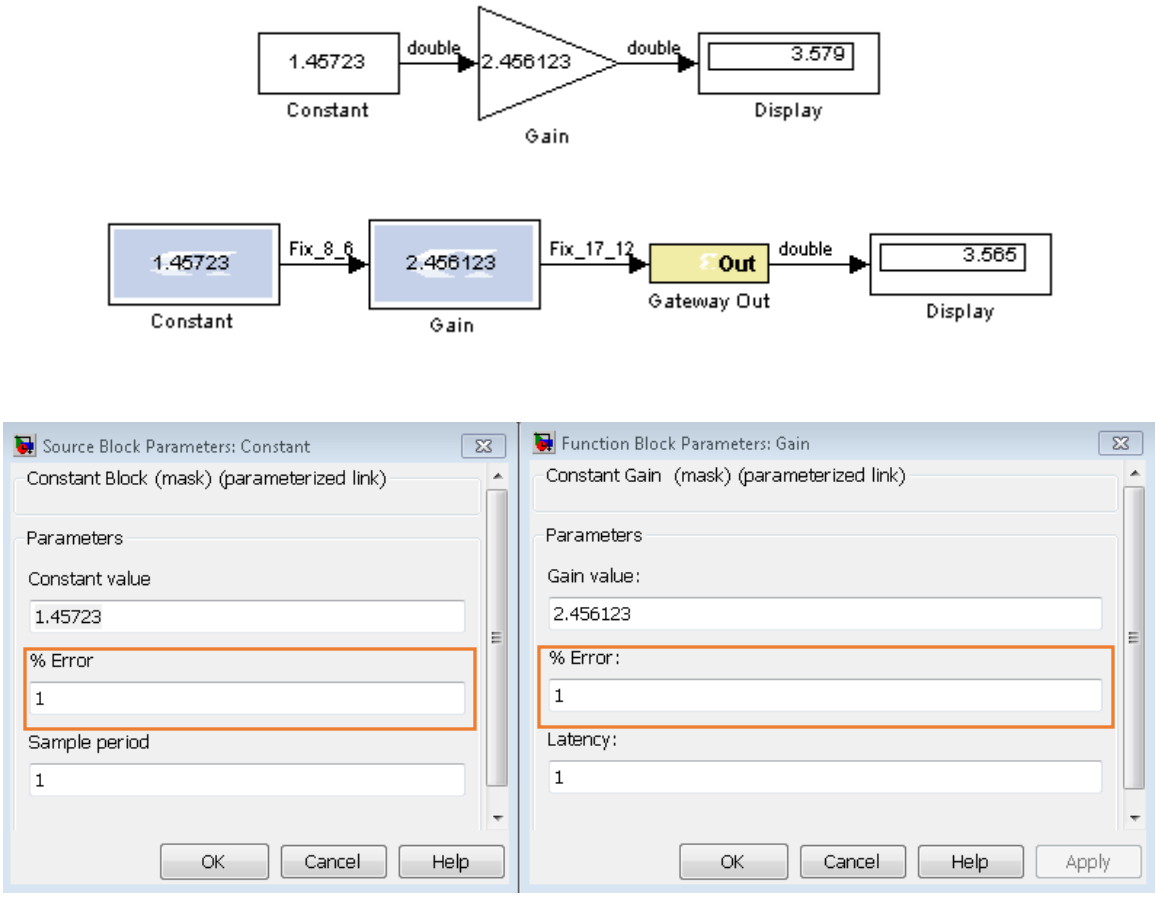


Fig. 3.1: Loss in precision of outputs while using Xilinx Blocks

As we can see there is an error in the arithmetic with the Xilinx blocks. In the Power and Motion Toolbox, the user is given an option of selecting the percentage of error of the outputs of each of the blocks, Constant and Gain. The error that the user selects directly affects the word length of the signals and hence costs more hardware resources to implement. This issue had been nearly non-existent in the Simulink simulation environment with floating point double precision numbers. Hence hardware realization needs to be designed keeping in mind this very critical limitation of fixed point processors.

In a typical design flow supported by System Generator, the following steps are followed:

1. Describe the algorithm in mathematical terms;
2. Realize the algorithm in the design environment, initially using double precision;
3. Trim double precision arithmetic down to fixed point;
4. Translate the design into efficient hardware.

In this study, Xilinx FPGA application board is taken as a basis for a real time application. When the control algorithm design of the controller is completed in Matlab Simulink environment by using Xilinx System Generator, it can be translated automatically into VHDL programming language and then can be embedded into the Xilinx FPGA application board.

Here, the Matlab Simulink environment forms the basis for the design of the controller utilized for the closed loop speed control of a DC motor. In an ideal simulation closed loop controller is operated on a mathematical model of the DC motor and the various outputs of the motor namely mechanical speed, current, torque etc. are fed back to the controller. The difference in the hardware based design is that it implements all stages of the control system except for the actual motor. The digital controller generates switching PWM signals to operate a VSI (or in the case of a DC motor, a full bridge converter) that

drives a motor. Physical quantities such as mechanical speed and motor current are fed back to the controller. The block diagram of complete system simulation model including the controller in Matlab Simulink developed for the closed loop DC motor control is shown in Fig. 3.2.

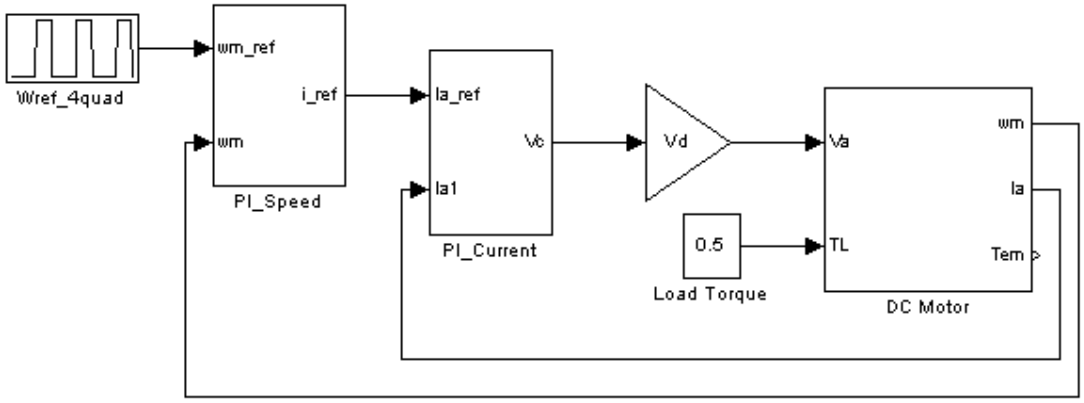


Fig. 3.2: Closed loop DC motor simulation

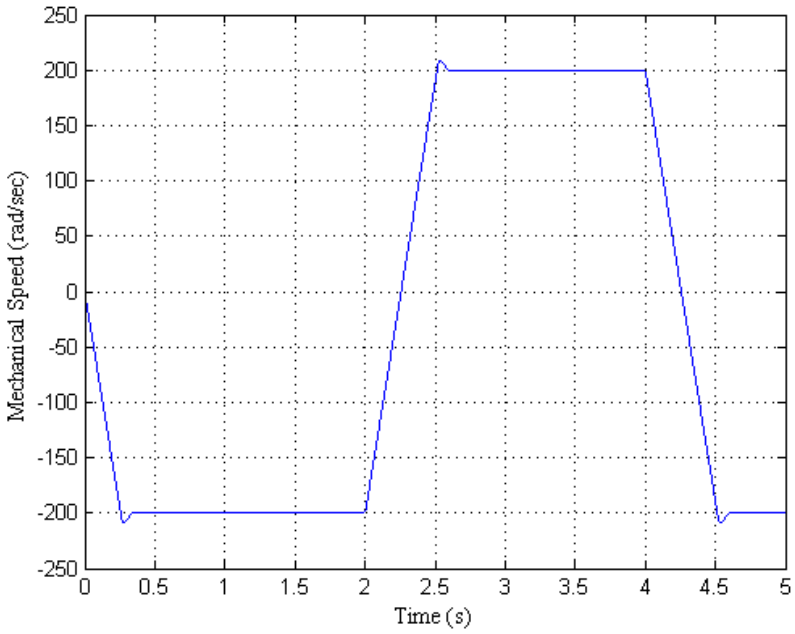


Fig. 3.3: Closed loop speed response

Required control algorithms within the FPGA are designed digitally with Xilinx blocksets whose general block diagram view is given in Fig. 3.4. After design the model

is generated into a bitstream file that can be programmed into the FPGA and a HIL co-simulation environment is set up to change various motor parameters during run time as shown in Fig. 3.5.

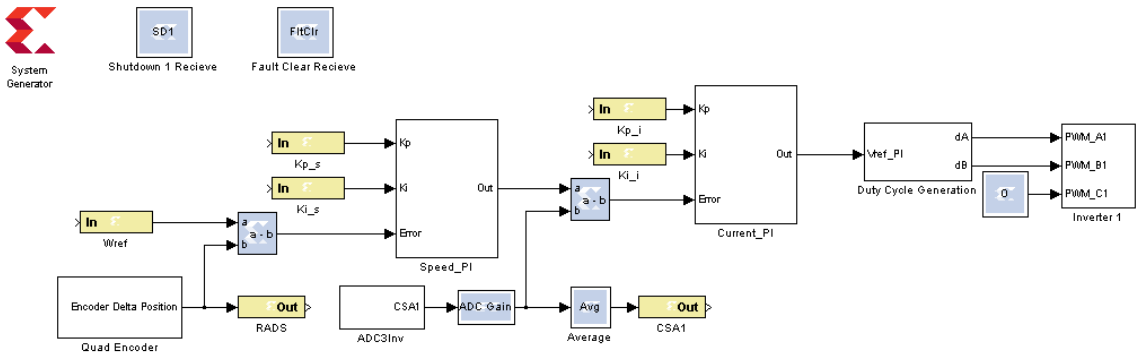


Fig. 3.4: Closed loop DC motor Xilinx model

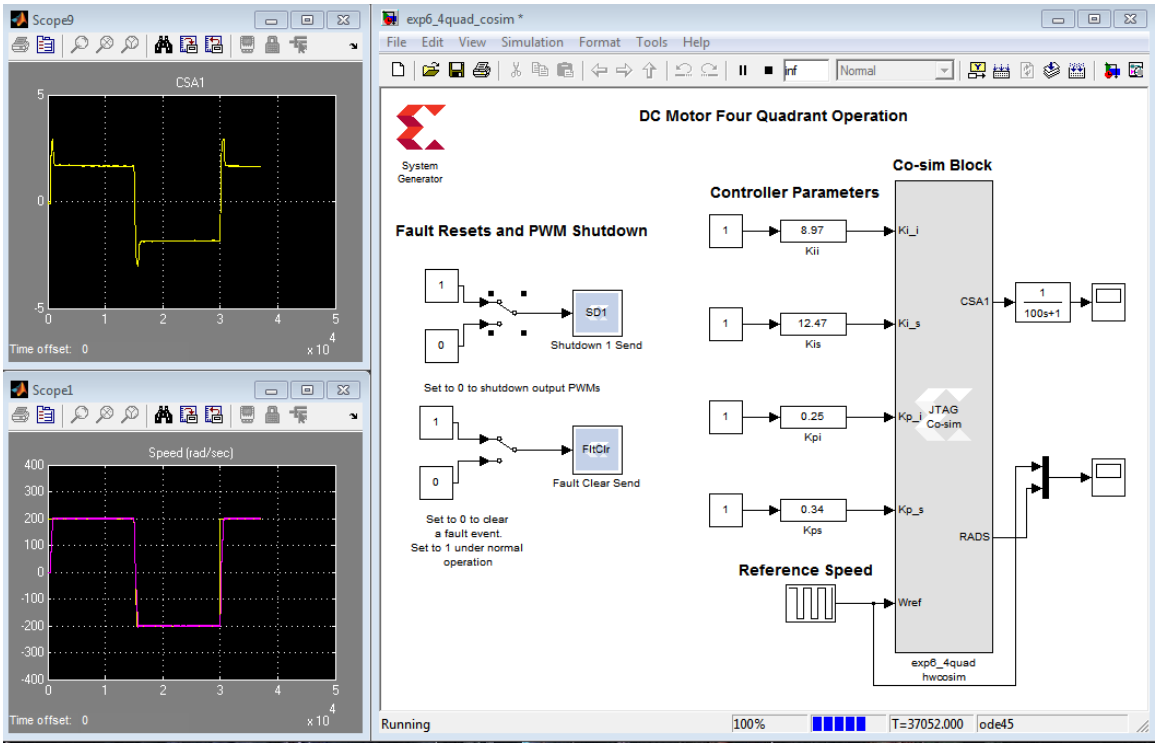


Fig. 3.5: Closed loop DC motor co-simulation GUI environment

Here, all the control blocks and related sub-blocks of the design developed using Xilinx Blocksets in Matlab Simulink Environment are shown and explained. The system model developed for each sub-block is explained further in the following subsections.

3.3 Design Stages for the FPGA Based Controller

In this study, the design stages for the required arithmetic and logical operations for the closed loop controller are carried out in a hierarchical and modular fashion. In this way, the construction, development and error checking steps are made easy. The general view of the complete controller design is given in Fig. 3.4, and the sub-blocks of the design can be described as follows:

- Speed Measurement
- Proportional Integral (PI) Controller
- Saturation Block
- Average Block
- Analog to Digital Converter
- PWM Modulator
- Inverter Switching Signals Generation Block

3.3.1 Speed Measurement

The encoder used in the study is a Timken M15 modular magnetic quadrature encoder with quadrature phase shifted outputs A and B to sense position as shown in Fig. 16. Using two code tracks with sectors positioned 90 degrees out of phase, the two output channels of the quadrature encoder indicate both position and direction of rotation. If A leads B, for example, the disk is rotating in a clockwise direction. If B leads A, then the disk is rotating in a counter-clockwise direction.

By monitoring both the number of pulses and the relative phase of signals A and B, you can track both the position and direction of rotation. The quadrature encoder also includes a third output channel, called a zero or index or reference signal (RP), which supplies a single pulse per revolution.

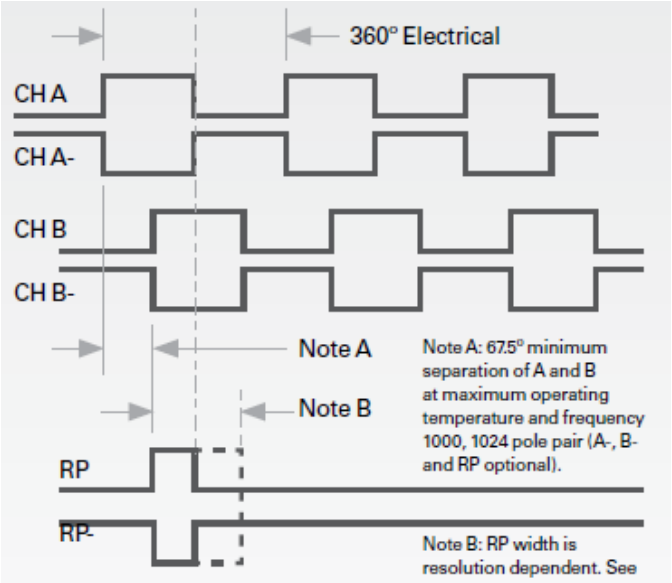


Fig. 3.6: A, B and Index Pulses of a quadrature encoder

The encoder channels “ENC_A”, “ENC_B” and “ENC_Z” are given as inputs to the speed measurement subsystem. The subsystem consists of various stages that perform the conversion of quadrature square pulses to the three quantities namely: Encoder Delta Position (rad/s), Encoder Position and Index.

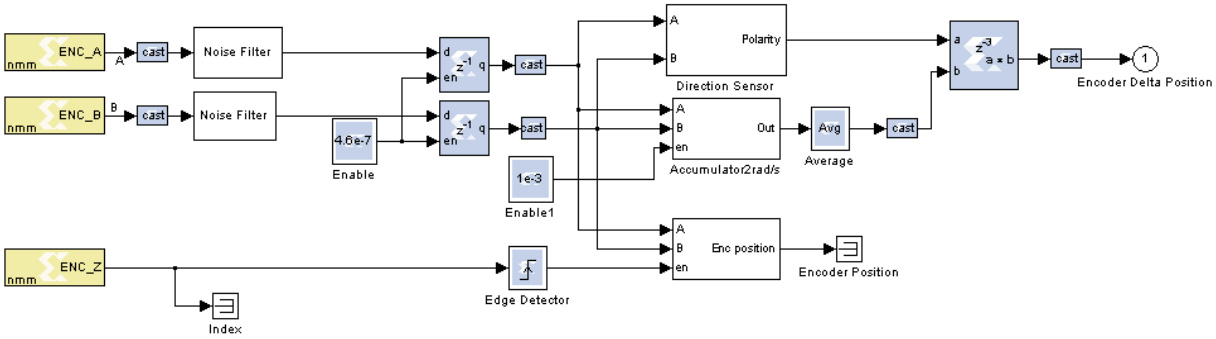
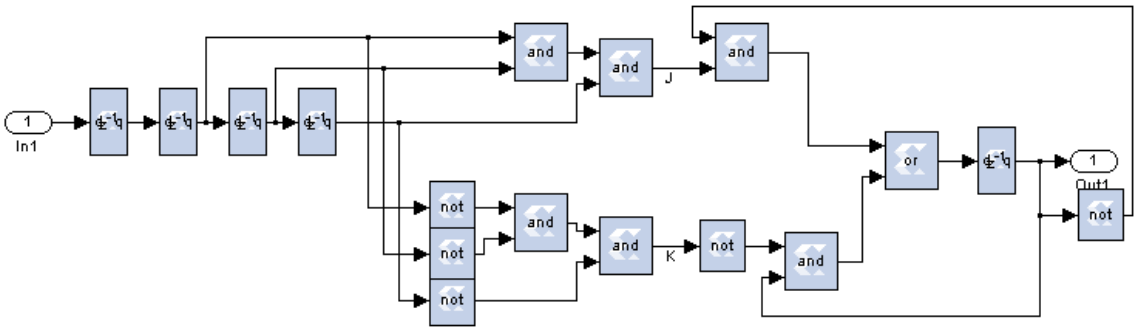
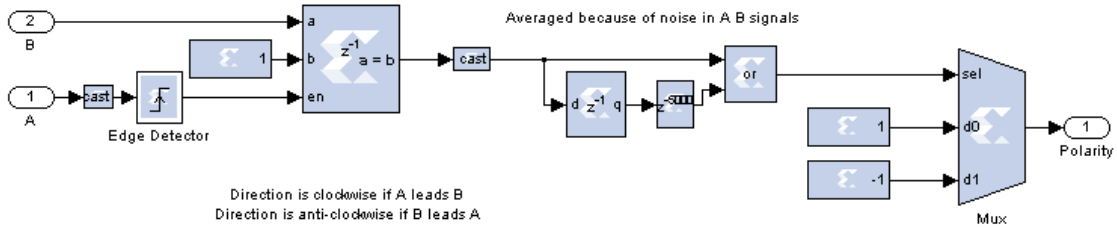


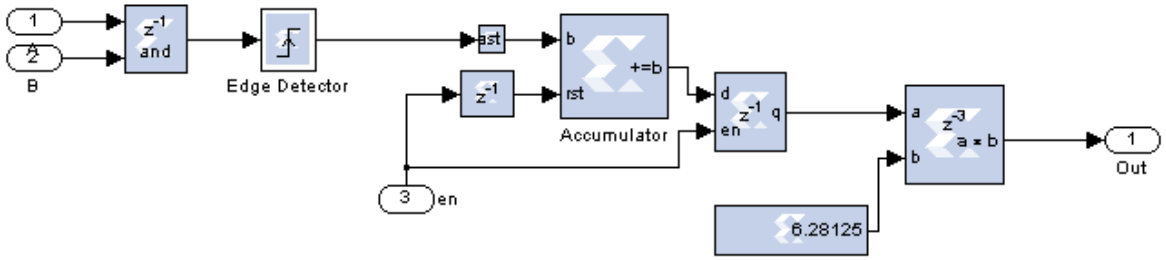
Fig. 3.7: Quadrature encoder overall diagram



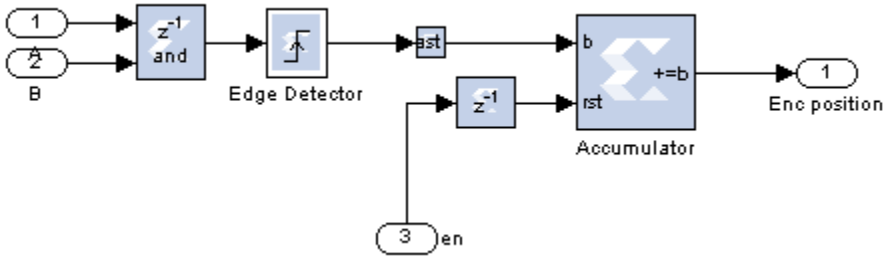
(a)



(b)



(c)



(d)

Fig. 3.8 (a): Digital noise filter; (b) Direction Sensor subsystem; (c) Accumulator2rad/s subsystem; (d) Enc position subsystem

3.3.2 Proportional and Integral Controller

Proportional gain value, K_p is selected for the design of the proportional block; that is, speed error is multiplied by K_p and the result is applied as the output of the proportional stage.

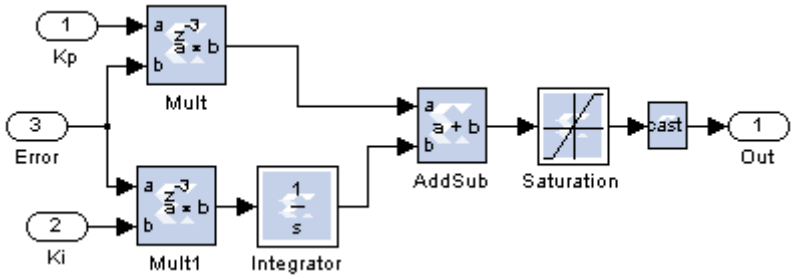


Fig. 3.9: PI controller subsystem

Integral gain value K_i is selected and a time step T_s is selected for the design of the integral block. Thus speed error is multiplied by K_i and then integrated. The details of the integrator is shown in Fig. 3.10.

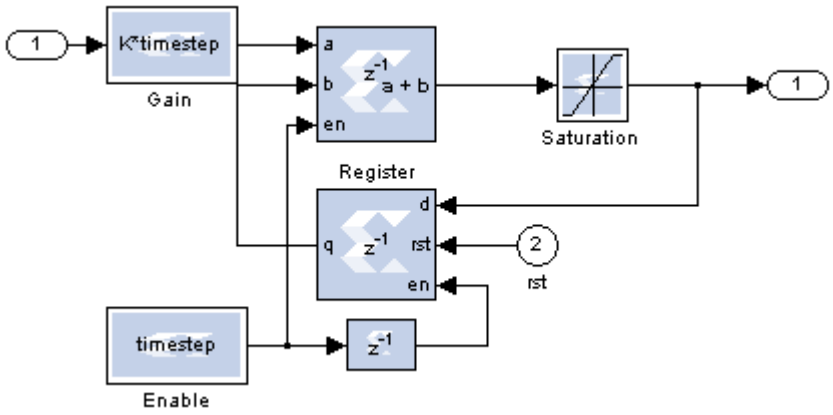


Fig. 3.10: Discrete integrator subsystem

The integrator is implement as a discrete time accumulator according to the Forward Euler equation given by:

```
y(n) = y(n-1) + K*Ts*u(n-1)
```

3.3.3 Saturation Block

The saturation block is used to limit a signal to upper and lower threshold values that are provided by the user. Table 1 summarizes the output states of the Saturation block based on the user settings of upper and lower thresholds.

Table 1: Saturation block output levels

"hi"	"lo"	Multiplexer Output	Comment
0	0	d0	No need to limit, transfer input as it is to Mux output
0	1	d1	Need to limit output at lower threshold, lth
1	0	d2	Need to limit output at upper threshold, uth
1	1	X	No state

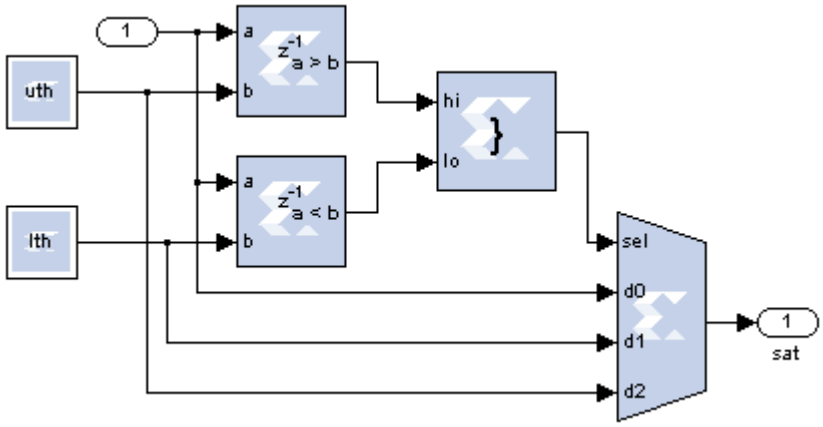


Fig. 3.11: Saturation subsytem

Let us consider its usage in the PI controller block. Output of the PI control block is applied as input to the Saturation block shown in Fig 3.9. This block limits the value coming from the PI output between upper threshold and lower threshold. The detailed design of the torque limiter block is shown in Fig. 3.11.

3.3.4 Average Block

This block implements a moving window average with a window span of 10 elements. The general expression for moving average is given by:

$$Avg = [u_m + u_{m-1} + \ldots + u_{m-(n-1)}]/n$$

As can be seen from Fig. 22, the incoming data values are stored in registers that are connected in a cascaded fashion. All the registers are enabled at a time rate specified by the user. The values in the registers are added sequentially and the result is divided by 10 to get the average over 10 samples.

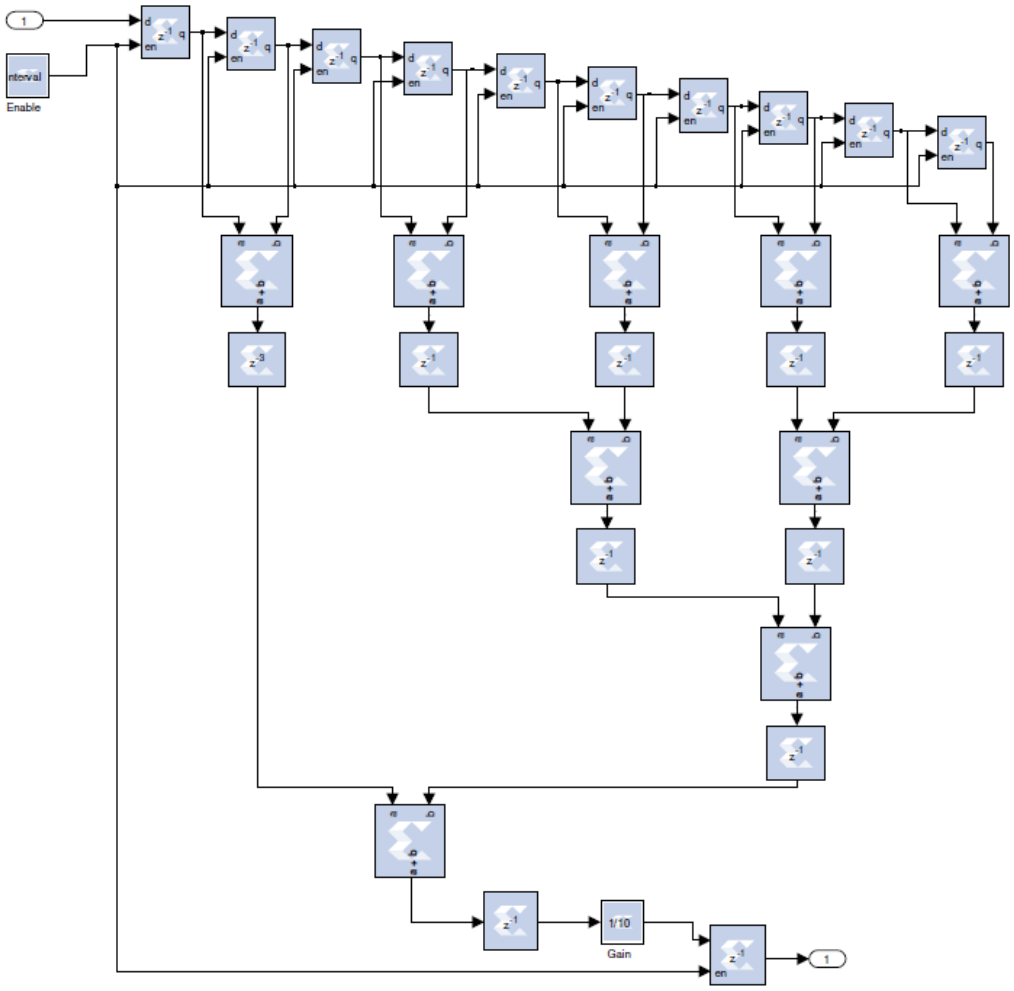


Fig. 3.12: Average subsystem

3.3.5 Analog to Digital Converter

The analog to digital converter is responsible for converting the measured analog current and voltages into a digital format that is suitable for use inside the FPGA processor. The ADC used in the application is an Analog Devices IC AD7366. It is a dual 12 bit true bipolar ADC with a throughput rate of 1 MSPS and analog range of +/-10V. More details on the ADC and its design is provided in Chapter 4. From the datasheet we have the timing diagrams of the ADC as shown in Fig. 3.13 and Fig. 3.14.

A conversion start signal, CNVST_B is pulled low for time "t_1" after which the ADC starts conversion for a period of "$t_{convert}$". After the time has passed the Chip Select line, CS_B is pulled low for a period during which the 12 data bits are sent out of the ADC in a serial fashion and is synchronized with the serial clock SCLK. This is shown in Fig. 3.14.

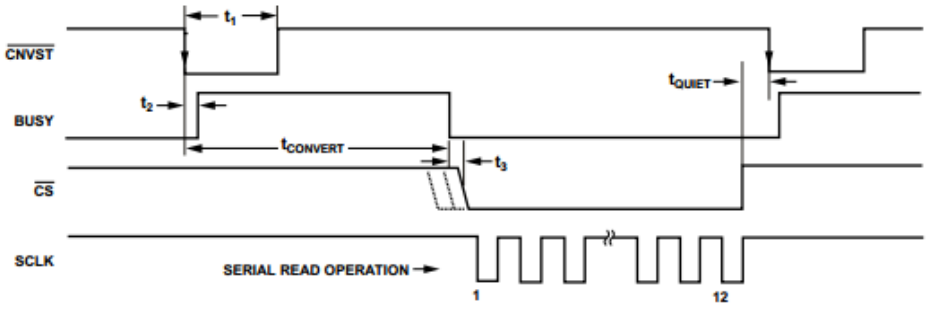


Fig. 3.13: Normal mode of operation of AD7366

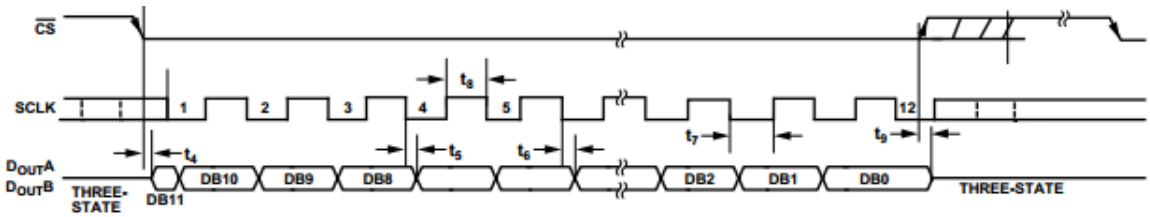


Fig. 3.14: Serial Interface Timing Diagram for AD7366

The implementation of the serial communication is shown in Fig. 3.15, Fig. 3.16 and Fig. 3.17. For further details refer the datasheet for AD7366.

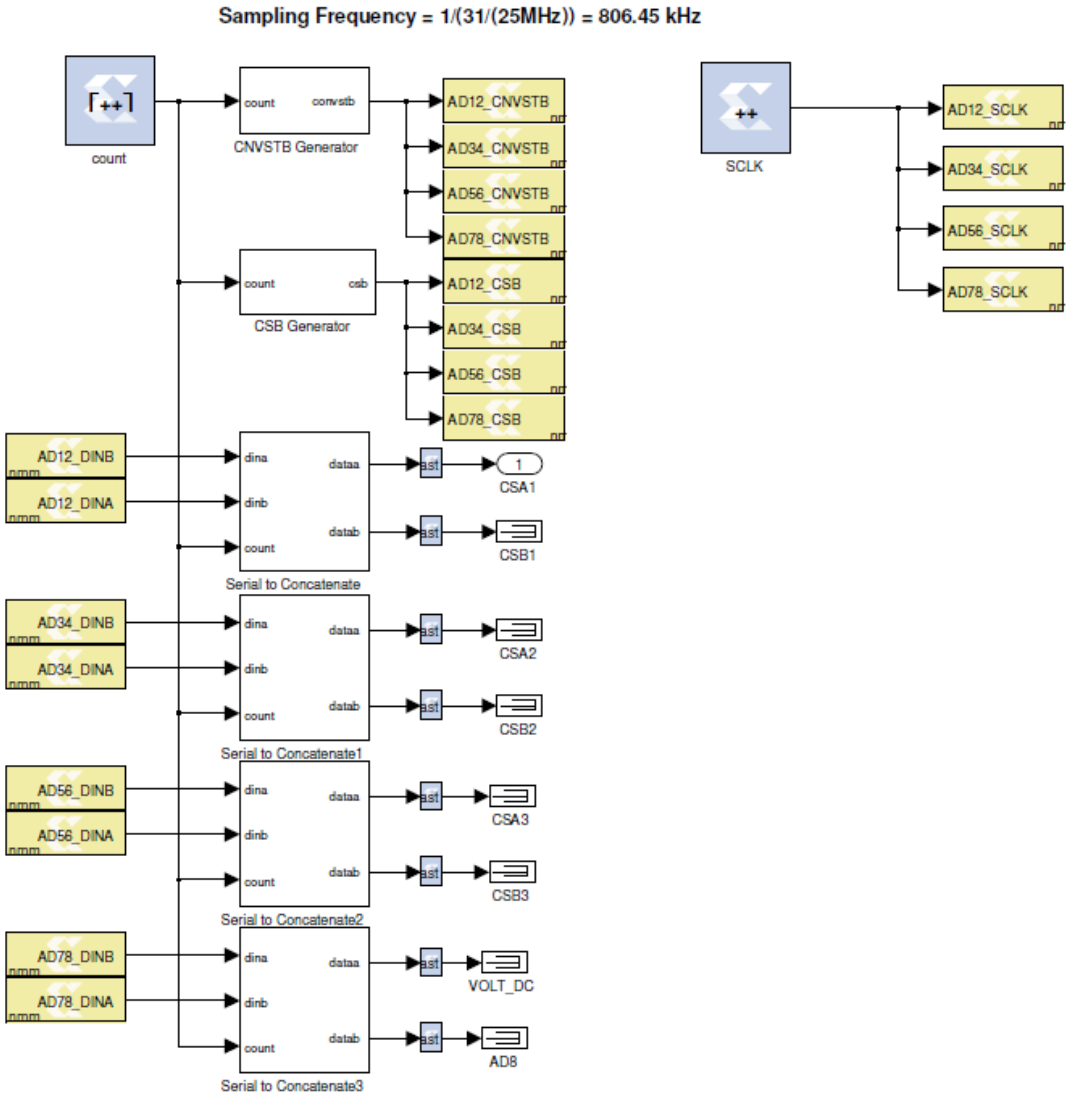


Fig. 3.15: ADC overall diagram

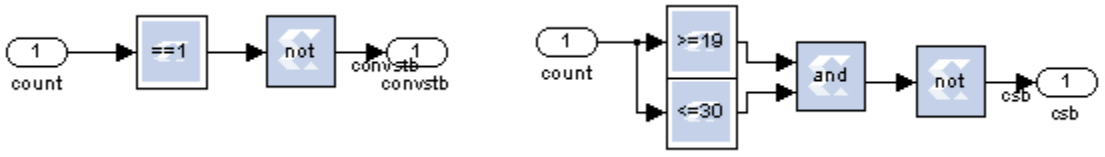


Fig. 3.16: ADC internal CNVSTB Generator and CSB Generator subsytems

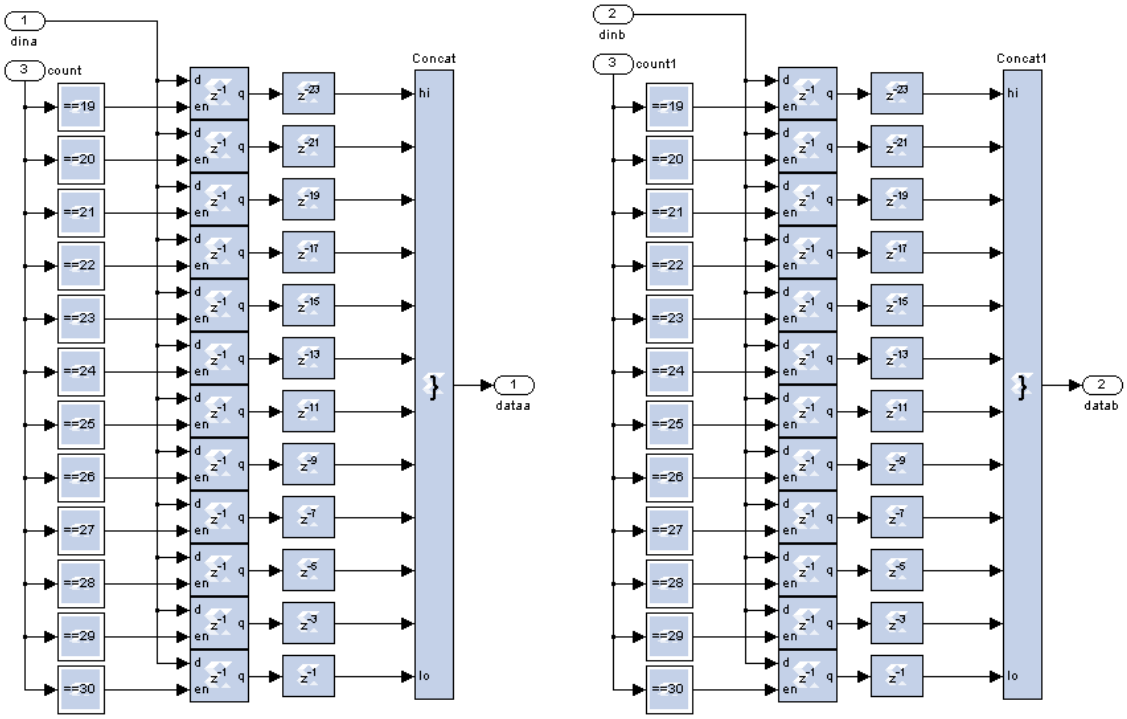


Fig. 3.17: ADC internal Serial to Concatenate subsytem

3.3.6 PWM Modulator

In the cascaded control system model under study, the inner current controller generates a reference voltage for the machine to operate. Since in the real hardware system there is a full-bridge 4 quadrant converter that is connected to the DC motor which needs to be modulated. Hence two duty signals dA and dB are generated as shown in Fig. 3.18 as per the equations:

$$dA = \tfrac{1}{2} + Vo/(2*Vd)$$

$$dB = \tfrac{1}{2} - Vo/(2*Vd)$$

These signals are average time varying quantities which need to be further converted to switched PWM signals. That is performed in the next stage.

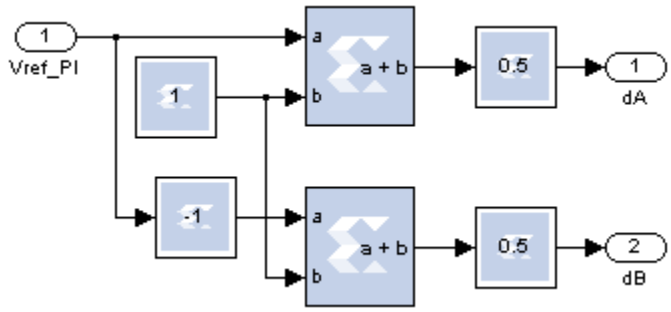
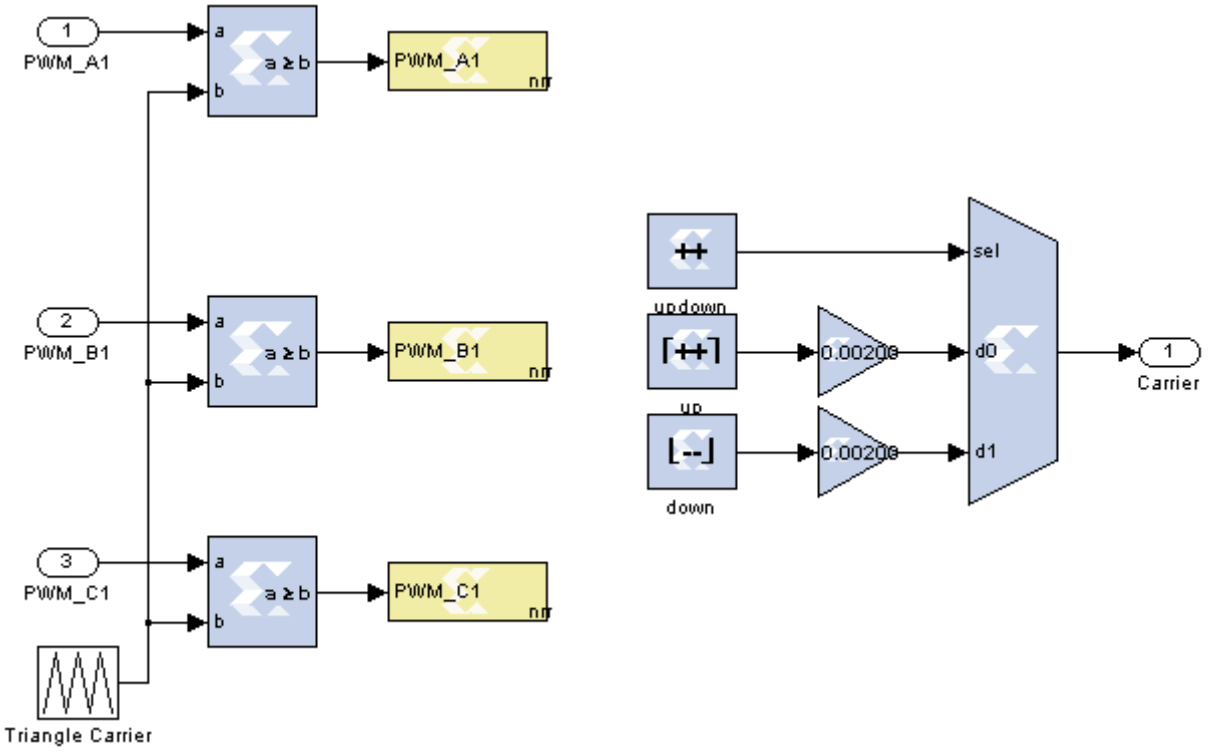


Fig. 3.18: DC motor PWM modulator

3.3.7 Inverter Switching Signals Generation Block

The inverter PWM signals are generated by comparing the average duty signals with a high frequency carrier to generate ON/OFF pulses. The carrier used here is a 10 KHz triangular signal and its implementation is shown in Fig. 3.19(b).



(a) (b)

Fig. 3.19: (a) PWM Generation; (b) Triangle generation

3.4 Additional Design Elements

Using the Xilinx blockset and the Power and Motion Toolbox various other subsystems have been created which are useful in building electric drive models. Some of these subsystems are briefly described below:

- Power Factor Calculator
- Slope Calculator
- DQ-ABC and ABC-DQ
- Space Vector Modulator

3.4.1 Power Factor Calculator

This block can be used to calculate the time difference between two signals (voltage and current) that are either lagging or leading and hence determine the power factor. The functioning of the calculator is listed below:

Step 1: Enable a counter if Voltage>0.
Step 2: Store the counter result in a register at the positive edge of the result Current>0. Also reset the counter when this relation is satisfied.
Step 3: If the result is positive and greater than previous value, send as output. Else repeat the process.

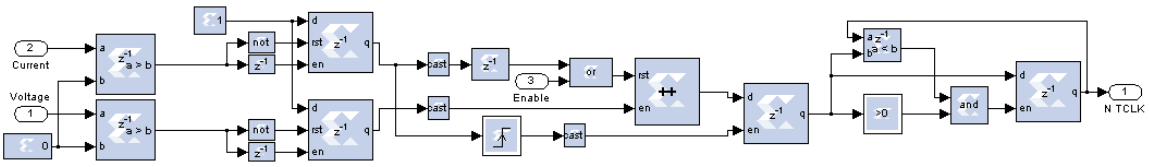


Fig. 3.20: Lagging Power Factor Calculator block

3.4.2 Slope Calculator

This block can be used to calculate the time between two values x1 and x2 in a signal and hence compute the slope. The functioning of the block is described below:

Step 1: Start counter when the condition Signal>x1 becomes true.
Step 2: Store the result of the counter at the positive edge of the relation Signal>x2. Also reset the counter after a unit delay.

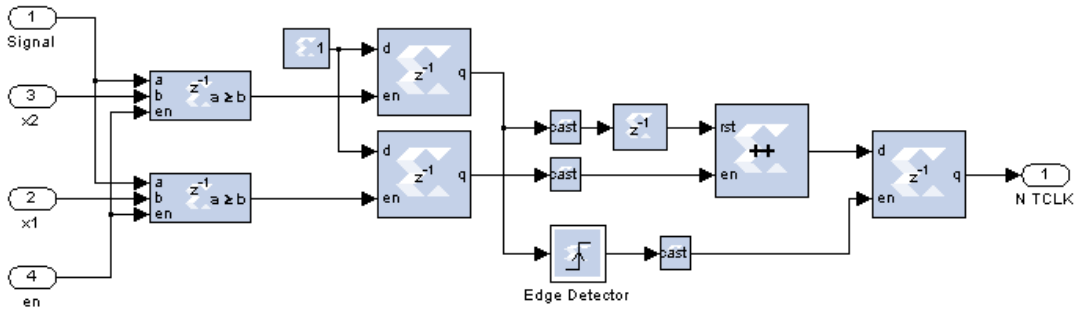
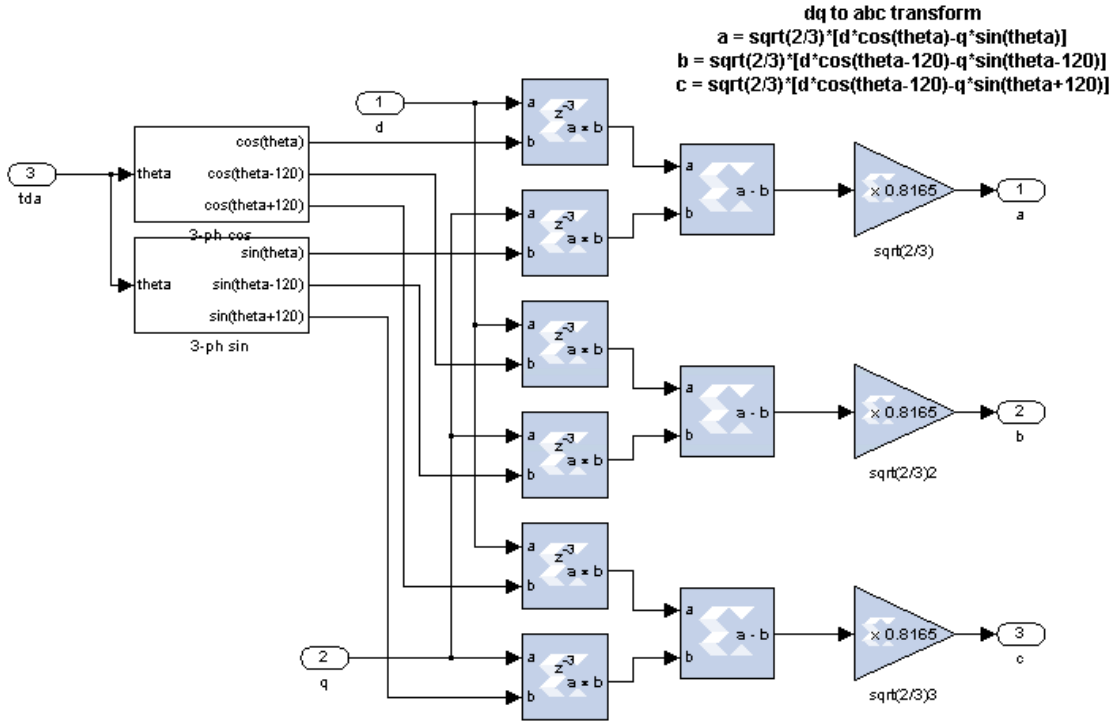


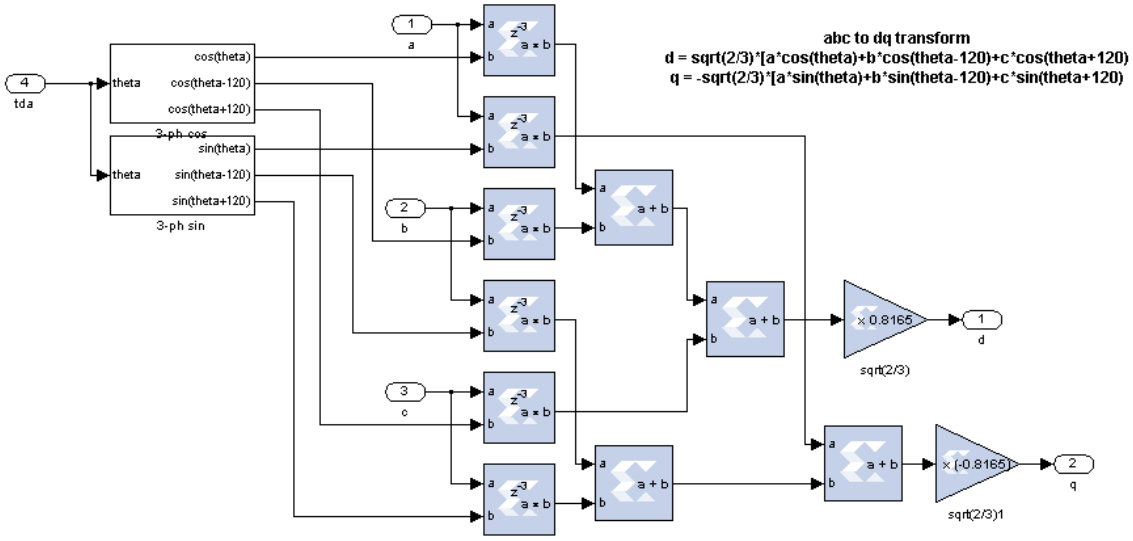
Fig. 3.21: Up Slope Calculator block

3.4.3 DQ-ABC and ABC-DQ Blocks

These blocks implement a rotational transformation of a 3 phase input and vice versa. The transformation is carrier out as per the Clark and Park transformation matrices. The implementation is shown in Fig. 3.22



(a)



(b)

Fig. 3.22 (a): DQ-ABC block; (b): ABC-DQ block

3.4.4 Space Vector PWM Modulator

The Space Vector modulation offers a superior performace over sine PWM in terms of the converter gain as well as reduced switching losses and lower THD. A flowchart depicting the SVPWM scheme is shown in Fig. 3.23. The SVPWM scheme has been implemented using System Generator and the various subsystems is shown in Fig. 3.24

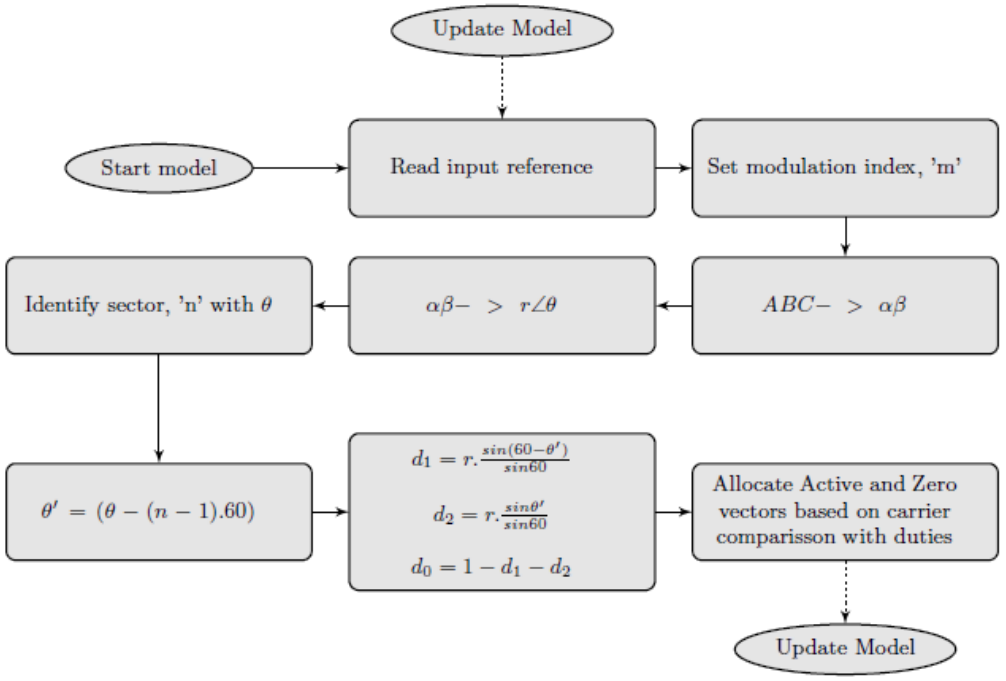
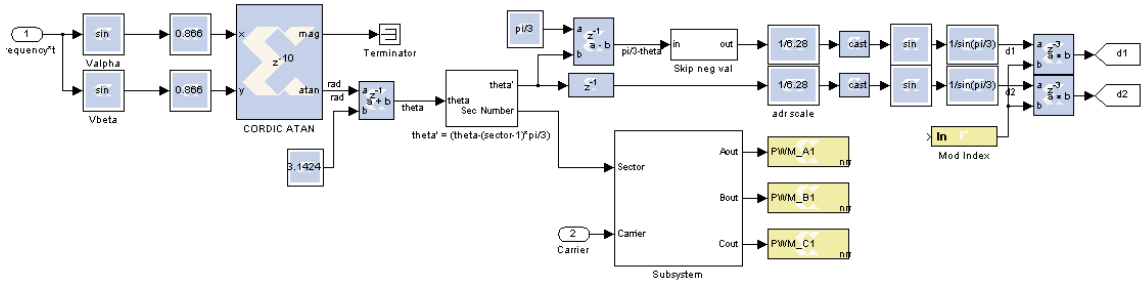
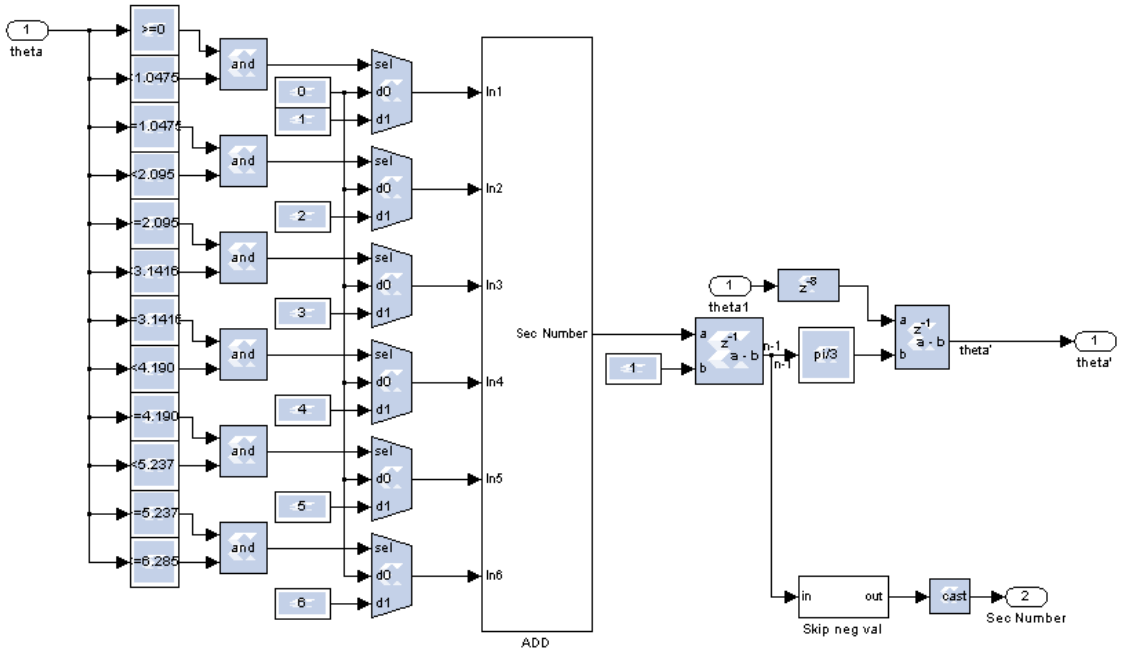


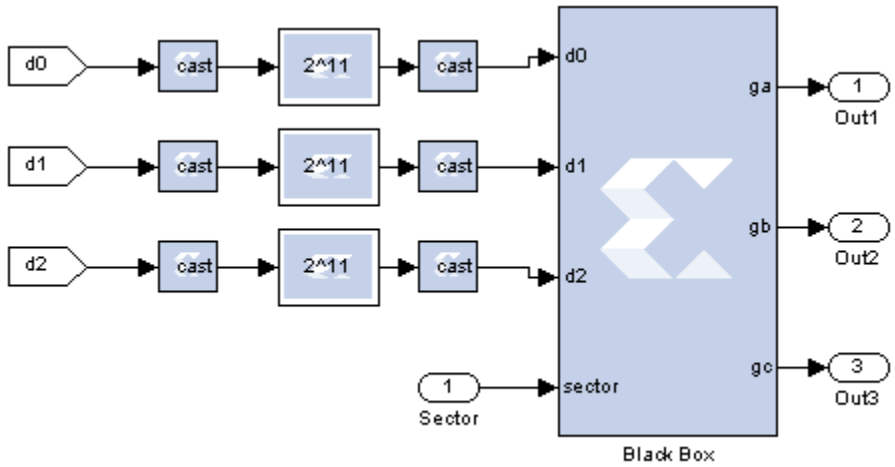
Fig. 3.23: SVPWM Flowchart



(a)



(b)



(c)

Fig. 3.24 (a): SVPWM overall diagram; (b): theta' = (theta-(sector-1)*pi/3) subsystem; (c) PWM generation subsystem

3.4.4.1 Simulation Results of SVPWM:

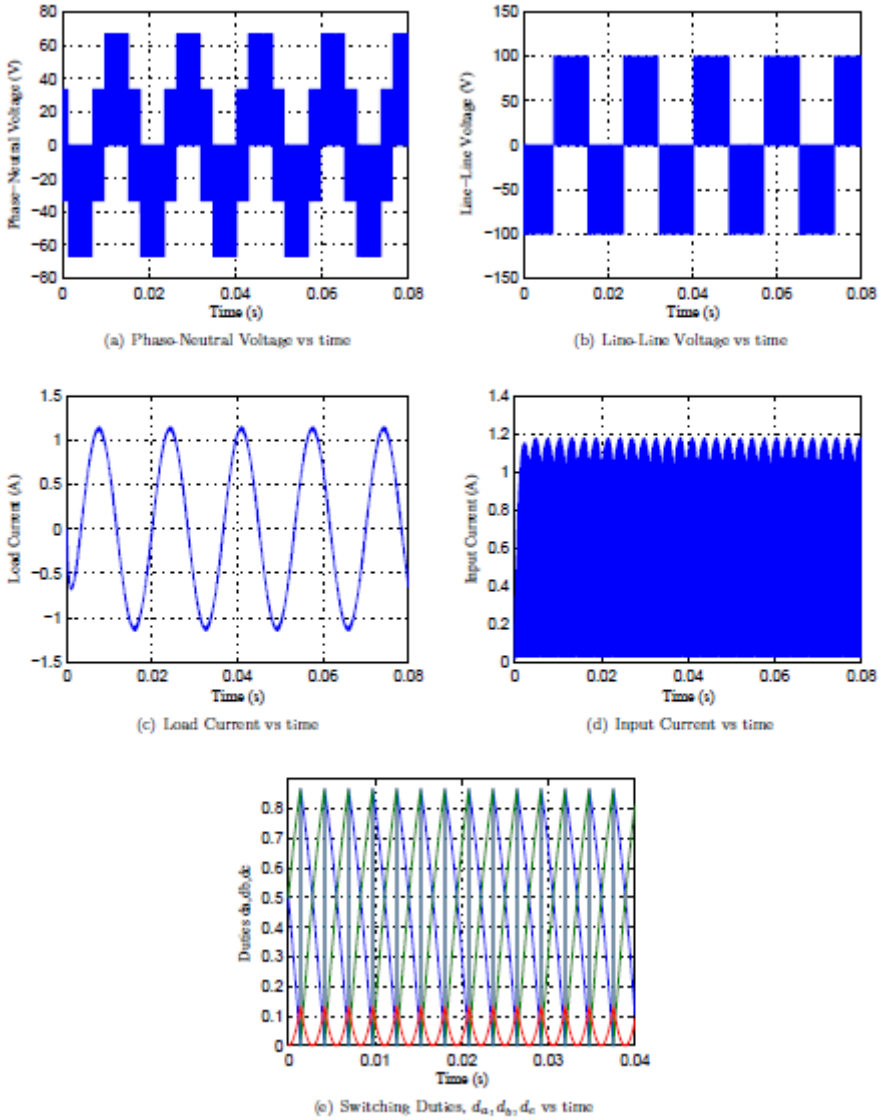
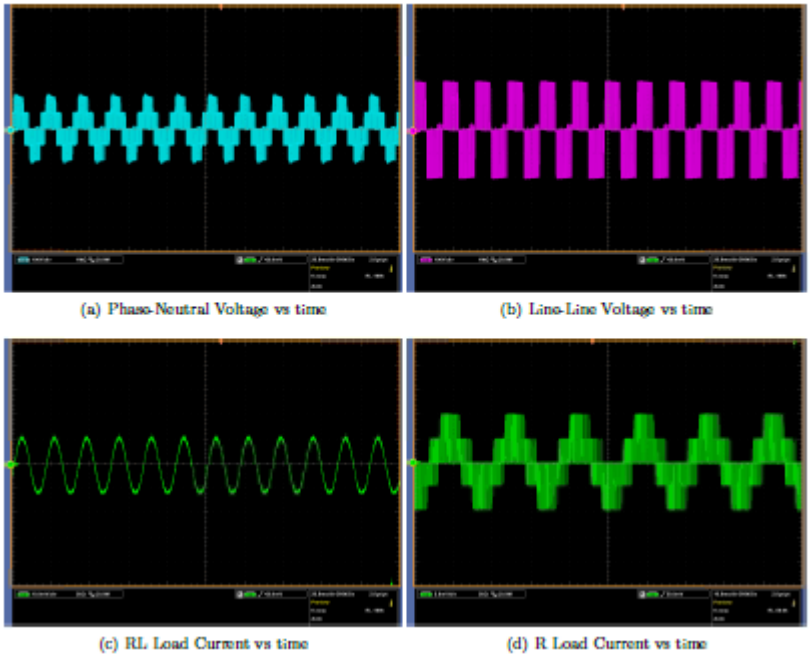


Fig. 3.25: SVPWM simulation results

3.4.4.2 Hardware Results of SVPWM:



(a) Phase-Neutral Voltage vs time

(b) Line-Line Voltage vs time

(c) RL Load Current vs time

(d) R Load Current vs time

Fig. 3.26: SVPWM hardware results

Chapter 4

Hardware Design

4.1 Overview

The desired goal is to design a system capable of performing hardware-in-the-loop simulations in power electronics and electric drive applications. The system should be capable of sensing various voltages, currents and other signals. It should be able to process the inputs based on a defined control algorithm and produce a set of outputs, which will be used to drive the gates in the power circuit.

The signals from the motor's encoder and the current and voltage sensors on the power board are sent to the hardware-in-the-loop controller, which has input buffers as well as analog-to-digital converters to read the signal. The voltages read are converted into digital words that can be used within the control algorithm. Various operational as well as controller parameters are read from the PC in real-time through a high speed JTAG link. The model produces outputs, which are duty ratios for switching the gates on the power board. A block diagram representation of the developed system is shown in Fig. 4.1.

This enables the user to quickly test their control algorithms on machines in real time. This approach utilizes the processing power of current FPGA processors and the high speed communication link offered by JTAG. This setup can be used as a tool for teaching motion control systems and also will be helpful in various stages of design and development of a motor controller.

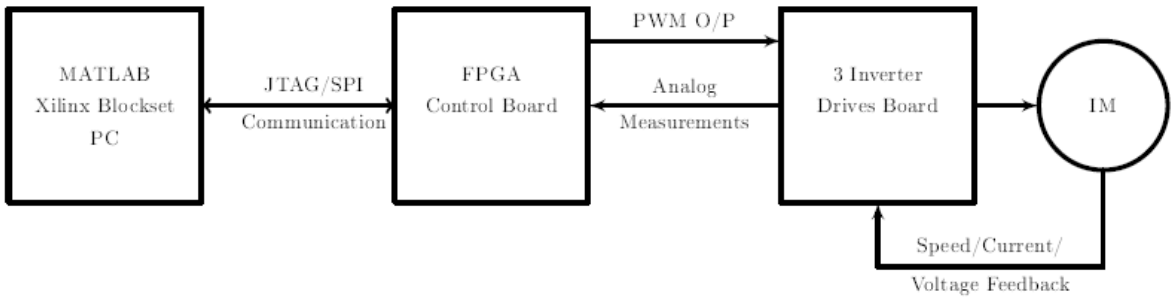


Fig. 4.1: Overall system block diagram

4.2 Design of the Hardware-in-the-loop controller

An FPGA based hardware-in-the-loop controller has been designed and developed as part of this thesis. Fig. 4.1 illustrates the outline of the FPGA controller board.

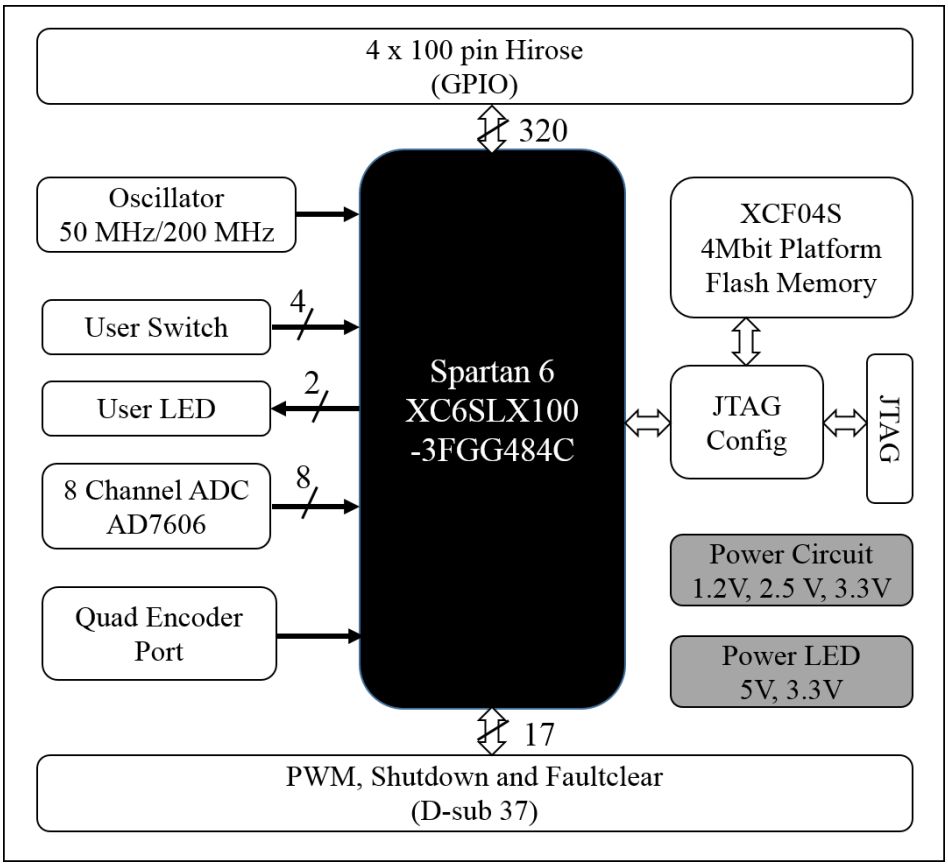


Fig. 4.1: FPGA controller card outline

The board was built around a Spartan 6 LX100 FPGA from Xilinx. The peripherals used for this design were an Analog Devices analog-to-digital converter IC AD7606, input buffer to receive a quadrature encoder's A, B and Z signals through a D-sub 15 connector and the output buffers connecting to a D-sub 37 connector to send out PWM and Shutdown signals. Communication between the FPGA and the computer via JTAG is taken care of by an onboard JTAG configuration circuit that also has a 4Mbit platform flash memory, XCF04S. There is also additional circuitry for selecting various modes in the ADC such as serial or parallel data select and input range select. The PWM signals are level-shifted to match the voltage levels of the driver circuit on the power board. The board needs to be powered from an external 5 volt supply. The power supply circuits are designed to supply power at various voltage levels required by the onboard components.

4.3 Component Selection

The main functions of the controller board are reading analog signals, encoder signals, generating PWM signals and communicating with the computer. Hence, the components were selected based those criteria. The analog to digital conversion capabilities were designed for a 16 bit conversion with a conversion speed of 200 kilo samples per second as determined by the maximum bandwidth of the control loops in this particular application (10 KHz). A minimum of eight ADC channels were required and a minimum of 9 PWM outputs were required. Beyond the requirements of the electric drives lab, General Purpose Input Output (GPIO) ports were also desired.

Another important design parameter was the processor operation frequency. Typical clock frequencies in the current state of the art are in the range of 10 MHz-200 MHz. Processors that operate in this range were considered. A high speed communication was required to communicate with the PC for hardware-in-the loop co-simulation. The JTAG communication was chosen as it has various operating frequencies, the common values being 30 MHz, 10 MHz, 7.5 MHz and 6 MHz (as selected by the user).

Xilinx's Spartan 6 LX100 FPGA had specifications that exceeded the I/O and speed requirements. Since the FPGA does not have any build in peripherals such as ADCs/encoder/PWM, external circuits had to be designed that integrated with the FPGA's I/O lines. Also, ready availability of development resources for Xilinx's FPGAs namely System Generator influenced the selection.

4.4 Features of Spartan 6 XC6SLX100 FPGA

Spartan 6 LX FPGAs are optimized for applications that require the absolute lowest cost. They support up to 147K logic cell density, 4.8Mb memory, integrated memory controllers, DSP slices, ease-of-use, and high performance Hard IP with an innovative open standards based configuration. Fig. 4.2 shows the Spartan 6 FPGA feature summary by device as provided in the Xilinx support documentation. The XC6SLX100 is one of the higher end products in the Spartan 6 family with sufficient number of Logic Cells, CLBs, user I/Os and DSP48A1 slices which provide sufficient arithmetic engines. Each DSP48A1 slice contains an 18 x 18 multiplier, an adder, and an accumulator.

Device	Logic Cells[1]	Configurable Logic Blocks (CLBs)			DSP48A1 Slices[3]	Block RAM Blocks		CMTs[5]	Memory Controller Blocks (Max)[6]	Endpoint Blocks for PCI Express	Maximum GTP Transceivers	Total I/O Banks	Max User I/O
		Slices[2]	Flip-Flops	Max Distributed RAM (Kb)		18 Kb[4]	Max (Kb)						
XC6SLX4	3,840	600	4,800	75	8	12	216	2	0	0	0	4	132
XC6SLX9	9,152	1,430	11,440	90	16	32	576	2	2	0	0	4	200
XC6SLX16	14,579	2,278	18,224	136	32	32	576	2	2	0	0	4	232
XC6SLX25	24,051	3,758	30,064	229	38	52	936	2	2	0	0	4	266
XC6SLX45	43,661	6,822	54,576	401	58	116	2,088	4	2	0	0	4	358
XC6SLX75	74,637	11,662	93,296	692	132	172	3,096	6	4	0	0	6	408
XC6SLX100	101,261	15,822	126,576	976	180	268	4,824	6	4	0	0	6	480
XC6SLX150	147,443	23,038	184,304	1,355	180	268	4,824	6	4	0	0	6	576
XC6SLX25T	24,051	3,758	30,064	229	38	52	936	2	2	1	2	4	250
XC6SLX45T	43,661	6,822	54,576	401	58	116	2,088	4	2	1	4	4	296
XC6SLX75T	74,637	11,662	93,296	692	132	172	3,096	6	4	1	8	6	348
XC6SLX100T	101,261	15,822	126,576	976	180	268	4,824	6	4	1	8	6	498
XC6SLX150T	147,443	23,038	184,304	1,355	180	268	4,824	6	4	1	8	6	540

Fig. 4.2: Spartan 6 FPGA Feature Summary by Device

4.5 Features of the Analog-to-digital converter AD7606

The AD7606 is a 16-bit bipolar 8 channel ADC with simultaneous input sampling. The functional block diagram of the IC is shown in Fig. 4.3. The ADC is specifically designed for motor control applications among others and has the following features:

- True bipolar analog input ranges: +/-10V, +/-5V
- Single 5V power supply
- Input buffer with 1M ohm analog input impedance
- Second order antialiasing analog filter
- On-chip accurate reference for SAR
- 16-bits with throughput rate of 200kSPS on all channels
- Digital output filter with oversampling capability
- Flexible parallel/serial interface

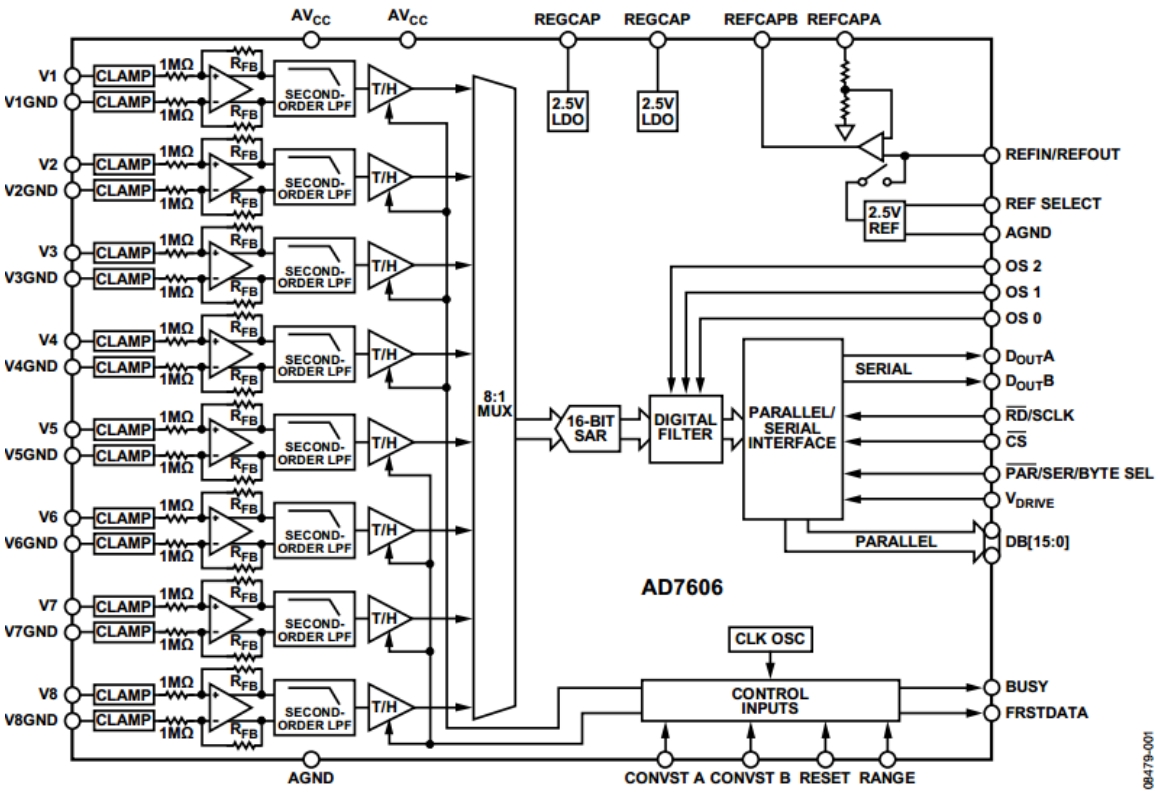


Fig. 4.3: AD7606 functional block diagram

4.6 Design of various circuits

In this section the design details of various circuits in the controller board is discussed.

4.6.1 Power Supplies and Decoupling Capacitors

The Spartan 6 XC6SLX100 FPGA has three operating voltages: 1.2V, 2.5V and 3.3V. Hence voltage regulators for these different levels has been designed as shown in Fig. 4.4. The main supply to the board is from an external 5V source. The FPGA needs I/O bank supplies, internal core supplies and auxillary circuit supplies. These values are designed from the IC datasheeet and decoupling capacitors are arranged for each bank in an increasing order of capacitance. They are physically placed as close to the chip as possible to help provide stable voltage levels. In this particular board, the capacitors are generally placed in the bottom layer and connected to the FPGA pins through a via. The power is supplied to the capacitors directly from the internal power planes. Care has been taken to ensure that the capacitors appear in the power chain before the FPGA pins.

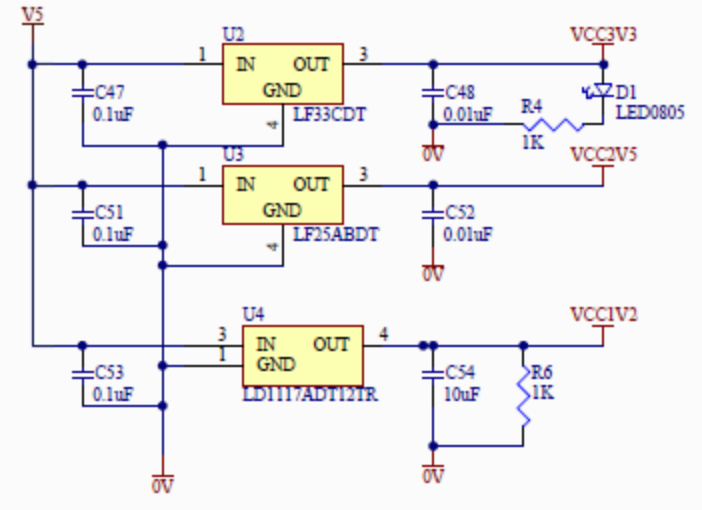


Fig. 4.4: Voltage regulator design

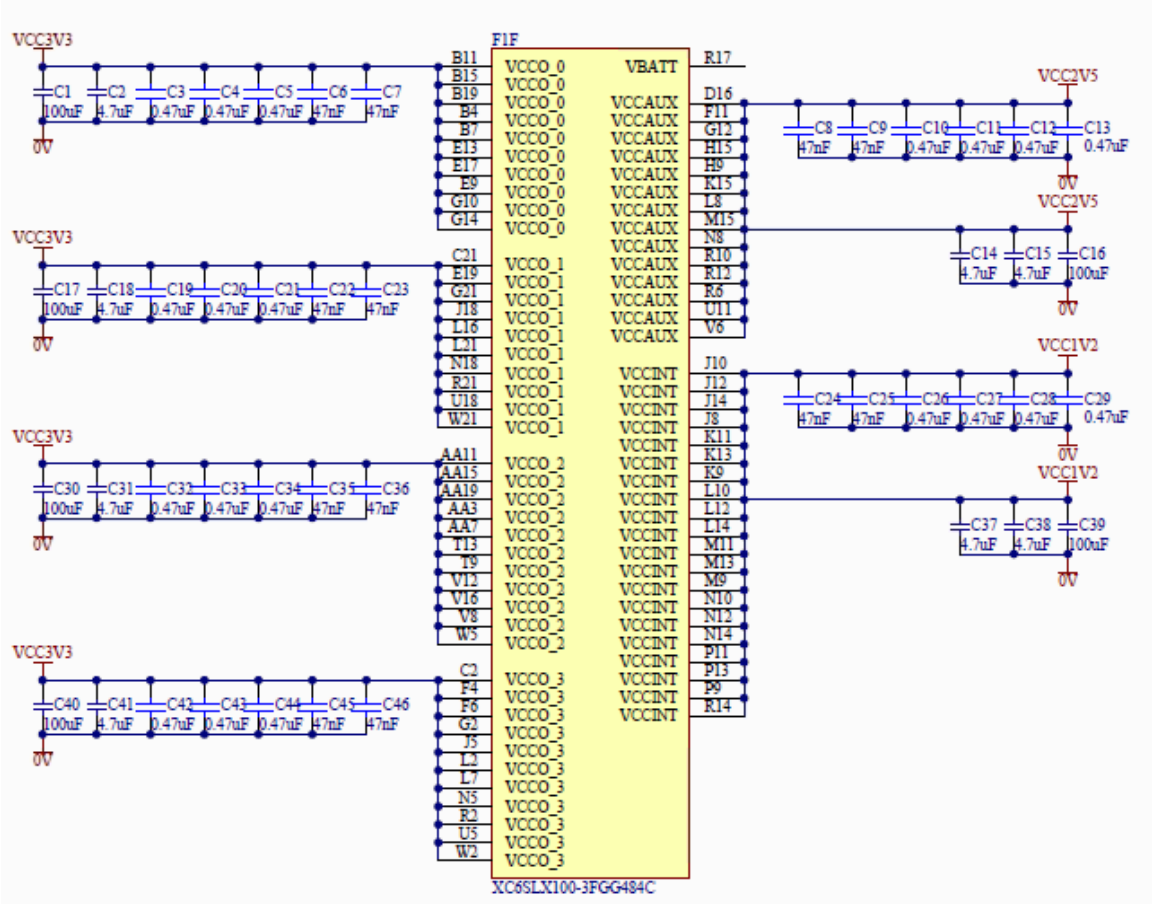


Fig. 4.5: FPGA decoupling capacitors

4.6.2 FPGA configuration and JTAG

A bitstream file that has been generated from the algorithm can be either downloaded into the FPGAs voltatile random access memory or in the non-voltatile external flash memory PROM (XCF04S). JTAG protocol which is a serial mode of communication is used to program either of the two devices. The circuit design for device configuration is shown in Fig. 4.6. The 3 pin header P3 is used for mode selection to switch between FPGA and PROM. Switch S1 is used for program reset for both devices and in case of the FPGA, clears the volatile program memory and sets all I/O pins to the high impedance state.

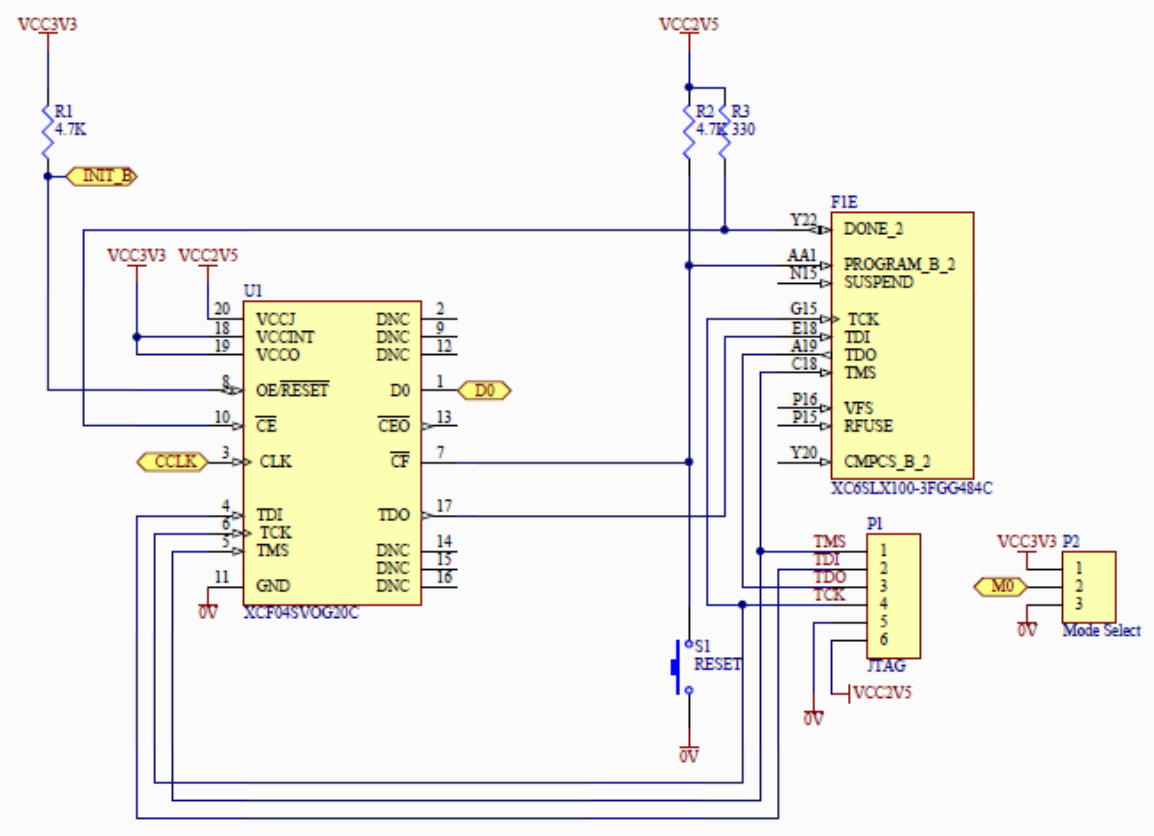


Fig. 4.6: FPGA configuration and JTAG

4.6.3 External Oscillator

An external oscillator is populated in the designator X1 as shown in Fig. 4.7. The user can select between a single ended 50 MHz option or a differential 200 MHz oscillator. The footprint has been designed to fit both packages. Clock configuration in the software needs to be correspondingly set to ensure correct operation.

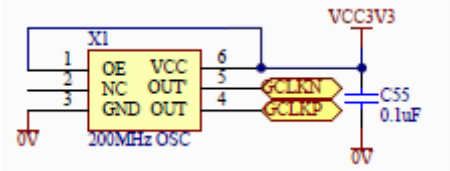


Fig. 4.7: External Osciallator

4.6.4 Analog to digital converter AD7606

The AD7606 is a 8 channel SAR ADC and the details are provided in section 4.5. Fig. 4.8 shows the circuit design of the ADC and its connection to the FPGA I/O banks. V1:V8 are SMA sockets where external analog signals are connected to the board. Switches P3 and P4 are used for input range and serial/parallel mode interface selections respectively. Various control lines such as CNVSTA, CNVSTB, RD'/SCLK, BUSY, CS' are used for communication between the FPGA and the ADC. For serial mode data is sent through DA and DB while in parallel mode the pins DB0:DB15 act as a 16 wire bus for data transmission. The conversion voltage reference is drawn internally. The analog inputs have a separate ground plane that is tied to the digital ground at a single point to ensure there is no noise coupling between the analog and the high speed digital circuits. Decoupling capacitors (not shown) are also provided as per datasheet recommendations.

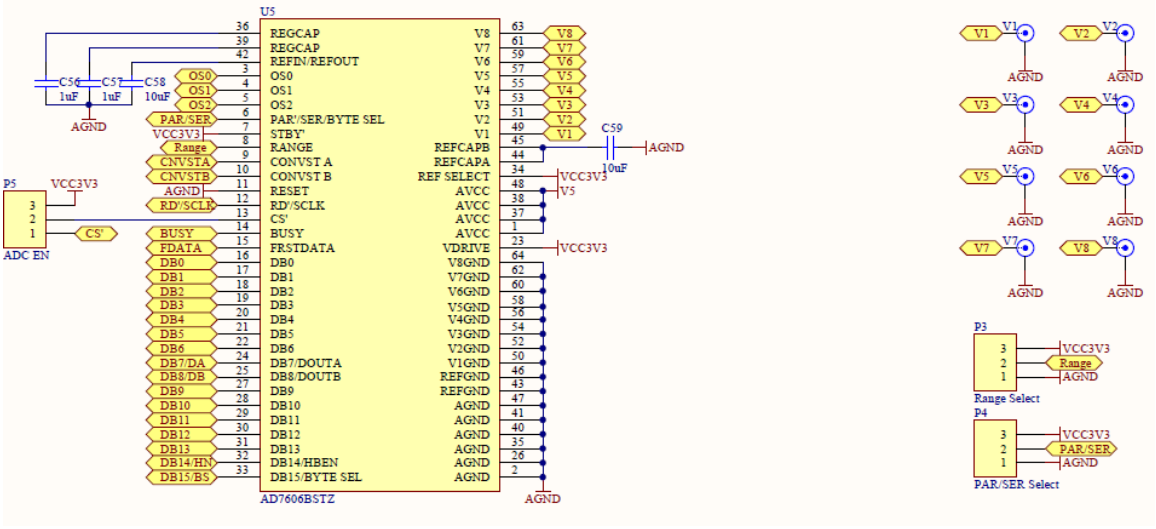


Fig. 4.8: ADC design using AD7606

4.6.5 Quadrature Encoder

The electric drives lab uses a Timken M15 magnetic encoder. To receive the quadrature pulses as well as the index pulse, a input buffer U10 is provided that provides low input impedance to the signals through connector J7. To disable the encoder circuitry and switch to using the pins associated with the encoder in the GPIO port J4, a selector port P7 is provided.

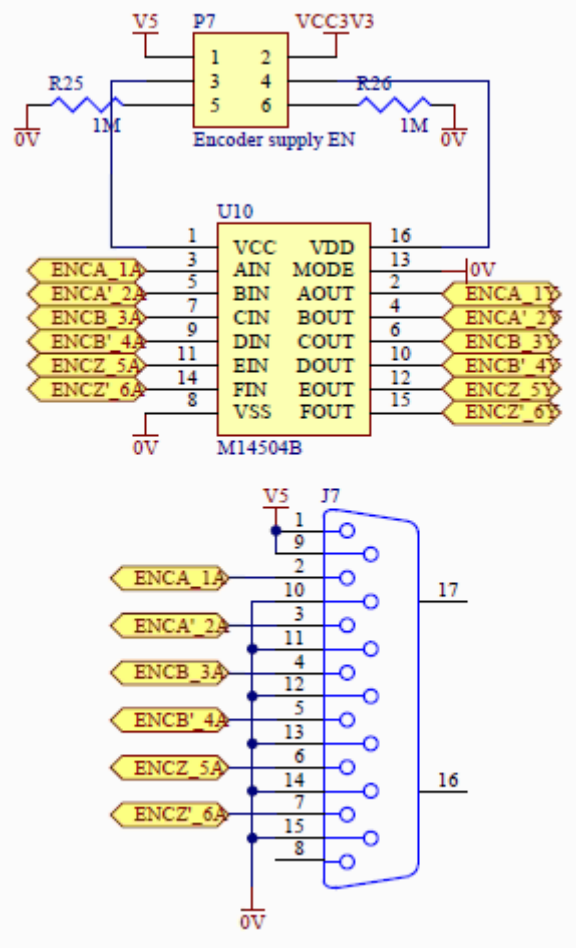


Fig. 4.9: Quadrature encoder design

4.6.6 Inverter interface port

The inverters used in the electric drives lab require 9 PWM channels to operate 3 inverters. It also requires inputs such as shutdown and fault clear logic lines. These signals are converter from the logic 3.3V to a 5V level through the logic translator ICs U6:U9. Through port P6 the logic translators can be disabled or enabled and the I/O lines can be cleared up for use in the GPIO port J2 if needed.

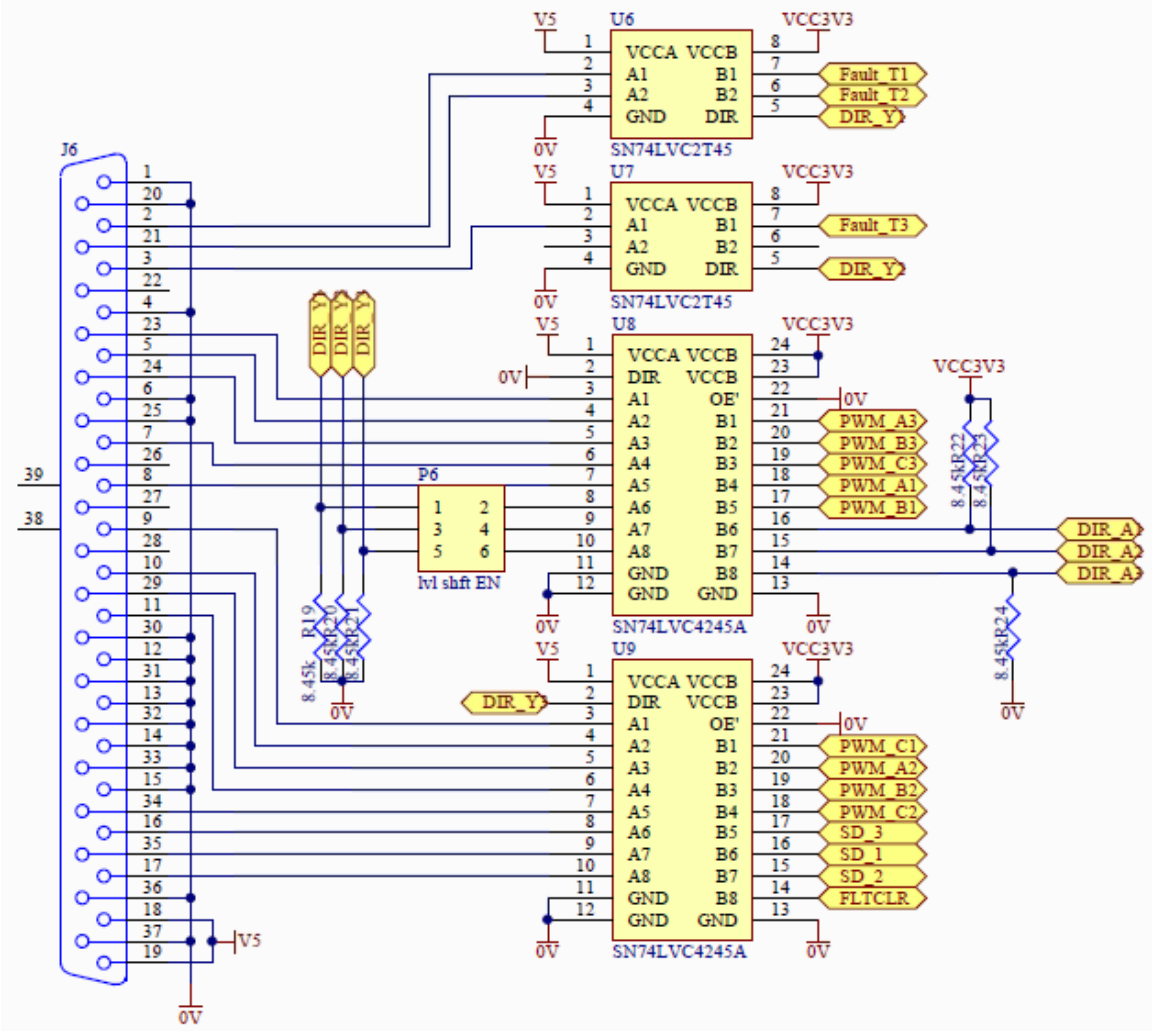


Fig. 4.10: Inverter interface design

4.6.7 User LEDs and Switches

Additional circuitry composed of tactile switches, slide switches and LEDs have been designed that are connected to the FPGA I/O pins. This can be used to provide external inputs to the FPGA and also to indicate change in internal states through the LEDs. The switches are connected to the I/O pins through a resistor network to limit the input current levels and hence have safe operation in the FPGA.

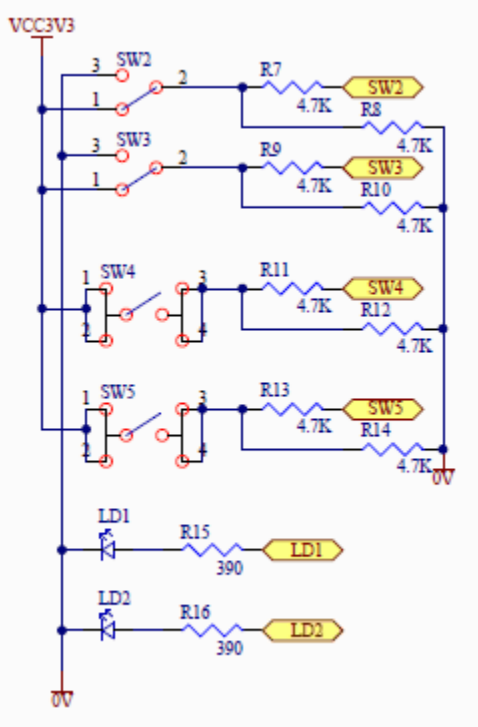


Fig. 4.11: User Switches and LEDs

4.6.8 General Purpose I/O (GPIO) Ports

The Spartan 6 XC6SLX100 -3FGG484C has 480 user I/O lines which is very useful for extending the chip for a multitude of applications. Hence 4 sets of 80 bidirectional I/O lines are routed to the 4 GPIO ports J2, J3, J4 and J5 as shown in Fig. 4.12. The ports have 100 pin Hirose connectors for easy expansion to other daughter board interfaces.

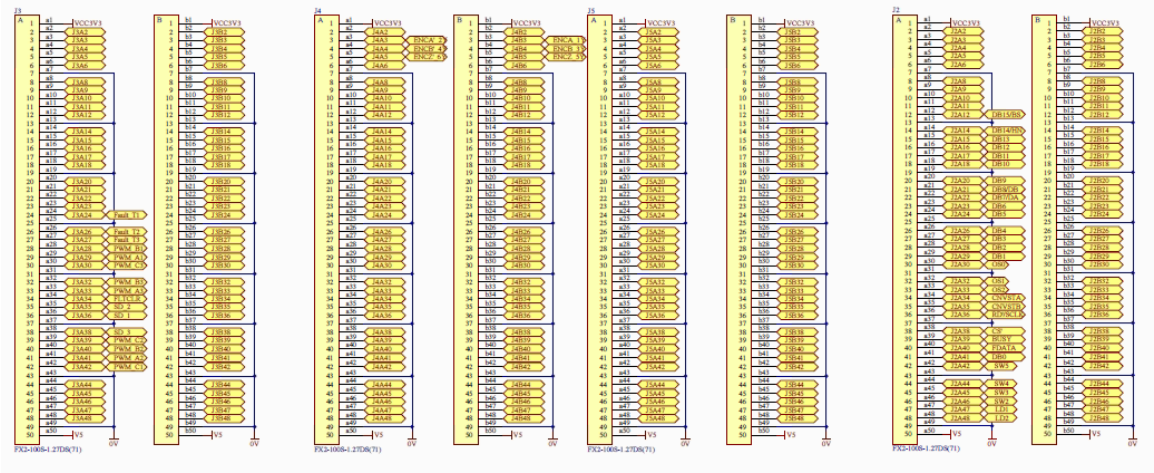


Fig. 4.12: GPIO ports

4.7 General outline of board layout

The dimensions of the board are 3.93 x 7.5 inches. A six layer design has been adopted with a layer stack shown by Fig. 4.14. There are four signal layers and two internal power layers of which one is dedicated to the board ground. Interconnection among the various layers is done by vias.

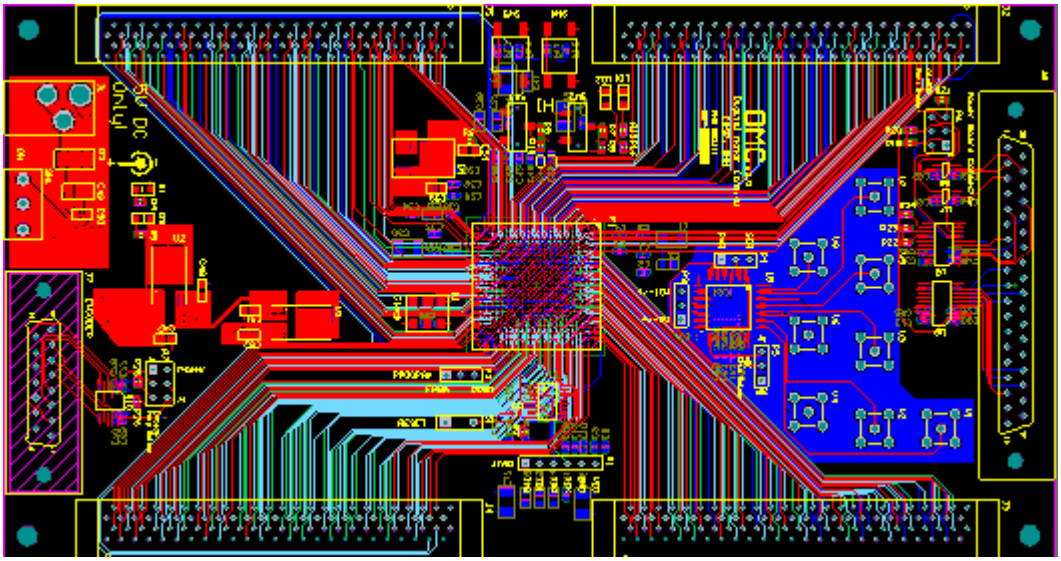


Fig. 4.13: PCB outline

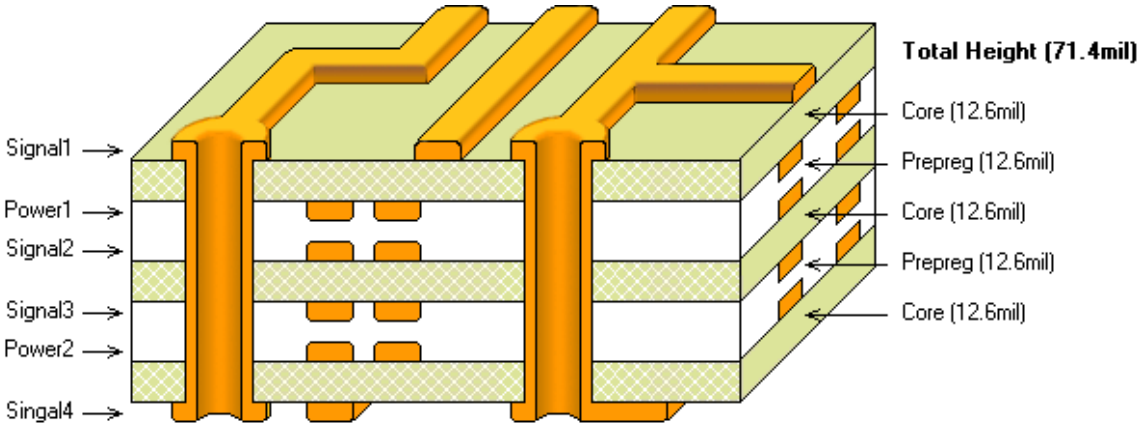


Fig. 4.14: Layer stack

As seen in Fig. 4.13, the SMA sockets for analog inputs occupy the top part of the board. A separate analog ground plane on the bottom layer of the board surrounds the SMA connectors and is tied to the internal ground plane through a single via. Power supplies, power socket and power supply indicators are placed in the bottom right of the board. Traces wider than 40 mils are used for the power circuit. The board has two internal power layers - one for ground and one for power. The ground and power lines are connected to the internal layers through a group of stitching vias. There are no dedicated power traces running to the components, since the power is taken directly from the power planes using appropriate vias. The FPGA and PROM chips are placed in the middle of the board. De-coupling capacitors surround the FPGA on all sides and are arranged in an ascending order of value. The capacitors are generally placed on the bottom layer, as close as possible to the power pins and appear in the power chain before the power pins of the chips. The external clock oscillator is placed as close as possible to the FPGA and it has been made sure they are interconnected through direct traces. All signals have a uniform trace width of 6 mils that ensures a compact design. In general, the board has been designed based on placement and layout recommendations found in the data-sheets and application notes from the manufacturers. The goal has been to minimize noise issues, with an emphasis on signal integrity and protection.

Chapter 5

Results and Discussion

The hardware-in-the-loop controller board was designed, laid out and assembled. Fig. shows a picture of the assembled board. Basic features of the board, such as connectivity, operation of power supplies, etc. were tested and found to be normal.

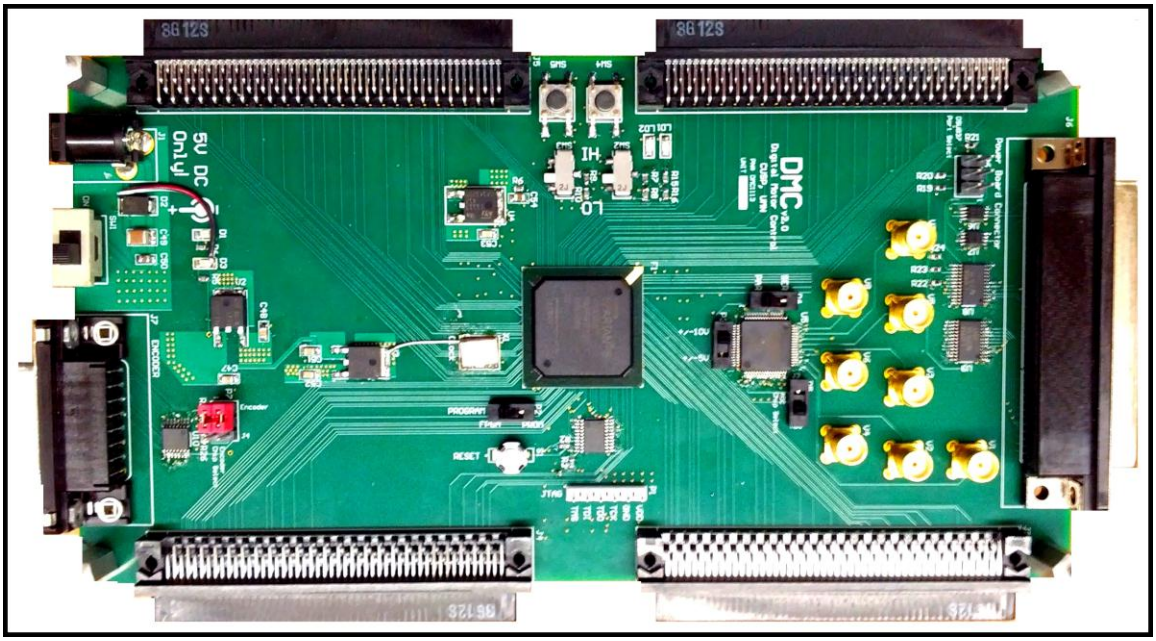


Fig. 5.1: Assembled board

5.1 Electric Drives Lab Experiments

To demonstrate the capabilities of the system that has been developed, the experiments conducted in the electric drives laboratory have been re-developed in the System Generator environment using the Xilinx Blockset and the Power and Motion Toolbox. All experiments have been tested to prove the efficacy of the new system. The experiments are listed below:

- Characterization of DC Motor
- DC Motor Speed Control
- Four-Quadrant Operation of DC Motor
- Determination of Induction Machine Parameters
- T-s Characteristics and Speed Control of Induction Machine
- PMAC Machine

5.1.1 Two Pole DC Motor Model

The model of a two-pole DC motor is shown in Fig. 46. The output voltage control of a two-pole DC switched mode converter was implemented in real-time. The purpose of the setup is to obtain a variable DC voltage at the output of the power converter, while controlling its amplitude with a hardware-in-the loop co-simulation using System Generator. The DC motor is connected to the output of the power converter and the variable voltage applied to the motor changes the speed of rotation.

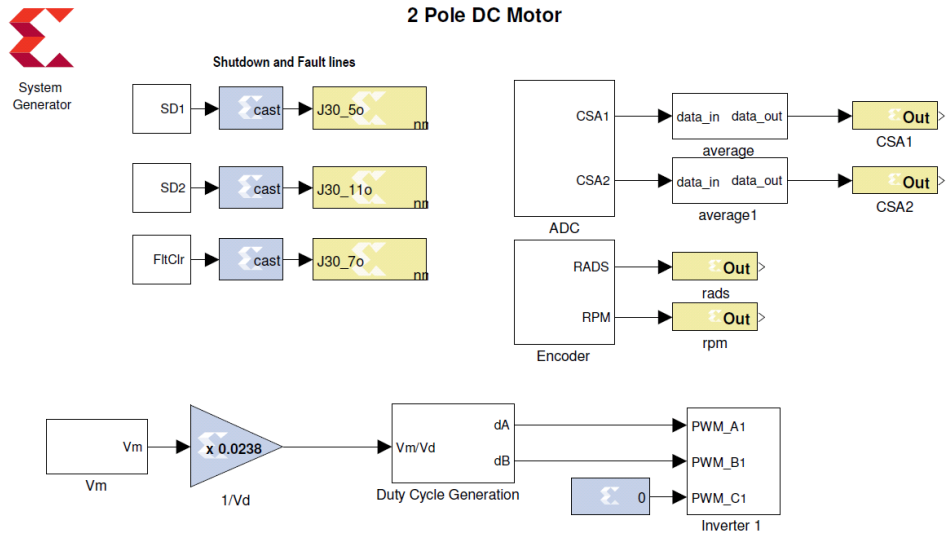
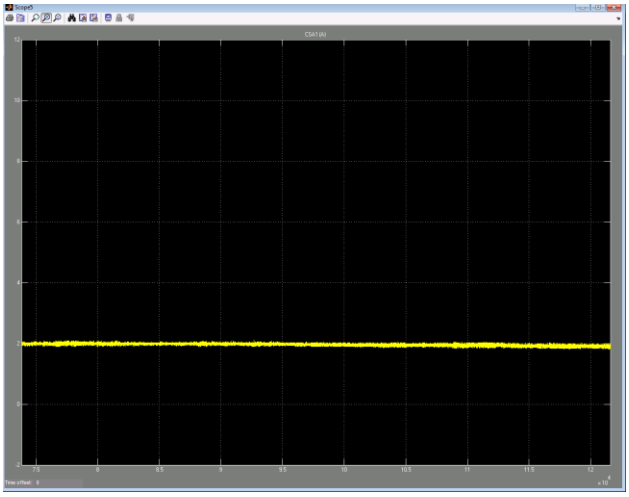
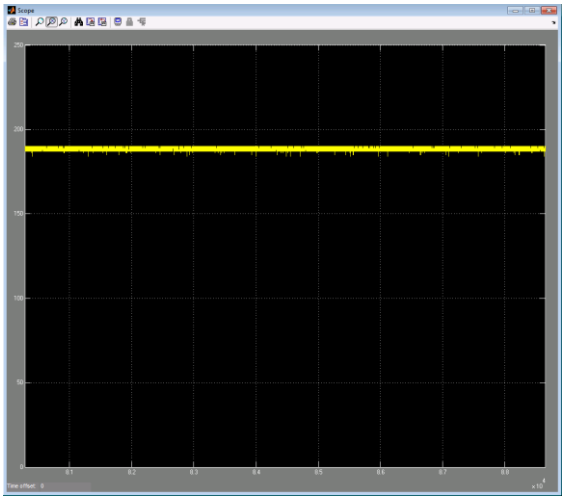


Fig. 5.2: Two pole DC motor model

Results:



(a) Motor Current, I_a = 2A



(b) Motor Speed W = 200 rad/s

Fig. 5.3: Motor Current and Speed for V_{motor} = 20V

5.1.2 DC Motor Speed Control

The model of a DC motor closed loop speed control is shown in Fig. 5.4. The purpose of this setup is to design and implement a closed loop speed control of a DC motor drive. The controllers have been designed and tested in a simulation model of the DC motor. Once the parameters are tuned the model of the DC motor is replaced with the real motor. The controllers can be further tuned in real-time using the hardware-in-the loop co-simulation environment offered by System Generator.

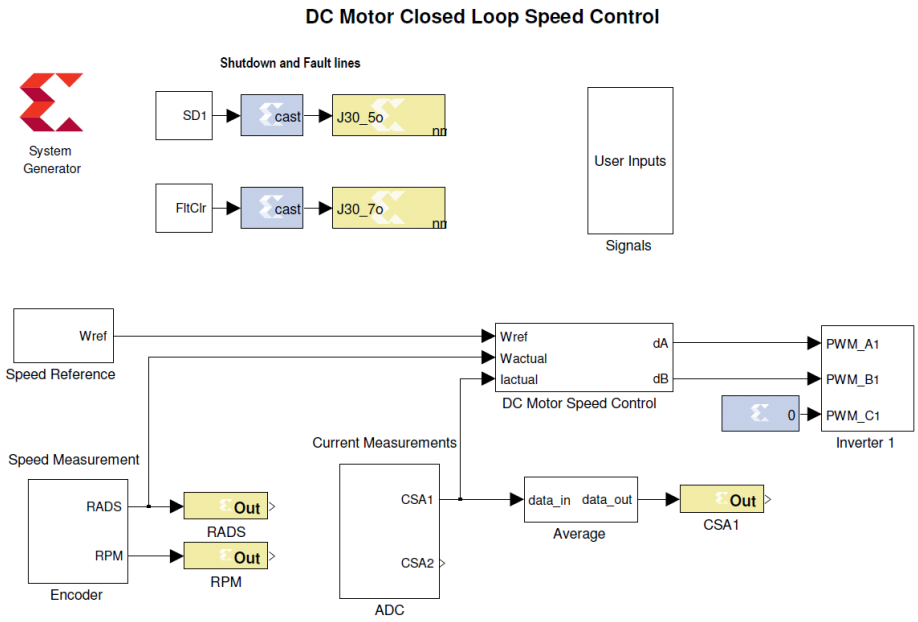


Fig. 5.4: DC motor closed loop speed control model

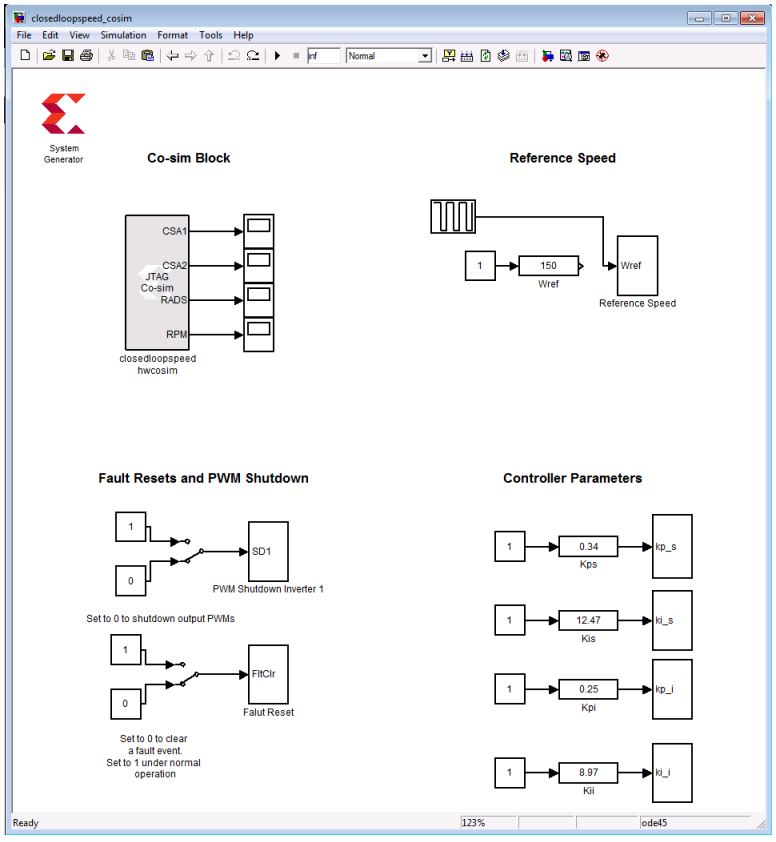
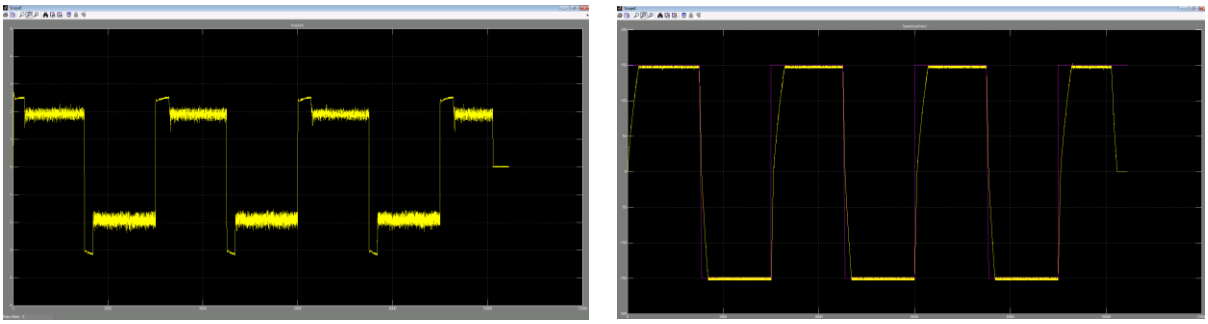


Fig. 5.5: DC motor speed control co-simulation GUI environment

Results:

The closed loop system was put to test with a speed reference signal that changed between +150 rad/s to -150 rad/s and as can be seen in Fig. 5.6(b), the actual motor speed closely tracks the reference speed signal (shown in pink). The motor current then correspondingly changes polarity and has a finite settling time between transitions. We do notice the current to have cut-off at 3A during its transient period. This is due to the saturation settings in the integral controller of the outer speed loop.



(a) Motor Current, I_a = =/-2A (b) Motor Speed W = +/-150 rad/s

Fig. 5.6: DC motor current and speed under closed loop

5.2 Conclusion and Future Work

This thesis presents the design and implementation of a hardware-in-the-loop converter for electric drive applications. The control system is designed in MATLAB/Simulink using System Generator for DSPs and the Xilinx Blockset. An additional library called Power and Motion Toolbox has been created which provide many of the blocks needed to implement various electric drive control models. A Spartan 6 FGPA based controller board has been designed which implements control algorithms and also forms a communication link between the power board and the computer passing various control system parameters in real time. The controller board also reads the analog signals from

the sensors on the power board and quadrature encoder data thereby generating PWM signals for driving the gates on the power board.

The system has been built and the essential features of the controller have been verified. All the nine experiments of electric drives laboratory have been designed and developed in the new system and verified in hardware to have the same performance as dSPACE. It was learnt that the concept is feasible from a cost perspective as it effectively eliminates the expensive dSPACE controller while providing a set of features that compare and often times exceed the capabilities of the former. There have been issues with the JTAG communication link between the PC and the FPGA: the bandwidth of data transmission is limited by the processing capabilities of the PC and hence there is a loss of data which has higher frequency. Future work would mainly involve working towards fixing this issue to make the co-simulation more seamless. The system must be put into further rigorous tests to match the speed and bandwidth requirements for motor drives. Further work might also involve updating the PCB layout and design to match new footprints and make minor changes in the schematics.

Building upon this controller, a set up could be developed for teaching electric drives lab in universities, which would enable the students to design a control system for motors and test them instantly, without the need for expensive industry grade hardware. The generous GPIOs could further be used for researches to develop applications using the powerful Spartan 6 FPGA and use the controller in many different fields as a powerful research tool.

References

[1] N. Mohan, Electric Drives an integrative approach, MNPERE ,2000.

[2] N. Mohan, Advanced Electric Drives: Analysis, Control and Modeling using Simulink R. MNPERE, 2011.

[3] Department of Electrical and Computer Engineering, DSP Based Electric Drives Laboratory User Manual. University of Minnesota, August 5th, 2011.

[4] J. Mailloux, S. Simard & R. Beguenane. FPGA implementation of Induction Motor Vector Control using Xilinx System Generator, *International Conf. on circuits, systems, electronics, control & signal processing*, Cairo, Dec. 29-31, 2007.

[5] Francesco Ricci and Hoang Le-Huy, Modeling and simulation of FPGA-based variable-speed drives using Simulink, *Mathematics and Computers in Simulation*, vol. 63, Issues 3-5, 17 Nov. 2003, pp. 183-195.

[6] F. Ricci, H. L. Luy, "An FPGA-Based Rapid Prototyping Platform for Variable-Speed Drives", Industrial Electronics Society, Vol. 2, pp. 1156-1161, IEEE, 2002.

[7] Xilinx Corp., "Xilinx System Generator v2.1 Reference Guide for Simulink, www.xilinx.com.

[8] ISE Design Suite User Guide and tutorial

[9] Franklin, Powell & Emami-Naeini, Feedback Control of Dynamic Systems, 4th Ed. Prentice Hall, New Jersey 2002.

Appendix A

Getting Started with System Generator

- An Example

A.1 Introduction

In this section an introduction to System Generator will be provided with an example design in Simulink. We will create a PWM modulator subsystem which compares two duty signals with a high frequency carrier. The duty signals have the following two equations:

$$d_A = \frac{1}{2} + \frac{V_O}{2 * V_d}$$

$$d_B = \frac{1}{2} - \frac{V_O}{2 * V_d}$$

A.2 Objectives

In this section we will accomplish the following:

Understand the basics of building a design in System Generator

Simulate the PWM modulator as a Simulink model

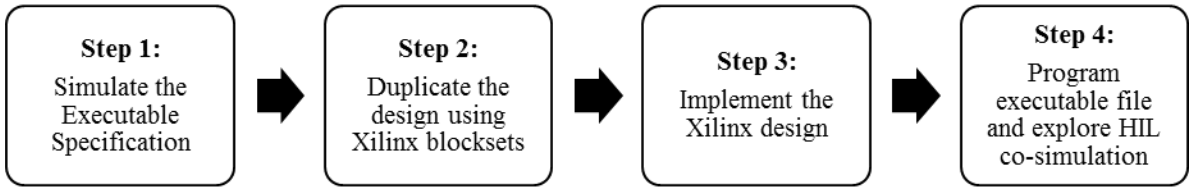
Re-create the PWM modulator using Xilinx blockset for simulation

Run the System Generator token to generate a Xilinx FPGA bitstream

Note: Please check the System Generator for DSP release notes to insure that the proper versions of ISE Design Suite and MATLAB are installed on your machine. Failure to have the proper tool versions installed may result in unexpected behavior.

A.3 Procedure

In Step 1, you open and simulate a Simulink blockset-based design that serves as an "executable specification". In Step 2, you re-create the Simulink design using the Xilinx blockset. In Step 3, you take the Xilinx executable specification through the full implementation flow. In Step 4, you program the executable file into hardware and work with hardware-in-the-loop co-simulation.



A.3.1 **Step 1:** Simulate the Executable Specification

- Create a new directory for the experiment (say *AppB*).
- Start Matlab and set the path to this directory.
- Type "simulink" at the command prompt and create a new model from **File** menu and save as duty_simulink.mdl.
- Access the Simulink library by clicking **View > Library Browser.**
- In the Library Browser expand the **Simulink** tree to find the blocks needed to create the two duty equations as shown in Fig. A.1.
- Simulink blocks usually have properties that can be modified by double-clicking on the blocks. Double click on the **Gain** block and edit the value of the field 'Gain' as

needed. Similarly set the values in the **Constant** block as desired. "Vd" is the DC voltage set as a global variable in the Matlab prompt.

- Add a **Scope** to the model from **Simulink → Sinks.** Double click the Scope to open it. Click on the icon in the top called "Parameters". Under the Axes field set the number of axes as 2 and press ok. Right click on the scope area to select Axes properties. Set the Y-min as 0 and Y-max as 1. Do this for the other window as well.

- Type the value of "Vd" at the Matlab prompt, Vd = 42 (DC source is 42V)

- The simulation model is now ready. However before running the simulation parameters need to be changed. Go to **Simulation** menu and select **Configuration Parameters.** Set the 'Stop time" parameter to 1.

- Run the simulation by clicking on the triangular play button on the top. Double click on the scope after the simulation finishes. The result should look similar to the one shown in Fig. A.2.

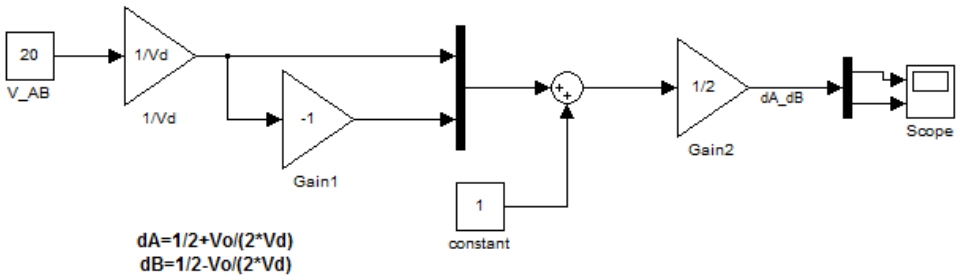


Fig. A.1 duty_simulink.mdl

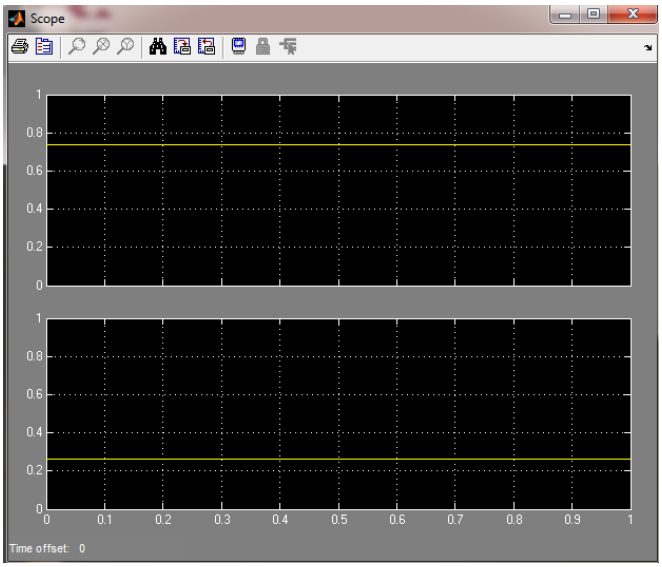


Fig. A.2 Plot of dA and dB for V_AB=20

A.3.2 **Step 2:** Duplicate the design using Xilinx blocksets

- In Maltab type "simulink" at the command prompt and create a new model from **File** menu and save as duty_sysgen.mdl.

- Open the Simulink Library and expand the **Power and Motion Toolbox** to find the needed blocks to complete the design as shown in Fig. A.3.

- Create a Xilinx version of the multiply/add design using the Xilinx blocks. Please note that you must use Xilinx **Gateway In**/ **Gateway Out** blocks to define the FPGA boundary and you must also place a Xilinx System Generator token in the design. Leave the **AddSub**, **Gateway In** and **Gateway Out** blocks to its default values. Make changes to the **Gain** and **Constant** blocks to reflect the model shown in Fig. A.3.

- The **Gain** and **Constant** blocks that are connected before the **Gateway In** are from the **Simulink** toolbox and are not to be confused with the Xilinx blocks in blue. Add a **Scope** to the model from **Simulink→Sinks.** Double click the Scope to open it. Click on the icon in the top called "Parameters". Under the Axes field set the

number of axes as 2 and press ok. Right click on the scope area to select Axes properties. Set the Y-min as 0 and Y-max as 1. Do this for the other window as well.

- Type the value of "Vd" at the Matlab prompt, Vd = 42 (DC source is 42V)
- The simulation model is now ready. However before running the simulation parameters need to be changed. Go to **Simulation** menu and select **Configuration Parameters.** Set the parameters to the following values:
    - Stop time : 1
- Run the simulation by clicking on the triangular play button on the top. Double click on the scope after the simulation finishes. The result should look similar to the one shown in Fig. A.2.

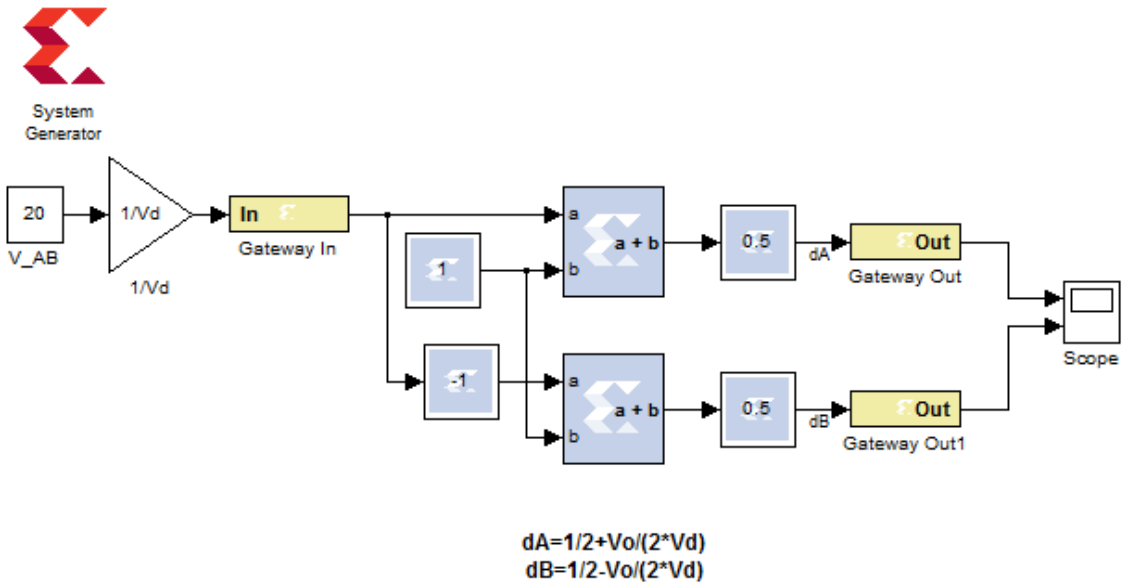


Fig. A.3 duty_sysgen.mdl

- Compare the results from the executable specification vs. the Xilinx implementation using a **Subtractor** from "Simulink/Math Operations" library as shown in Fig. A.4.

This is an important model-based design concept. Observe that there is no computation error displayed in the Scope as the two implementations are “bit and cycle accurate”.

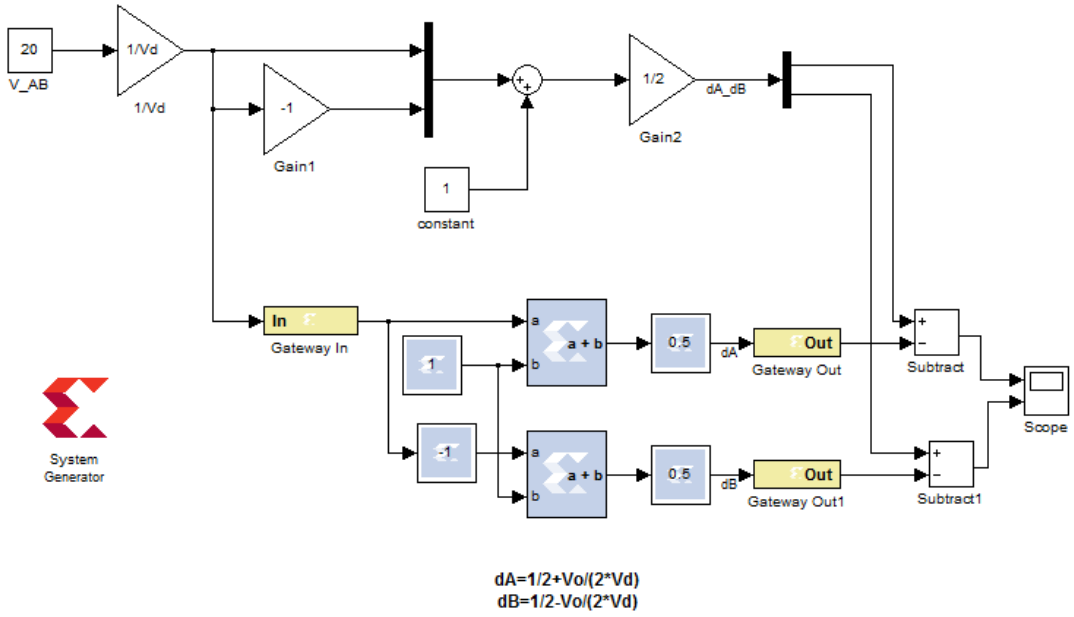


Fig. A.4 duty_compare.mdl

A.3.3 **Step 3:** Implement the Xilinx design

The next step is to perform the FPGA implementation steps that include RTL generation, RTL synthesis and Place and Route.

- Remove the Simulink blocks from your Xilinx design so that it looks like Fig. 5. Save this as a different model file called “duty_sysgen_xilinx.mdl”. This is done because only the Xilinx blocks can be implemented in the FPGA. The Simulink stages before and after the **Gateway In** and **Gateway Out** blocks will be connected later in the co-simulation model as explained in Step 4.

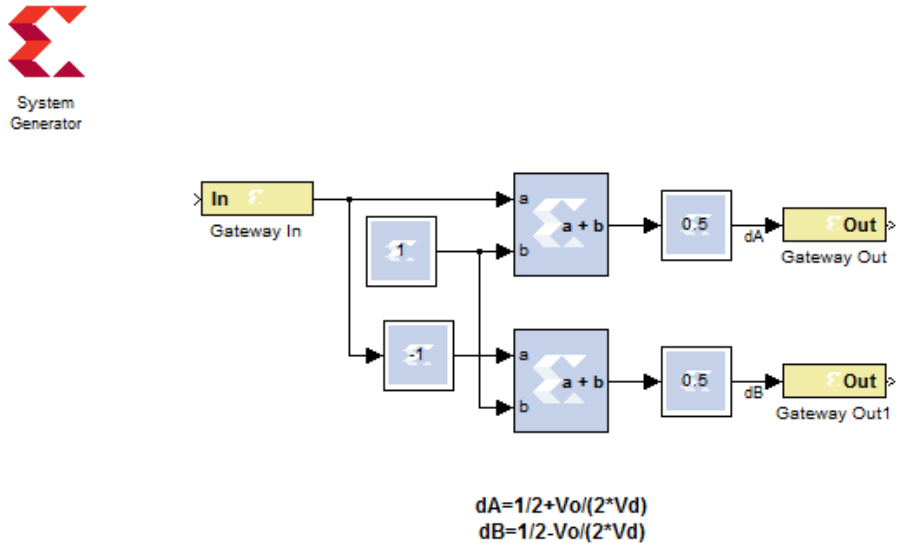


Fig. A.5 duty_sysgen_xilinx.mdl

- Double-click on the System Generator token and set the Compilation target to Hardware Co-Simulation/DMCv1.0 or DMCv2.0 (based on the hardware platform that is being used), as shown in Fig. A.6.

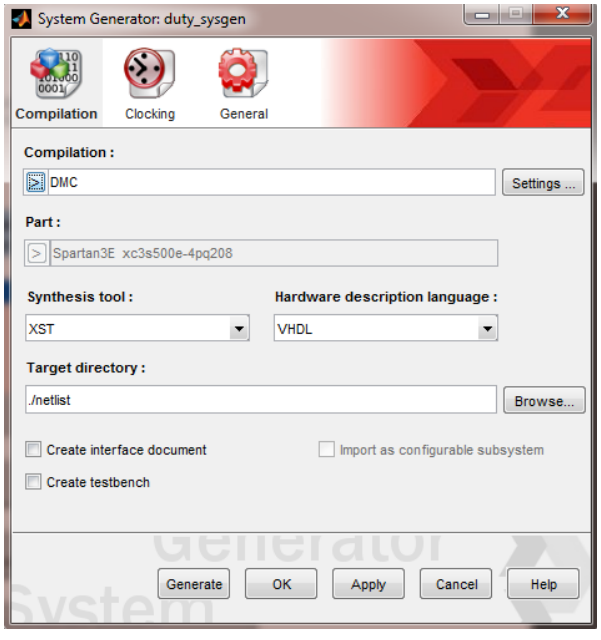


Fig. A.6 System Generator token

- Click the Generate button to initiate the implementation process. System Generator will automatically execute the RTL generation, logic synthesis and place and route programs to create an FPGA programming file. Optionally, you can select to generate an intermediate format such as HDL (logic synthesis) or NGD (place and route) and run these steps interactively. We will, for the purpose of the lab work with Hardware Co-Simulation.

- When the generation is complete, click on the Show Details button on the Compilation status dialog box as shown below. This will display the implementation transcript.

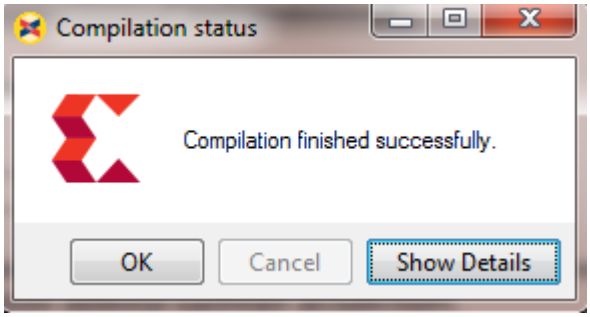


Fig. A.7 Compilation status

- You will find a new window that has the "duty_sysgen_xilinx hwcosim" block as shown in Fig. A.8. This is a block representing the generated "bitstream" file that will be programmed into the FPGA. The block displays the **Gateway In** and **Gateway Out** ports that were present in the design. Blocks from the Simulink toolbox can now be connected directly to these ports.

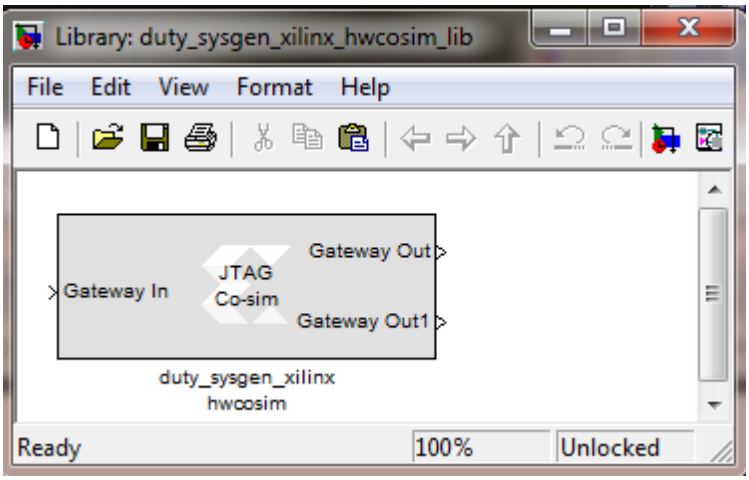


Fig. A.8 hwcosim block generated

- Create a new model file called "duty_sysgen_xilinx_cosim.mdl" in your working directory and connect it as shown in Fig. A.9. Remember to place the **System Generator** token in the model. Set the Stop time to "inf" (infinity) to run the model for an unconstrained time duration. Double click the gray block to open up the parameters. In the "Basic" tab under "Clocking", select the "Clock source" to "Free running". In the "Cable" tab under "Cable Settings", select the "Type" to "Digilent USB JTAB Cable". Press ok to complete the actions.

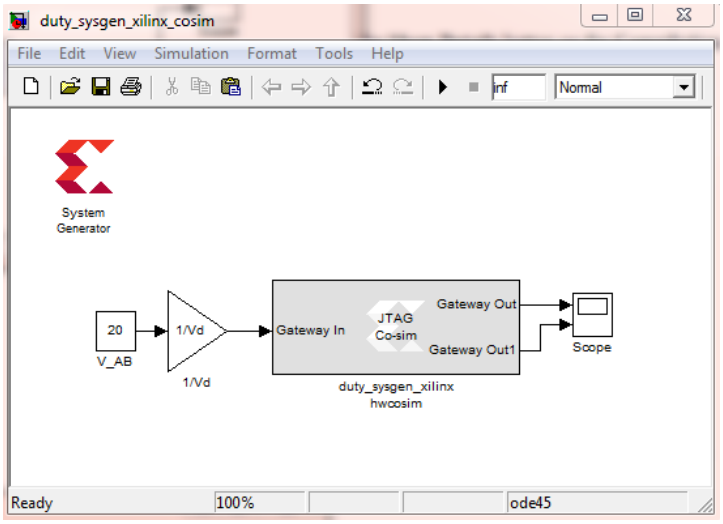


Fig. A.9 duty_sysgen_xilinx_cosim.mdl

A.3.4 **Step 4:** Program executable file and explore HIL co-simulation

The next step is to program the hardware platform and observe the hardware-software interaction in real-time. Hardware-in-the-loop allows the user to have a graphical interaction with the FPGA controller to send and receive data to and from the controller in real-time. In this example, we send the signal V_AB/Vd into the FGPA and receive the duty signal values dA and dB from the controller. The duty equations are however programmed in the FPGA.

- After following the instructions for the cosim model press the triangular play button in the top to initiate the model. When the play button is pressed, the bitstream file that is linked with the duty_sysgen_xilinx hwcosim block gets transferred to the FPGA through the high speed JTAG link. This establishes the co-simulation environment and hence data can be sent and received in real time.

- Observe the variation of the duty signal values dA and dB for different values of V_AB.

Appendix B

Power and Motion Toolbox

B.1 Introduction

For the purpose of the Electric Drives lab the Power and Motion Toolbox has been created which has a complete set of blocks that can be used to create drives models. The blocks have been created using the Xilinx blockset library using the concept of masked subsystems. The blocks are categorized as follows:

Control blocks

Inverter Signal blocks

Math blocks

Peripheral blocks

Sources blocks

Time Estimator blocks

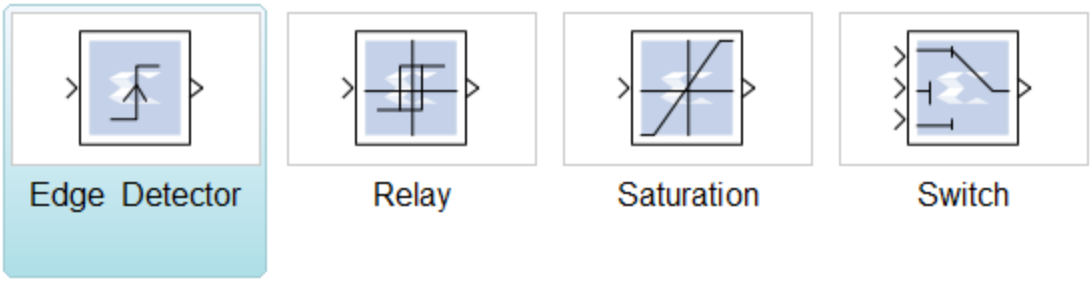
B.2 Description

In this section a brief description of each block will be provided along with the block constraints and usage examples. Additionally wherever required timing diagrams and output waveforms are provided for a better understanding. An overall list of blocks is show in Fig. B.1.

Fig. B.1 Power and Motion Toolbox blocks

B.1.1 Control blocks

Under the category of control blocks the following have been designed.



Edge Detector

Detect rising, falling or both edges of an input signal.

Description

This block compares its present input to its previous input. The output of the block is TRUE for one FPGA clock interval when the specified edge: Rising, Falling or Either is detected.

Data Type Support

The Edge Detector accepts signals of the type

- Boolean

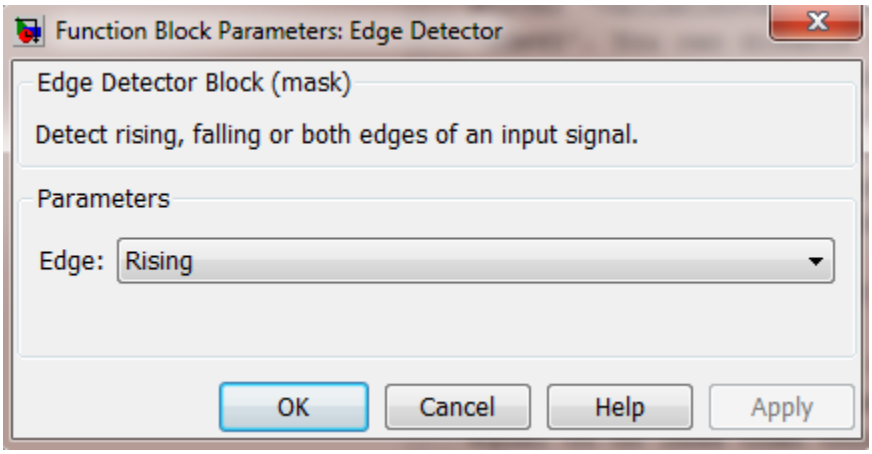
Output Data Type

- Boolean

Latency: 0 CLK

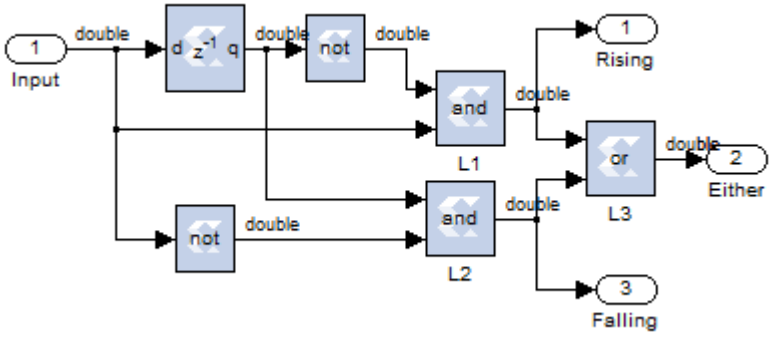
Parameters and Dialog Box

The Edge Detector block dialog box appears as follows:



Edge The type of edge that needs to be detected: Rising, falling or either.

Implementation



$$Rising = Input(n) \ \& \ \overline{Input(n-1)}$$

$$Falling = \overline{Input(n)} \ \& \ Input(n-1)$$

$$Either = \{Input(n) \ \& \ \overline{Input(n-1)}\} \mid \{\overline{Input(n)} \ \& \ Input(n-1)\}$$

Relay

Switch output between two constants. Output the specified 'on' or 'off' value by comparing the input to the specified thresholds. The on/off state of the relay is not affected by input between the upper and lower limits.

Description

The Relay block allows its output to switch between two specified values. When the relay is on, it remains on until the input drops below the value of the **Switch off point** parameter. When the relay is off, it remains off until the input exceeds the value of the **Switch on point** parameter. The block accepts one input and generates one output.

The **Switch on point** value must be greater than or equal to the **Switch off point**. Specifying a **Switch on point** value greater than the **Switch off point** models hysteresis, whereas specifying equal values models a switch with a threshold at that value.

Note: When the initial input falls between the Switch off point and Switch on point values, the initial output is the value when the relay is off.

Data Type Support

The Relay accepts signals of the type

- Fixed point signed
- Fixed point unsigned

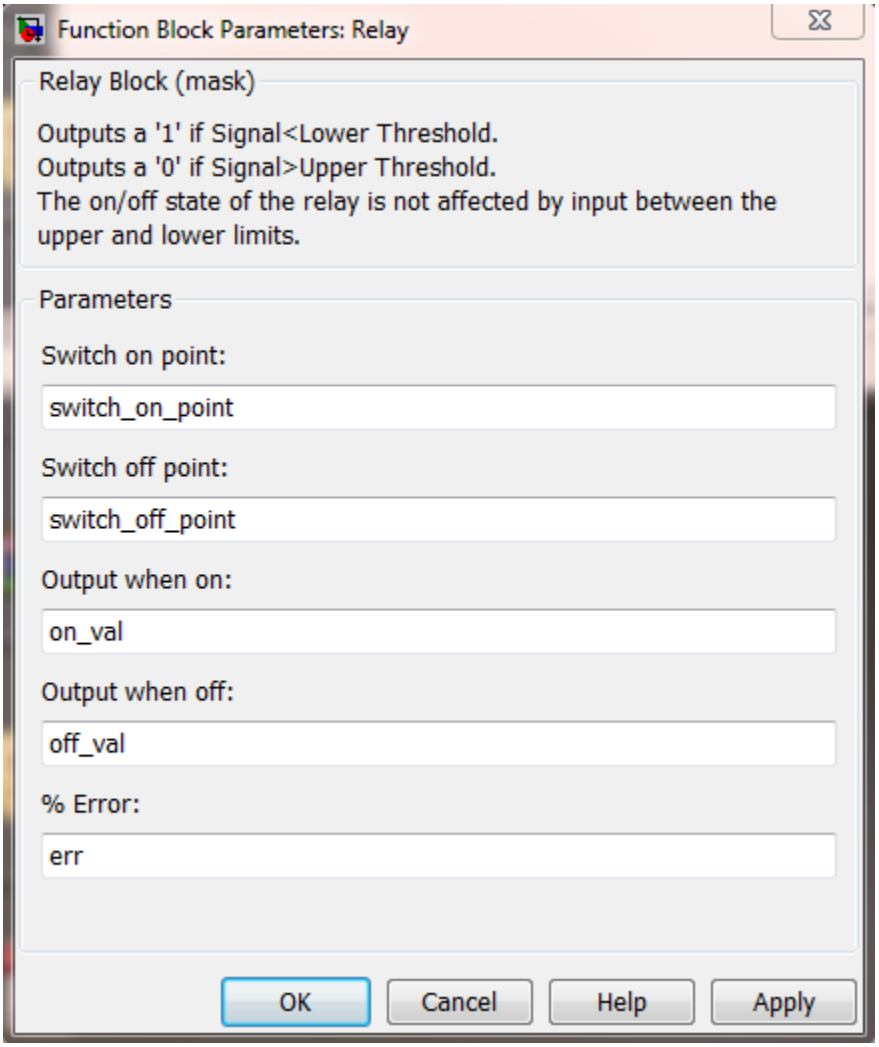
Output Data Type

- Fixed point signed

Latency: 2 CLK

Parameters and Dialog Box

The Relay block dialog box appears as follows:



Switch on point

The "on" threshold for the relay.

Switch off point

The "off" threshold for the relay.

Output when on

The output when the relay is on.

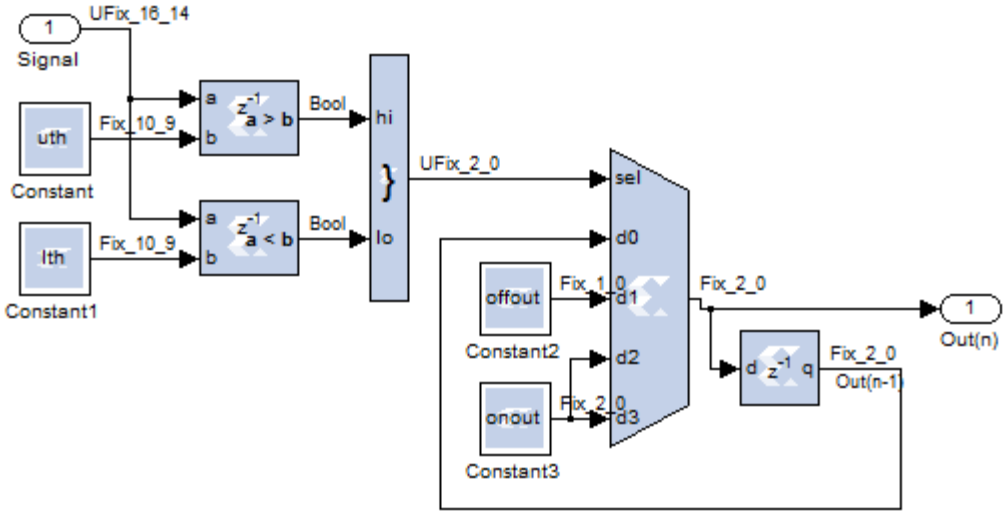
Output when off

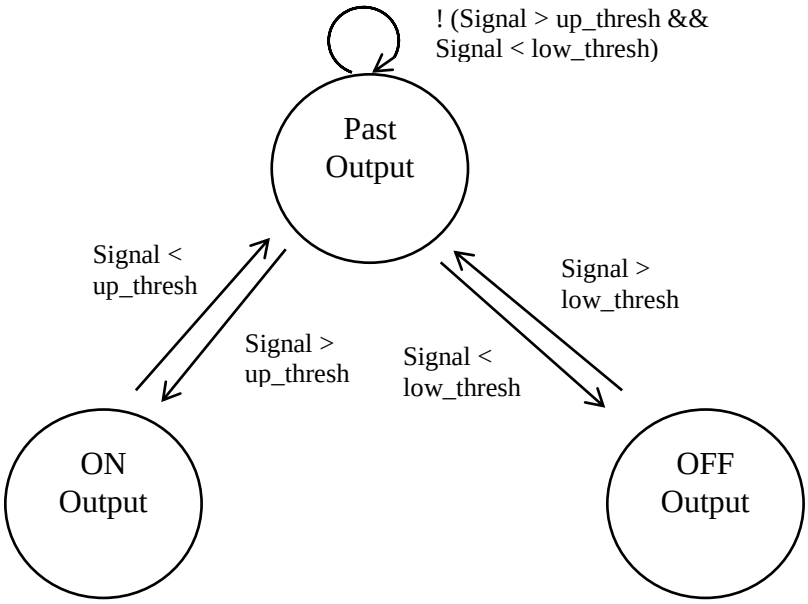
The output when the relay is off.

% Error

The percentage of error in the output when it is of type Fixed point. For the type Integer the error is zero.

Implementation





Relay Output State Machine

Saturation

Limit input signal to the upper and lower saturation values.

Description

The Saturation block imposes upper and lower limits on an input signal.

When the input is:	Where:	The block output is the:
Within the range specified by the Lower limit and Upper limit parameters	Lower limit ≤ Input value ≤ Upper limit	Input value
Less than the Lower limit parameter	Input value < Lower limit	Lower limit
Greater than the Upper limit parameter	Input value > Upper limit	Upper limit

When the Lower limit and Upper limit parameters have the same value, the block output is that value.

Data Type Support

The Saturation accepts signals of the type

- Fixed point signed
- Fixed point unsigned

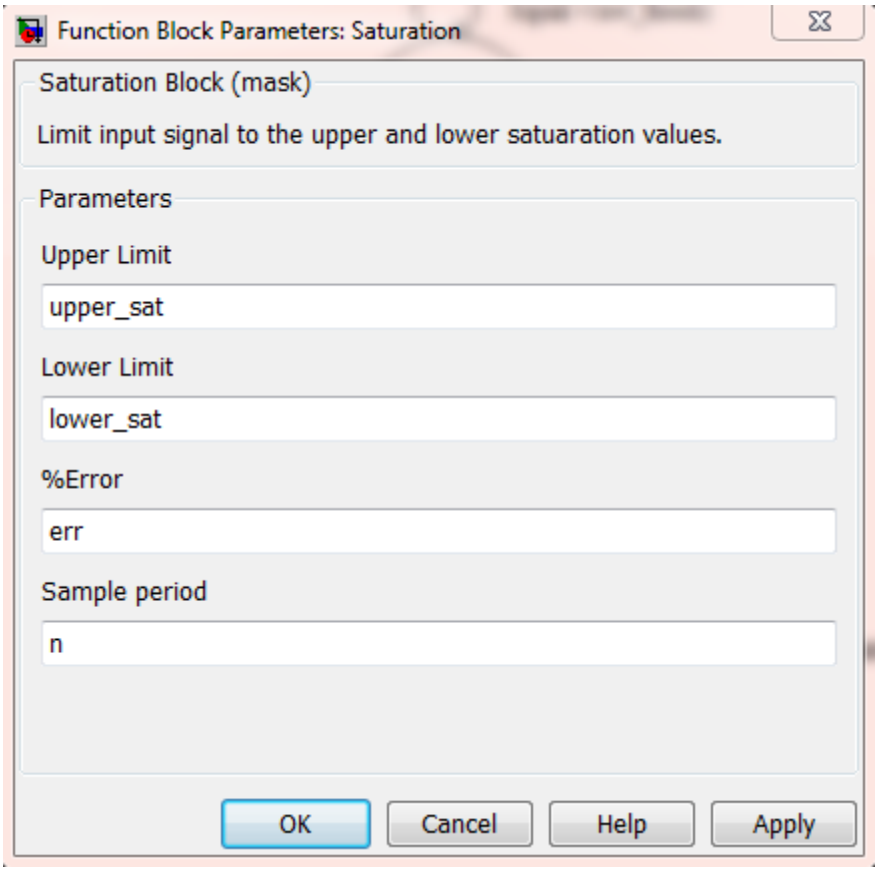
Output Data Type

- Fixed point signed

Latency: 1 CLK

Parameters and Dialog Box

The Saturation block dialog box appears as follows:



Upper Limit

The upper limit of the saturation value.

Lower Limit

The lower limit of the saturation value.

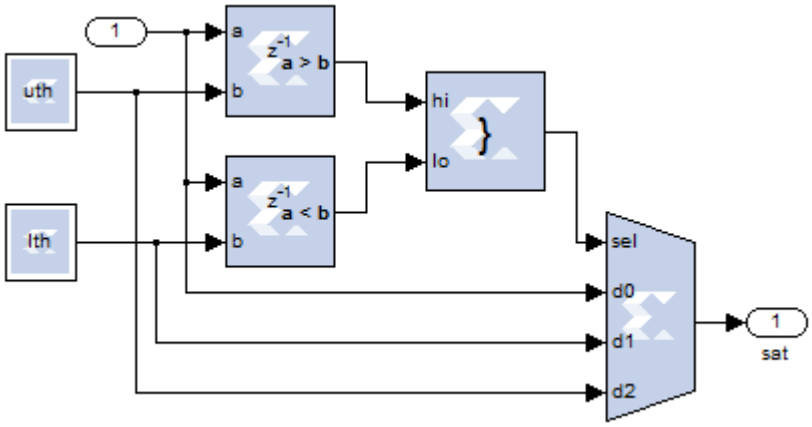
% Error

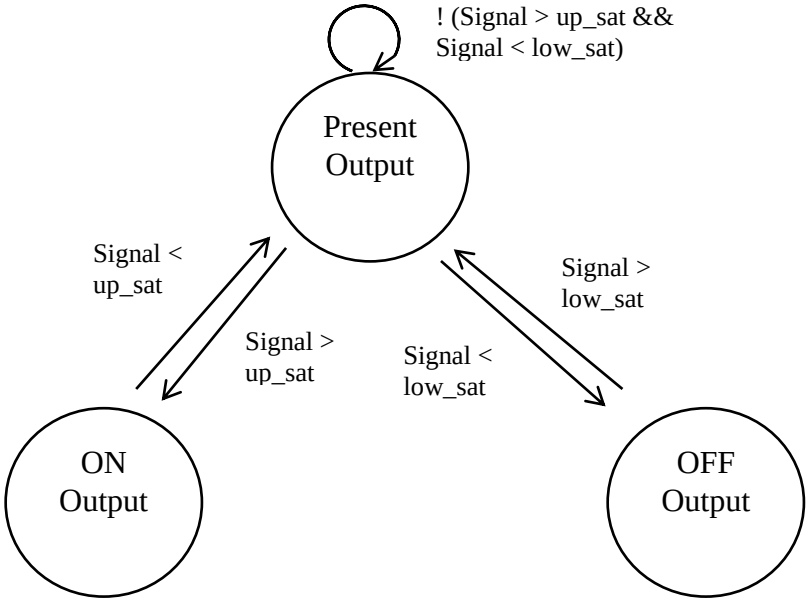
The percentage of error in the output when it is of type Fixed point. For the type Integer the error is zero.

Sample Period

Sets the sampling rate of the internal blocks of the Saturation subsystem.

Implementation





Switch

Switch output between first input and third input based on value of second input

Description

The Switch block passes through the first input or the third input based on the value of the second input. The first and third inputs are called data inputs. The second input is called the control input. Specify the condition under which the block passes the first input by using the Criteria for passing first input and Threshold parameters.

Limitations on Data Inputs

The sizes of the two data inputs can be different if you select Allow different data input sizes on the block dialog box. However, this block does not support variable-size input signals. Therefore, the size of each input cannot change during simulation or in hardware.

Block Icon Appearance

The block icon helps you identify Criteria for passing first input and Threshold without having to open the block dialog box.

Data Type Support

The Switch accepts signals of the type

- Fixed point signed
- Fixed point unsigned

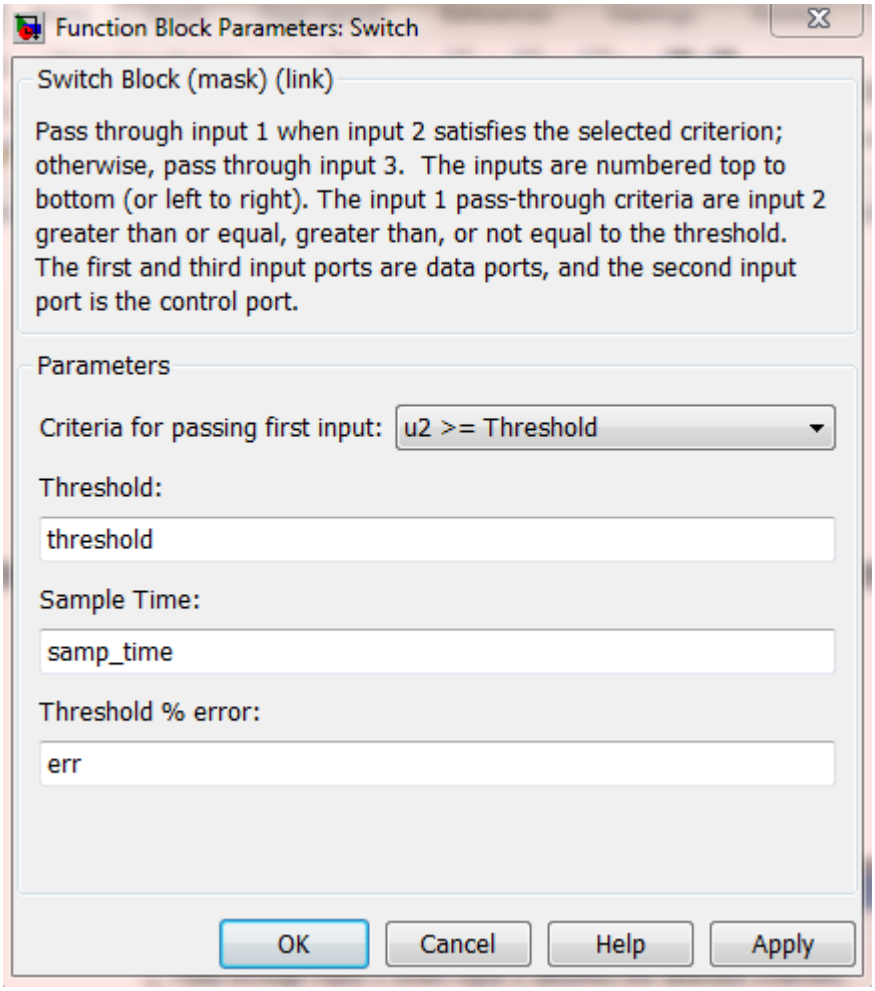
Output Data Type

- Fixed point signed

Latency: 1 CLK

Parameters and Dialog Box

The Switch block dialog box appears as follows:



Criteria for passing first input

Select the condition under which the block passes the first input. If the control input meets the condition set in the Criteria for passing first input parameter, the block passes the first input. Otherwise, the block passes the third input.

Settings:

- u2 >= Threshold, Checks whether the control input is greater than or equal to the threshold value. (Default)
- u2 > Threshold, Checks whether the control input is greater than the threshold value.

- u2 ~=0, Checks whether the control input is nonzero.

Note: The control input cannot be of the type Boolean.

Threshold

Assign the switch threshold that determines which input the block passes to the output.

Sample Time

Sets the sampling rate of the internal blocks of the Saturation subsystem.

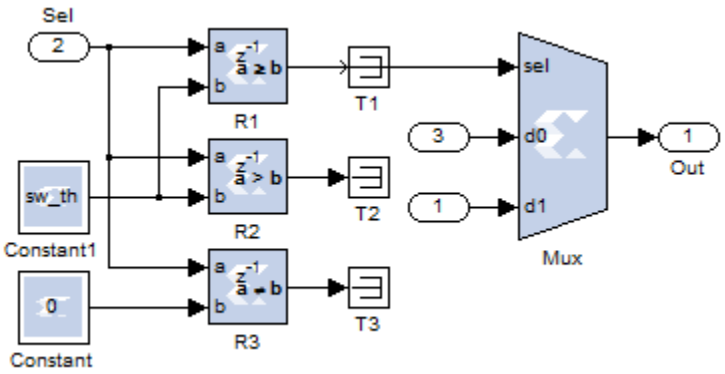
Threshold % error

The percentage of error in the threshold value when it is of type Fixed point. For the type Integer the error is zero.

Dependencies

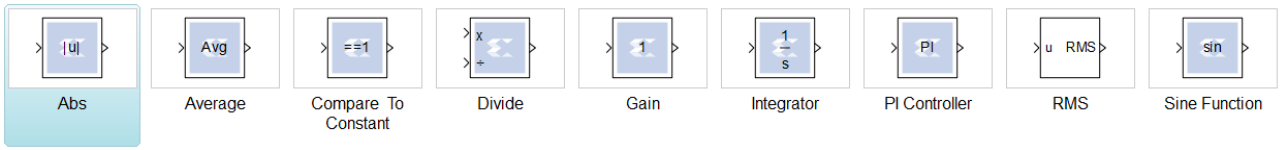
Selecting u2 ~=0 disables the Threshold parameter.

Implementation



Based on the settings of the parameter **Criteria for passing first input** the output of either R1, R2 or R3 gets connected to the select line of Mux. For the settings u2 >= Threshold and u2 > Threshold the parameter **Threshold** is compared with u2 signal to produce the select value for connecting either input u1 or u3 to the output through the Mux. For the parameter set as u2 ~=0 the select logic is created by the relational operation with R3.

B.2.3 Math blocks



Abs

Ouput absolute value of input

Description

The Abs block outputs the absolute value of the input.

Data Type Support

The Switch accepts signals of the type

- Fixed point signed
- Fixed point unsigned

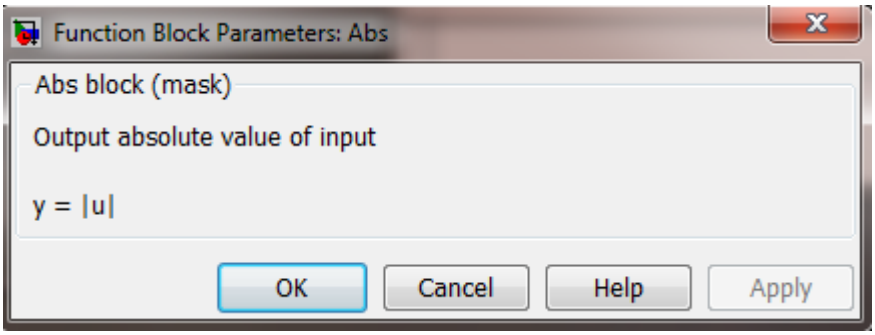
Output Data Type

- Fixed point signed

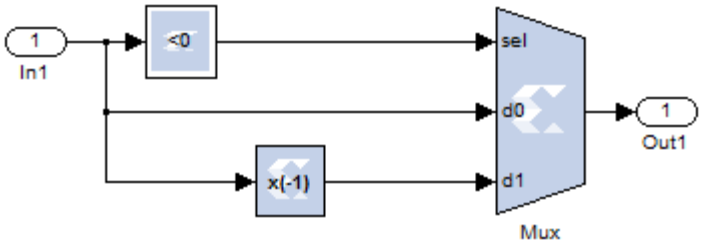
Latency: 0 CLK

Parameters and Dialog Box

The Abs block dialog box appears as follows:



Implementation



The input to the block is compared to zero to find the signal polarity. If the signal's polarity is positive, the input is propagated to the output unaltered. If not the signal's value is negated and passed to the output.

Average

Outputs a moving average of the input with a window span of 10.

Description

The averaging block is provides a moving window average for an input signal. The span of the window is 10 and thus this block has an inherent latency of 10 clock cycles. The user can also set the sampling rate of the averaging block to influence the data rate for 10 samples that are being averaged in a single window.

Data Type Support

The Switch accepts signals of the type

- Fixed point signed
- Fixed point unsigned

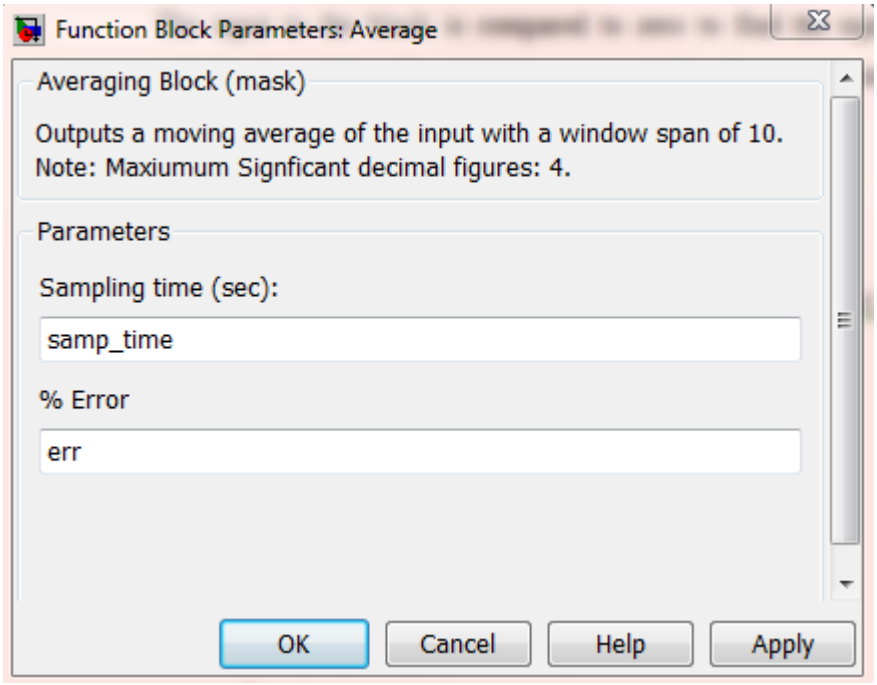
Output Data Type

- Fixed point signed

Latency: 10 CLK

Parameters and Dialog Box

The Averaging block dialog box appears as follows:



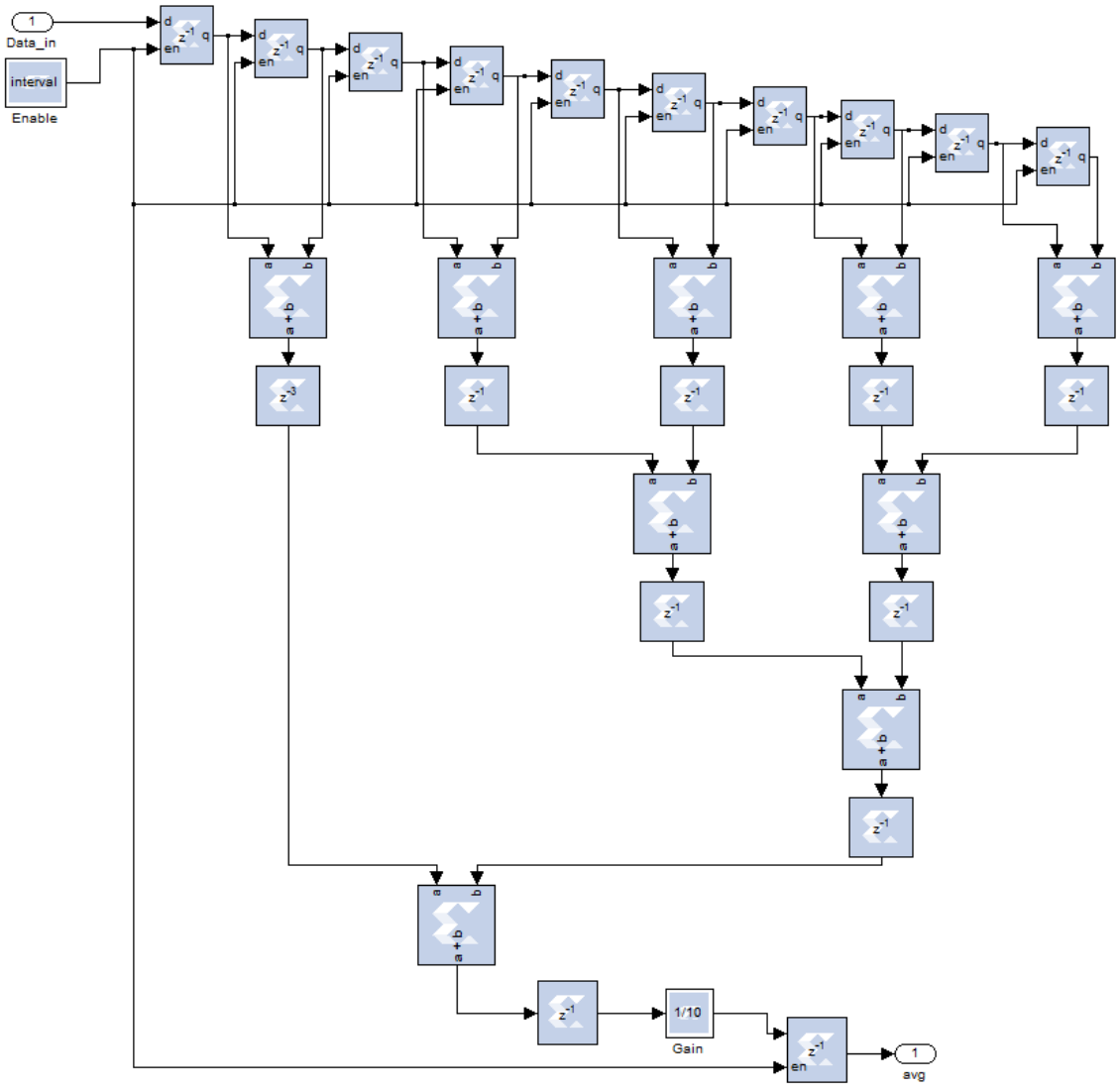
Sampling Time (sec)

Sets the sampling time of the averaging block. Every internal calculation gets updated at the time rate specified.

% error

The percentage of error in the output value when it is of type Fixed point. For the type Integer the error is zero.

Implementation

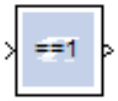


As seen in the picture above, ten samples of the input data are stored in registers connected in a cascaded fashion. These registers are enabled at a rate set by the user using the parameter **Sampling Time (sec)**. The register values are sequentially added and the final result is divided by a factor 10 to get a 10 window average result of the input data. Delay elements are added in suitable places to maintain data flow consistency and to match timings.

Compare to Constant

Compares an input u to the constant value. Output is of type 'bool'.

Description



The Compare To Constant block compares an input signal to a constant. Specify the constant in the Constant value parameter. Specify how the input is compared to the constant value with the Operator parameter. The Operator parameter can have the following values:

==: Determine whether the input is equal to the specified constant.
~=: Determine whether the input is not equal to the specified constant.
<: Determine whether the input is less than the specified constant.
<=: Determine whether the input is less than or equal to the specified constant.
>: Determine whether the input is greater than the specified constant.
>=: Determine whether the input is greater than or equal to the specified constant.
The output is 0 if the comparison is false, and 1 if it is true.

Data Type Support

The Compare To Constant accepts signals of the type

- Fixed point signed
- Fixed point unsigned

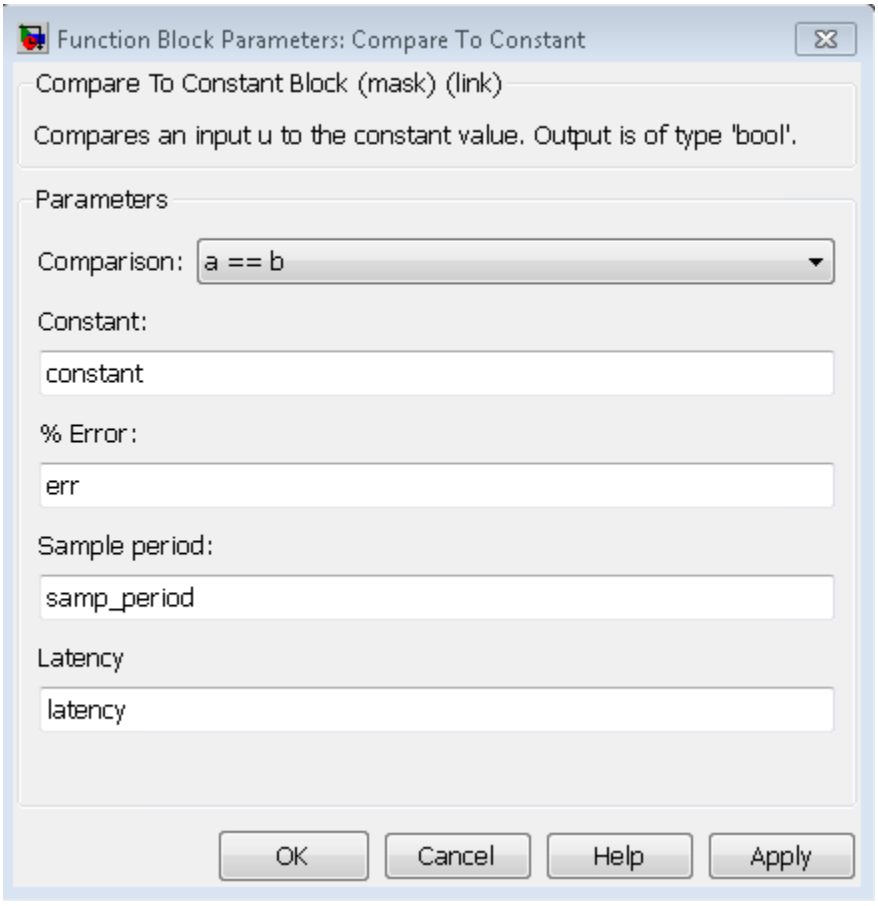
Output Data Type

- Boolean

Latency: 1 CLK

Parameters and Dialog Box

The Compare To Constant block dialog box appears as follows:



Comparison

Specify how the input is compared to the constant value

Constant

Specify the constant value to which the input is compared

% Error

The percentage of error in the constant value when it is of type Fixed point. For the type Integer the error is zero.

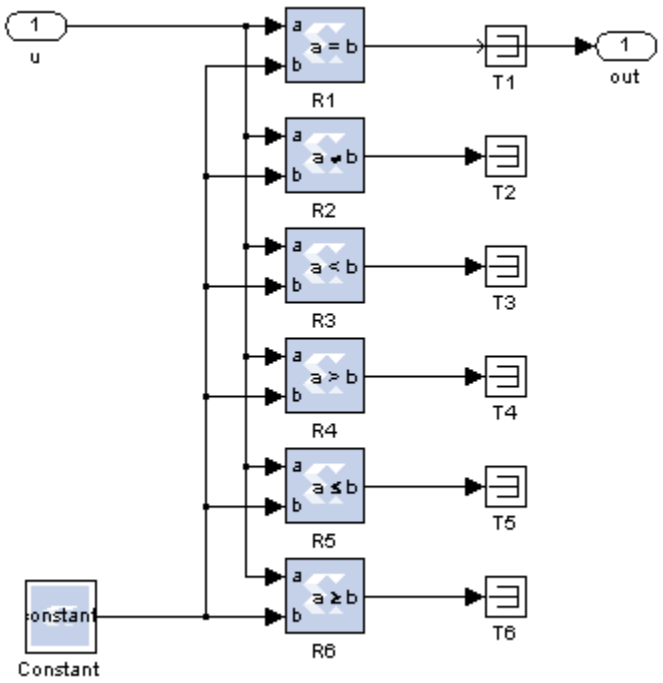
Sample Period

Sets the sampling rate of the internal blocks of the Compare To Constant subsystem.

Latency

Sets the latency of the internal blocks of the Compare To Constant subsystem.

Implementation

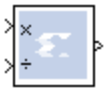


The block is implemented as shown above. The input signal u is connected to relational blocks R1:R6 which have various settings. The second input to the relational blocks is the constant to which the signal is compared. Through the mask and selection of a particular value in the field “Comparison”, the output of one of the six relational blocks gets connected to the outport labelled “out”. The rest of the blocks get connected to terminator blocks as shown in the figure.

Divide

Performs division of two numbers.

Description



The Divide block performs real number division of two fixed point numbers using the Cordic Divider.

Data Type Support

The Divde block accepts signals of the type

- Fixed point signed
- Fixed point unsigned

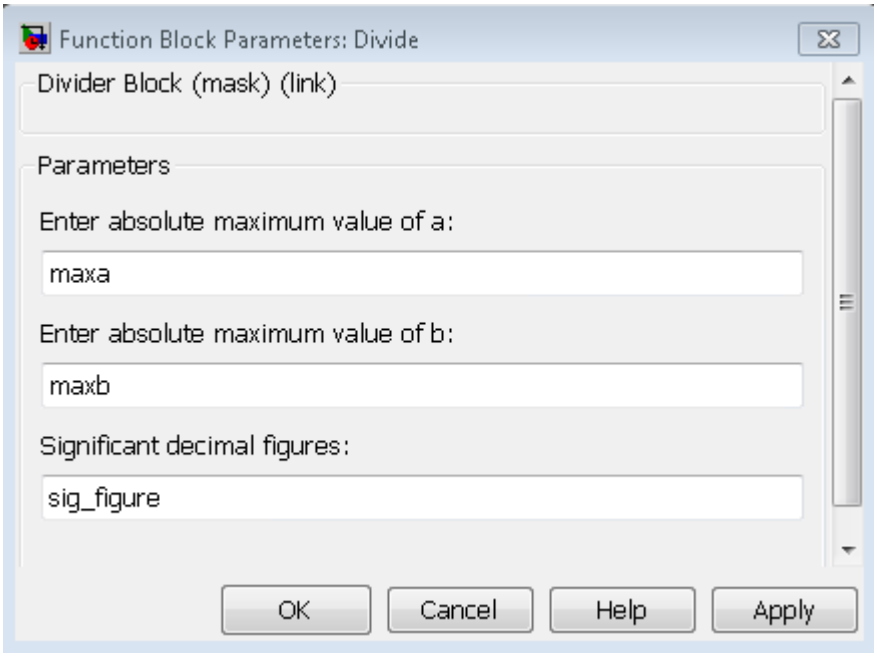
Output Data Type

- Fixed point signed
- Fixed point unsigned

Latency: 31 CLK

Parameters and Dialog Box

The Divide block dialog box appears as follows:



Enter absolute maximum value of a

Set the maximum possible value of input a

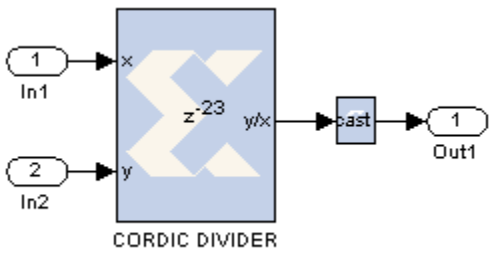
Enter absolute maximum value of b

Set the maximum possible value of input b

Significant decimal figures

Set the number of significant decimal figures of the output

Implementation

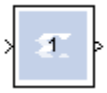


The block is implemented as shown above. The configuration of the Cordic Divider is automated through the masked subsystem and simple parameters are asked from the user.

Gain

Multiply input with a constant, K (y = K*u)

Description



The Gain block performs real number multiplication of a constant value with the user input. It utilizes the DSP48A1 slices that are present in the FPGA for multiplication.

Data Type Support

The Gain block accepts signals of the type

- Fixed point signed
- Fixed point unsigned

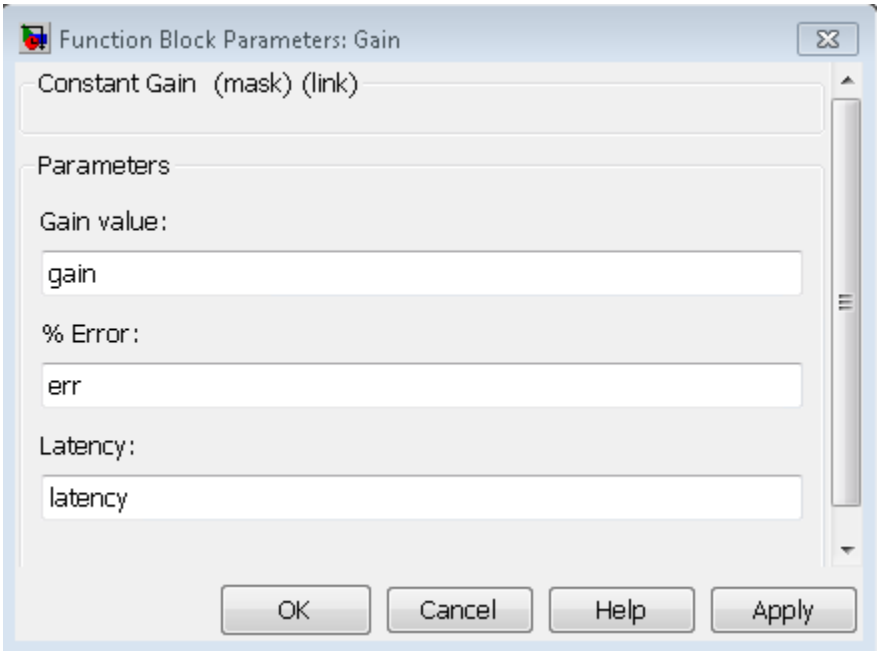
Output Data Type

- Fixed point signed
- Fixed point unsigned

Latency: 1 CLK

Parameters and Dialog Box

The Divide block dialog box appears as follows:



Gain

Specify the constant value with which the input is multiplied

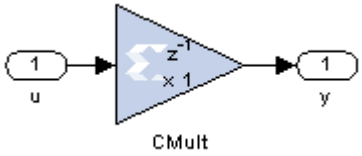
% Error

The percentage of error in the output value when it is of type Fixed point. For the type Integer the error is zero.

Latency

Sets the latency of the internal blocks of the Gain subsystem.

Implementation

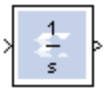


The block is implemented as shown above. The configuration of the CMult is automated through the masked subsystem and simple parameters are asked from the user.

Integrator

Discrete-time integration or accumulation of the input signal.

Description



Performs discrete-time integration of an input signal at a rate defined by the step time of the block.
Limitations: The value K*Ts should be at least 3 orders of magnitudes smaller than the maximum input signal.

Data Type Support
The Divde block accepts signals of the type

- Fixed point signed
- Fixed point unsigned

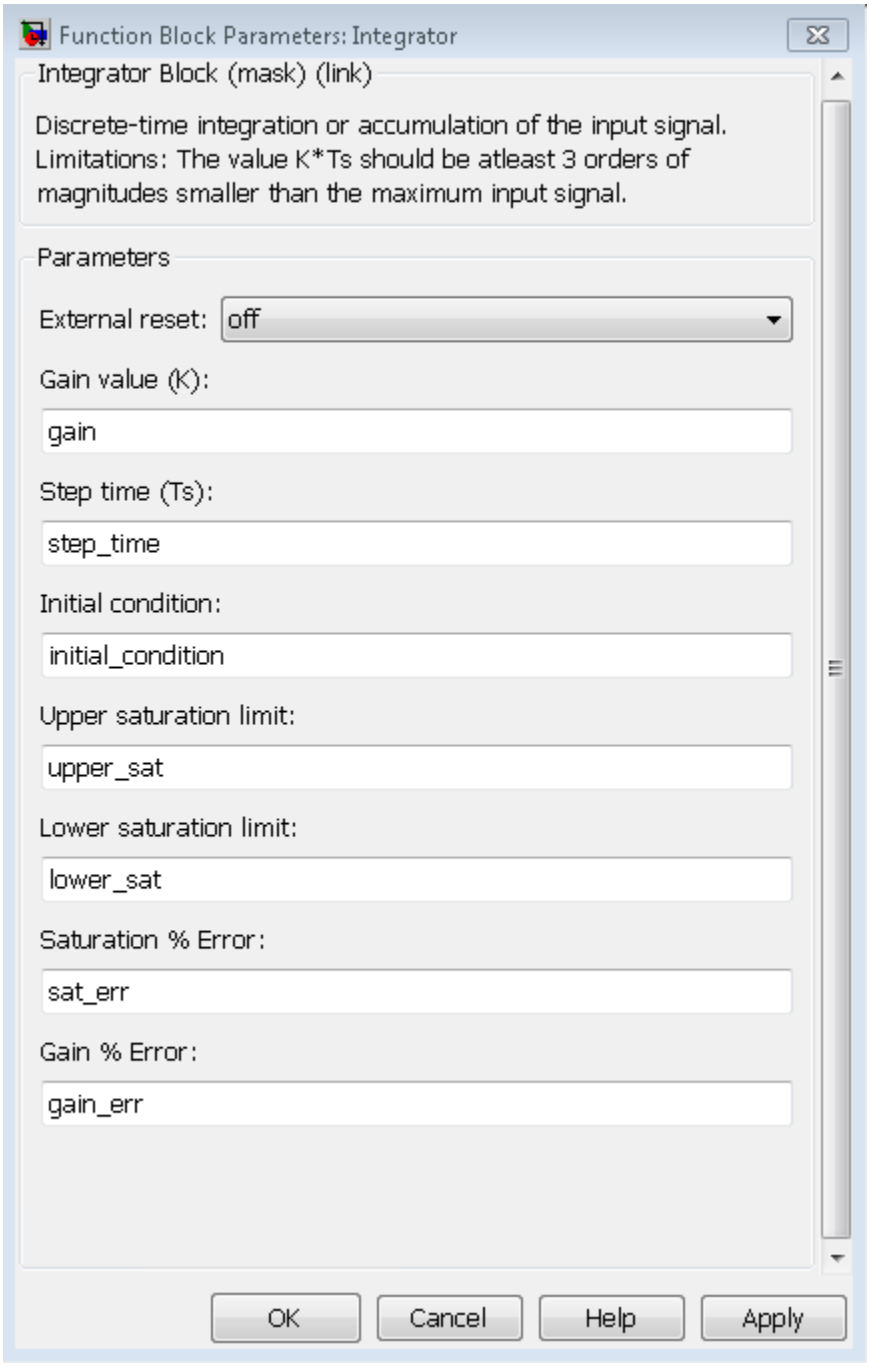
Output Data Type

- Fixed point signed
- Fixed point unsigned

Latency: 2 CLK

Parameters and Dialog Box

The Integrator block dialog box appears as follows:



Gain

Enter the gain of the integrator

Step time (Ts)

Enter the step time of the integrator

Initial Condition

Enter the initial value of the integrator at the start of operation

Upper saturation limit

Enter the upper saturation limit of the integrator

Lower saturation limit

Enter the lower saturation limit of the integrator

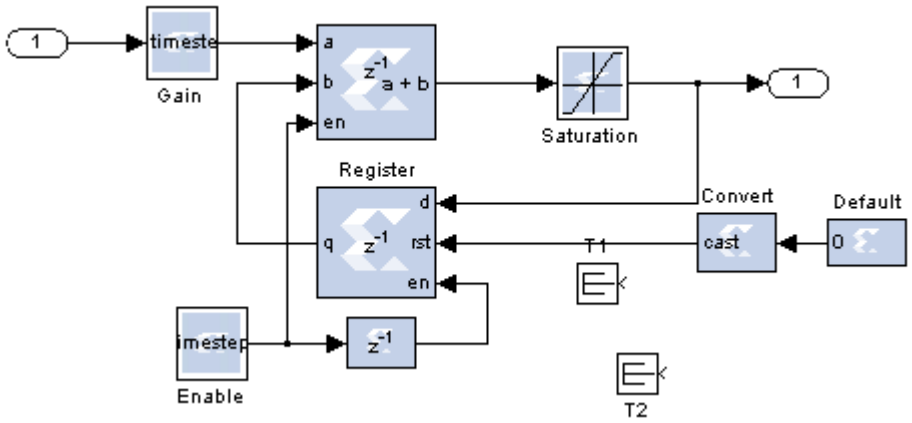
Saturation % Error

The percentage of error in the Saturation values when it is of type Fixed point. For the type Integer the error is zero.

Gain % Error

The percentage of error in the Gain value when it is of type Fixed point. For the type Integer the error is zero.

Implementation



The block is implemented as shown above. The input signal is multiplied with the time step value and accumulated at the rate of the time step. The output of the accumulator is limited by a Saturation block. An optional reset is also provided which clears the internal register.

PI Controller

This block implements discrete-time proportional and integral control using Forward Euler equation.

Description

Performs discrete-time PI control of an input signal at a rate defined by the step time of the block.

Limitations: The value K*Ts should be at least 3 orders of magnitudes smaller than the maximum input signal.

Data Type Support

The Divde block accepts signals of the type

- Fixed point signed
- Fixed point unsigned

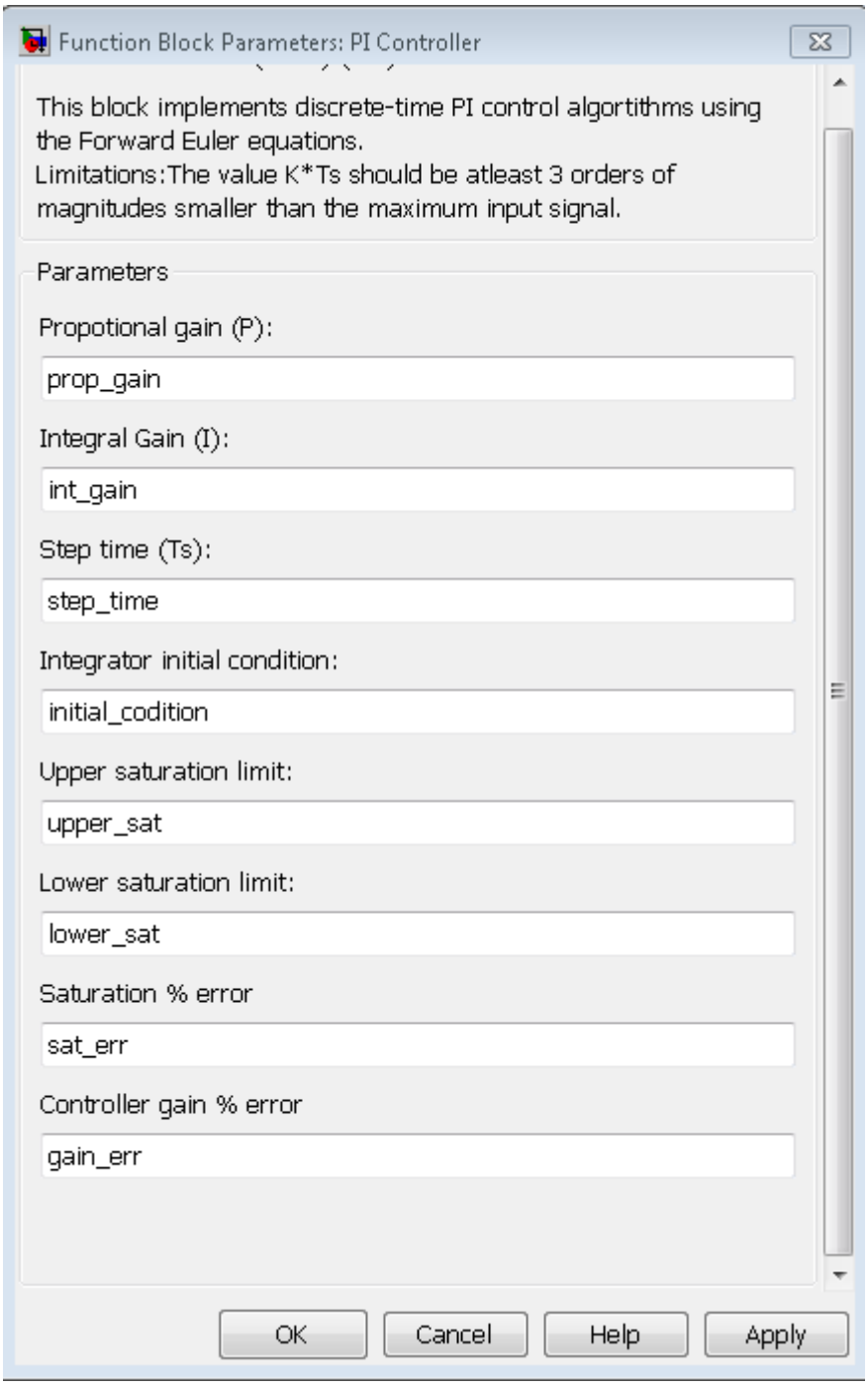
Output Data Type

- Fixed point signed
- Fixed point unsigned

Latency: 2 CLK

Parameters and Dialog Box

The PI block dialog box appears as follows:



Proportional Gain

Enter the gain of the proportional block

Integral Gain

Enter the gain of the integral block

Step time (Ts)

Enter the step time of the integrator

Initial Condition

Enter the initial value of the integrator at the start of operation

Upper saturation limit

Enter the upper saturation limit of the integrator

Lower saturation limit

Enter the lower saturation limit of the integrator

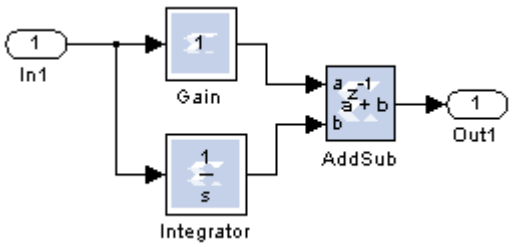
Saturation % Error

The percentage of error in the Saturation values when it is of type Fixed point. For the type Integer the error is zero.

Gain % Error

The percentage of error in the Gain value when it is of type Fixed point. For the type Integer the error is zero.

Implementation



The block is implemented as shown above. The input signal is multiplied with the time step value and accumulated at the rate of the time step in the integrator. The output of the integrator is added to the propotional gain constant and the resultant output is produced. An optional reset is also provided which clears the internal register.

Sine Function

Outputs a sinusoidal wave.

Description

Output a sine wave as per:

Output = Sin (Range*(0:No. of Samples-1)/(No. of Samples)+Phase)

Output sinusoidal waveform is in the range [-1 1].

Note: Input to the block is a normalized number between 0 and 1.

Data Type Support

The PI Controller block accepts signals of the type

- Fixed point unsigned in the range 0 to 1

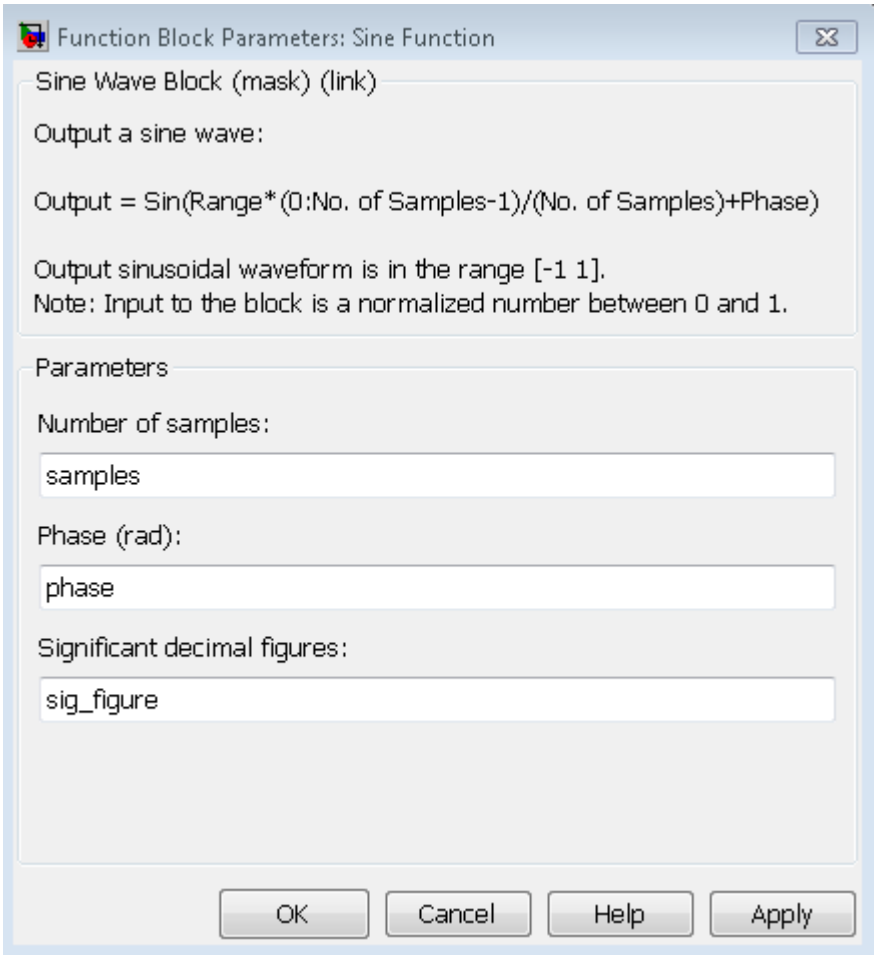
Output Data Type

- Fixed point signed

Latency: 2 CLK

Parameters and Dialog Box

The Sine Function block dialog box appears as follows:



Number of samples

Enter the number of samples of the sine wave output

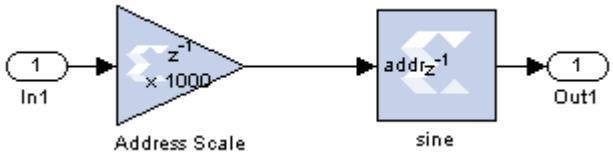
Phase

Enter the phase in radians of the sinusoidal output. Phase can be either leading or lagging.

Significant decimal figures

Set the number of significant decimal figures of the output

Implementation



The block is implemented as shown above. The input signal which is a value between 0 and 1 is scalled by the value in the field Number of samples and given as input to a Single Port ROM block.