DEVELOPMENT OF A HARDWARE IN THE LOOP TEST SETUP BASED ON SYSTEM ON CHIP BOARD

A MASTER'S THESIS

in

Electrical and Electronics Engineering

Atilim University

by

RABEI ABDULHAFID S. SHWEHDI

NOVEMBER 2017

DEVELOPMENT OF A HARDWARE IN THE LOOP TEST SETUP BASED ON SYSTEM ON CHIP BOARD

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

BY

RABEI ABDULHAFID S. SHWEHDI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

NOVEMBER 2017

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

_____

Prof. Dr. Ali Kara

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Dr. Kemal Efe Eseller

Head of Department

This is to certify that we have read the thesis "Development of a Hardware In the Loop test setup based on System On Chıp board" submitted by "Rabeı Abdulhafıd S. Shwehdı" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Asst. Prof. Dr. Kutluk Bilge Arikan	Asst.Prof.Dr Mehmet Efe Özbek
Co-Supervisor	Supervisor

Examining Committee Members

Asst.Prof.Dr Mehmet Efe Özbek	_____
Asst. Prof. Dr. Enver Çavuş	_____
Asst.Prof.Dr Yaser Dalveren	_____

Date: (23/11/2017)

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: RABEI SHWEHDI

Signature:

ABSTRACT

DEVELOPMENT OF A HARDWARE IN THE LOOP TEST SETUP BASED ON SYSTEM ON CHIP BOARD

Rabei Abdulhafid S. Shwehdi

M.S., Electrical and Electronics Department

Supervisor: Asst.Prof.Dr Mehmet Efe Özbek

Co-Supervisor: Asst. Prof. Dr. Kutluk Bilge Arikan

November 2017, 53 pages

This thesis proposed a hardware in the loop (HIL) test setup based on system on chip board (SoC) board, that can be used for testing a plant model developed using Simulink HDL coder and Xilinx System Generator (XSG) library. The plant model was converted to Verilog code using XSG system generator and uploaded to SoC board using Vivado Design Suite. The programmed SoC board was interfaced to real input signals through analog to digital converters (ADC) and the output signal of plant was carried to the measurement device through a digital to analog converter (DAC). Drivers, in the form XSG models, were developed for ADC and DAC modules which implement the communication protocols of the modules to achieve data transmission between these modules and the SoC board. The operation of the HIL setup has been verified for a simple plant model.

Keywords: HIL simulation, Model based design

ÖZ

YONGA ÜZERİ SİSTEM KARTI TABANLI DONANIMLA BENZETİM TEST DÜZENEĞİ GELİŞTİRİLMESİ

Rabei Abdulhafid S. Shwehdi

Yüksek Lisans, Elektrik ve Elektronik Bölümü

Tez Yöneticisi: Yrd.Doç.Dr Mehmet Efe Özbek

Ortak Tez Yöneticisi: Yrd.Doç. Dr. Kutluk Bilge Arikan

Kasım 2017, 53 sayfa

Bu tezde, bir yonga üzeri sistem (SoC) kartı kullanılarak, Simulink HDL coder ve XSG kütüphanesi ile geliştirilmiş bir tesis modelini sınamak için kullanılabilecek bir donanımla benzetim (HIL) test düzeneği geliştirilmiştir. Tesis modeli, Simulink HDL kodlayıcı ve Xilinx System Geneator (XSG) kütüphanesi kullanılarak Verilog koduna dönüştürmüş ve Vivado Design Suite kullanılarak ZedBoard'a yüklenmiştir. Programlanan SoC kartı, gerçek giriş sinyallerine analogdan dijitale dönüştürücüler (ADC) ile arabağlanmış, tesisin çıkış sinyali ölçüm cihazına dijitalden analoga dönüştürücü (DAC) aracılığıyla taşınmıştır.

Anahtar Kelimeler: HIL simülasyonu, modele dayalı tasarım

To My Parents and my Wife

ACKNOWLEDGMENTS

I express sincere appreciation to my supervisor Assoc.Prof.Dr. Mehmet Efe Özbek for his guidance and insight throughout the research. To my My Parents and Wife, I offer sincere thanks for thier continuous support and patience during this period.

TABLE OF CONTENT

LIST OF FIGURES

Chapter 1

1.1. Introduction

One of most important phase in designing and producing any product is the testing phase. However, testing is very expensive and time consuming and due to increasing competition between the companies in the rapidly growing industry an excessive time to market cannot be tolerated. The cost and the time are the key words to win the competition and most of the companies headed to develop techniques to make the designing and testing phase as short as they can and to keep the cost as low as possible. At the same time, these techniques must be reliable to ensure that the products are ready to release in the market, especially for products that may put consumers life's in danger if they have any manufacturing defects. Moreover, testing environment may not be suitable for humans and can involve safety risks for the personnel in sectors such as nuclear systems or space programs [1].

Simulating the electronic systems in order to test them in virtual environments has solved these problems, which inspired these companies to invest in developing computer simulation environments that mimic the behavior of the system under test. However, because of the rapid scientific and technological development, the systems have become more and more complex. The complex systems under test, simulation may not be accurate since it is not easy to create accurate mathematical models for complex components that can run fast during simulation. In that case, there is going to be difference between the system's real-time behavior and the system's simulated behavior. Besides, need for powerful and fast computers to do the simulation is another barrier facing the users of simulation systems since these computers are very costly.

Moreover, in industrial automation sector, after development phase the automation hardware and software need to tested before it is commissioned. A rigorous test requires that the automation hardware should be connected to the plant to be controlled. However, this is very difficult, if not impossible, since the plant may be in continuous use, under development, in procurement or may not be available to some other reason.

Hardware in loop (HIL) technique is one of solutions that was developed to solve the problems especially encountered in the test of complex real-time embedded systems.

Hardware-in-the-loop (HIL) simulation is a type of real-time simulation which we use to test our controller design. HIL simulation shows how the controller responds, in real time, to realistic virtual stimuli [2]. We can also use HIL to determine if the physical system (plant) model is valid to procced to next development steps. In a HIL simulation an embedded system which is running mathematical representation referred to as the "plant simulation" replaces the plant under control. Besides computing and control units, HIL hardware may include analog-to-digital and digital-to-analog converters through which the electrical signals of the sensors and actuators on the actual plant can be generated artificially.

Matlab (MATrix LABoratory) from Mathworks is one of leading simulation software that has been adopted the HIL simulation. It is widespread across all scientific disciplines mainly due to a simple syntax, stability and a wide range of applicability. There is also a lot of additional libraries and add- ons from various areas such as control and identification of systems, neural networks, fuzzy logic, statistics, symbolic math… The most important add- on for engineers is certainly Simulink, which allows modeling, simulation and analysis of dynamic systems.

However, even when the capabilities and the advantages of HIL technique and MATLAB are combined, the time needed to test and validate the plant under test is still considered not fast enough, since the computers that host the simulation software are sequential devices executing the algorithms one instruction at a time. In other words, to execute ten lines in the algorithm a certain number of clock periods should elapse, which can be a long time in matter of design and test of real time systems. The need for parallelization in execution of algorithms leaded to utilizing of the FPGA (Field Programmable Logic Array).

FPGA can be considered as parallel processing device in which a large portion of the algorithm can be executed in a few clock ticks. In addition, it is cheaper and reprogrammable which means that it can be used to design and test various plants without any change in hardware.

HDL CODER, is an add-on to MATLAB/Simulink which allows the system designers to create the systems prototypes using blocks from Simulink library, and then convert the designs to an HDL language to program the FPGA's. In this way the extensive

engineering effort for manual programming the FPGA in HDL is avoided and an easier design methodology is provided.

SOC is defined as a complex IC that integrates the major functional elements of a complete end-product into a single chip. In addition to FPGA fabric, a SOC incorporates a programmable processor, on-chip memory, and accelerating function units implemented in hardware. A system design that uses an SOC encompass a hardware component implemented using the FPGA resources and a software component which runs on the processor. Although an FPGA may provide a considerably higher processing performance compared to general purpose processors, implementation of a whole system on the FPGA is practical due the heavier design effort associated. Instead, the system is partitioned so as to assign the computationally intensive and iterative components to the FPGA and to leave the ones involving a less volume of data to the processor. A SoC platform makes this hardware-Software partitioning more feasible by proving the FPGA fabric and the processors on the same chip with high speed link between.

Nowadays, SOC's are integrated in development boards or evaluation kits, that contains all necessary component to perform the simulations in order to validate and verify the systems under test. Zynq is an SOC presented by Xilinx company which includes an FPGA and two ARM processors. ZedBoard, is low cost development board that contains a Zynq chip. Simplicity of communicating with the outside world and considerable MATLAB/Simulink support makes it one of the most popular development boards and a good choice for the verification and validation of moderately sized plant models.

The Xilinx System Generator for DSP (XSG) is a plug-in to Simulink that provides a high-level development environment for Xilinx FPGAs. A model based design methodology can be followed by letting XSG generate synthesizable HDL code from Simulink models. XSG also provides automatic generation of a HDL test bench, which enables design verification upon implementation. Simulink environment facilitates hardware software partitioning by allowing a mix of XSG blocks with other Simulink blocks in a system model. The components of the model which are intended to run the general-purpose processor are selected in an easy manner.

1.2. Motivation

This thesis aims to develop hardware in the loop (HIL) setup platform, the proposed test platform intended to be used in verification and validation of a Simulink model named (plant) which designed using HDL coder or XSG blocks. The developed plant has been converted to Verilog code using XGS system generator. The generated code then exported to Vivado design suite to generate bitstream file and program the host boar. The host board equipped with analog to digital converter ADC input and digital to analog DAC both are used to interface the host board to real word. The ADC to supply the plan on the board with real input signals, the DAC to deliver the output of plant to real word.

1.3. Layout of the Dissertation

Chapter 2 literature review, which introducing a brief review for some studies that have been done related to this thesis subject. Chapter 3 describes the software, board, components, instruments that used in this work. Chapter 4 gives a thorough explanation of the how the test setup was implemented. The connection, interfaces and the designed Simulink Xilinx system generator model are explained in detail in this chapter. Chapter 5 the conclusions and suggestions for future work.

Chapter 2

Literature review

This chapter discusses and reviews some works that studied HIL simulation with based on SOC boards to.

Muraspahic et.al. [3] present an industrial IT systematic approach for implementing the HIL simulation for active heave compensated (AHC) draw-works model. The AHC simulated on PC with PLC to control.

This work done by Mhiai and Andreescu [4] this work aims to develop a real-time HIL system in order to evaluate and test control strategies in thermal power plants. The proposes of this work are to reduce downtime in control system design stage in addition to determine which test platform is the best in term of accuracy, cost and efficiency of control system between those three systems. In order to choose the most suited cost-effective solution, the differences in control and process signals using HIL-HRT system versus HIL-SRT, and fully simulated system, have been compared and analyzed.

Anne-Laure et.al. [5] This paper proposes a HIL simulation of an innovative subway traction that uses supercapacitors as an energy source. This HIL simulation is an intermediary step before developing the full-scale system

The study done by Tessari and Fantuzzi [6] follows model-based methodology and utilizes an ordinary computer for HIL simulation instead of special HIL hardware. HIL is performed by running both the controller software and the plant model on the same PLC. Simulink models have been converted to Twin CAT C++ code modules.

Chaaban, Walid, et al [7] introduced an automated test tool design procedure, they used MATLAB Simulink environment to develop it. The tool designed to perform software verification while the software is running on PLC (Programmable Logic Array) or in other word HIL (Hardware- In- The- Loop) for control application developed in model based environment. The work aimed to checking the functional correctness of the automatically generated C++ code by real time workshop which designed and developed in Simulink on hardware target, moreover the proposed tool will compare the results that obtained from HIL with results of MIL (Model-In-The-Loop) that performed in first stages of development for the same application.

Zouari et.al. [8] proposed a hybrid solution in designing robot manipulator, by using hardware parts inside loop simulation HIL (Hardware In The Loop), aiming to reduce both the cost and development time and keeping the reliability in high level. They designed electrically driven robot manipulator controllers and their simulations, using DSP (Digital Signal Processing) builder library in MATLAB Simulink, then compared the obtained results with the designed HIL simulation results which implemented in FPGA board, in order to evaluate the HIL's reliability.

This work presented by, Yoon el.al. [9] this work utilizes HILS (Hardware-In-Loop-System) in evaluating of motor system drive of HEV (Hybrid Electrical Vehicle). The motor system for HEV has been developed basis on mathematical modeling using matlab Simulink.

This paper done by Sarikan and Aydemir. [10] in this paper a RTDS (Real-Time Digital Simulation) with HIL support for brushless DC motor approach has proposed. PC that equipped with appropriate hardware and software, has used to executed the developed framework. I/O board that attached to PCI SLOT, is used to connect the PC to analog and digital input and output. In addition, they created Simulink simulation models to simulate brushless DC motor in order to compare the results of HIL and Simulink simulation.

Selvamuthukumaran et.al. [11] in this work they aimed to develop a research plate form to reduce the time needed to prototyping of power electronic converters that used in solar photovoltaic (PV) applications. FPGA used to perform the HIL simulation of voltage source inverter VSI that used for PV system power converter. MATLAB Simulink/ Xilinx system generator used to generate the hardware description language VHDL code which used to program the FPGA.

Chapter 3

Development Environment and Tools

3.1. Hardware Tools

3.1.1. ZedBoard:

ZedBoard is a development kit, it contains most of the supporting function and interfaces necessary to develop and test a wide range of application. besides that, it may be connected to Pmods (peripheral module) to expand capability to interface with outside world. Figure 3.1 shows the ZedBoard

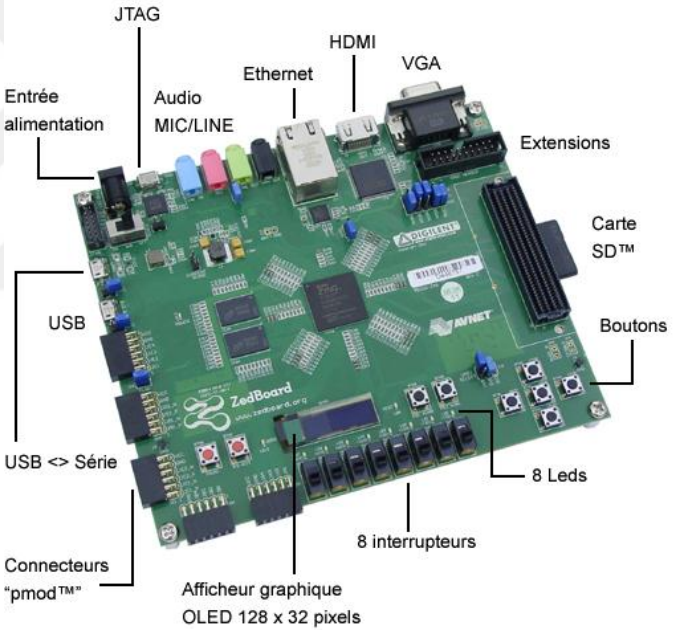

Figure 3.1 The ZedBoard

3.1.2. Pmod:

Pmod (Peripheral module) is an i/o interface boards, designed to extend the capabilities of microcontroller and FPGA boards.

Pmod is small size interface that comes with 6, 8 or 12 pins that connect it to targeted boards. Pmod modules use various communication protocols such as GPIO, I2C and SPI. In this work the used Pmod is interfaced to ZedBoard via SPI (Serial Peripheral Interface) [12].

3.1.2.1. SPI interfacing protocol:

SPI communication protocol is a simply interfacing method that developed by Motorola basically it uses four interfacing channels or pins to drive the communication between the Pmod and the host board. Generally, the host board is the master and the Pmod is the slave.

The four signals that used in this protocol are:

1. CS (Chip Select) sent by the host to activate the slave (Pmod).
2. Clock sent by host to be synchronized with the slave (Pmod).
3. MOSI (Master Out Slave In) data sent form the host to slave.
4. MISO (Master In Slave Out) data sent from slave to host.

The slave can be activated by driving the CS signal low and keeping this status until the number of pulses needed to finish the targeted operation has elapsed. Figure 3.2 shows the connections between SPI master and slave. In this thesis Pmod DA2 which uses this protocol has been used [13].

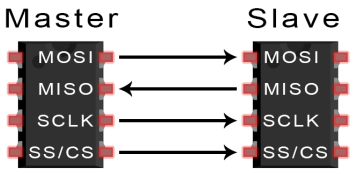

Figure 3.2 SPI master to slave interfacing lines

3.2. Pmod DA2:

Pmod DA2 is a 12-bit digital to analog converter that uses SPI interface protocol and can provide two analog output signals simultaneously. The maximum sensitivity that may achieved is about 1mV [14]. Figure 3.3 shows the Pmod DA2.

Figure 3.3 Pmod DA2

16-bit data word should to be sent to DA2. The first two bits are redundant. Bit 2 and 3 set the power down mode for the DA2 and are zeros in our application. Starting from the

fifth clock tic, 12-bit data which needs to be converted to analog is send with most significant bit first. Figure 3.4 shows Pmod da2 data stream.

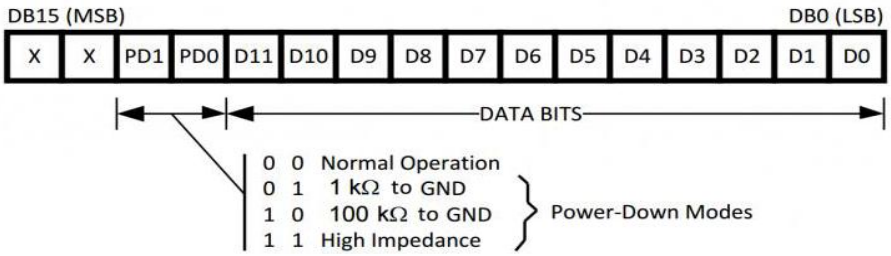

Figure 3.4 Pmod DA2 data stream

Since the DA2 doesn't has an input MISO and MOSI has designed to be an input. Figure3.5 shows the circuit diagram of DA2.

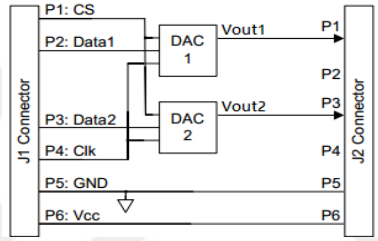

Figure 3.5 Pmod DA2 circuit diagram

The output voltage V_{out} of DA2 can be calculated as, $Vout = Vref * (\frac{decimal\ input}{resolution})$ (1)

where:

- $Vref$: is the reference voltage of DAC.
- decimal input: the input for DAC in decimal.
- $resolution$: is 2^{12}=4096.

In this study is V_{ref} is 3.334V.

3.2.1. Interfacing with Pmod DA2:

The Pmod is activated by selecting ~SYNC or chip select signal to low voltage state for 16 clock pulses meanwhile the 16-bit package [12-bit digital data that need to be converted to analog signal headed by four zeros] starts to send bit by bit on each clock pulse to the Pmod DA2. Figure 3.6 shows the timing diagram of analog to digital converter.

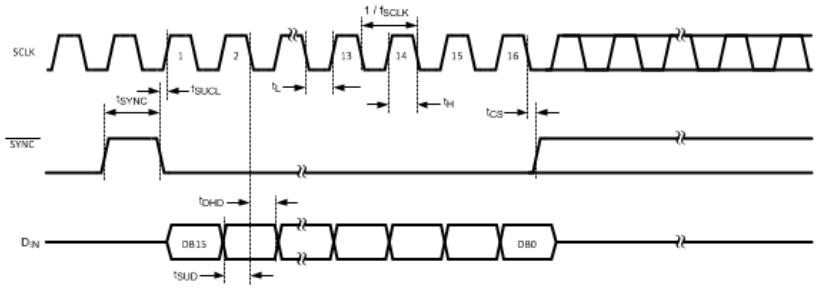

Figure 3.6 The timing diagram digital to analog converter

3.3. Pmod AD1

Pmod AD1 is 12-bit analog to digital converter that interfacing with host using SPI protocol. With sampling rate up to 1 million sample pre-seconds and has 6 pin connectors. Figure 3.7 shows the Pmod AD1[15].

Figure 3.7 Pmod AD1

Since the AD1 only sending data to the host board both MISO AND MOSI has made to work as an output. Figure 3.8 shows AD1 circuit diagram.

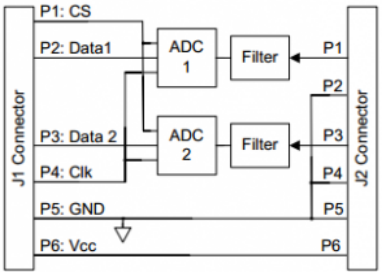

Figure 3.8 Pmod AD1 circuit diagram

3.3.1. Interfacing with Pmod AD1:

Figure 3.9 shows the timing diagram of digital to analog converter module AD1. The module could be activated by pulling the chip select signal CSEL to low state for 16 clock pulses. At the moment the CSEL signal goes to zero, analog input conversion will start, and the 12-bit data which represent the analog input voltage value will be

transmitted after four leading zeros, through the SDATA terminal. On the falling edge of the 16th clock tick the CSEL signal should rise again.

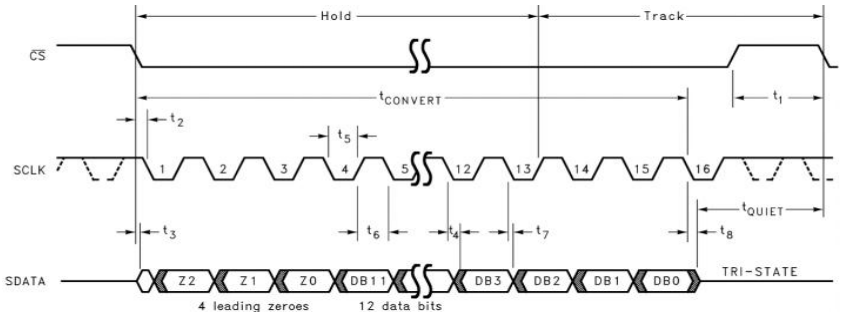

Figure 3.9 The timing diagram analog to digital converter

3.4. Software Tools

3.4.1. Xilinx System Generator:

Xilinx System Generator (XSG) is a system-level-modeling tool developed to simplify FPGA hardware design. The models which developed in XSG environment may easily compiled and converted to HDL code then uploaded to FPGA [16].

The Simulink XSG blocks that used to in this work are listed below then described.

- BlackBox.
- Parallel to Serial converter.
- BitBasher.
- Gateway In.
- Gateway Out.
- Counter.
- Adder.
- Gain.
- System generator.

3.4.1.1. BlackBox:

The BlackBox allows the use of Hardware Description Language in XSG system, both VHDL and Verilog code could be incorporated figure 3.10 [16].

Figure 3.10 BlackBox

3.4.1.2. Parallel to Serial converter:

This block takes the parallel input and splits it to N time-multiplexed output words where N is the ratio of number of input bits to output bits figure 3.11 [16].

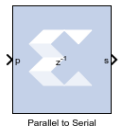

Figure 3.11 Parallel to serial converter

3.4.1.3. BitBasher:

This block allows to concatenation and slicing and augmentation the input attached to the block figure 3.12 [16].

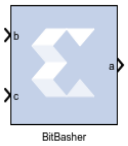

Figure 3.12 Bitbasher

3.4.1.4. Gateway In

This block is the input of Xilinx part of the Simulink design, it converts the data type from that used in Simulink into XSG fixed-point type figure 3.13 [16].

Figure 3.13 Gateway In

3.4.1.5. Gateway Out:

This block is the output from Xilinx part of the Simulink design, it converts the system generator data type into Simulink data type figure 3.14 [16].

Figure 3.14 Gateway Out

3.4.1.6. Counter

This XSG block can be selected to work as free or limited up or down counter figure 3.15 [16].

Figure 3.15 Counter

3.4.1.7. Adder

This block can be used as a fixed adder/subtractor or controlled dynamically to be adder or subtractor figure 3.16 [16].

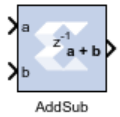

Figure 3.16 Adder

3.4.1.8. Gain

This block is a gain operator that multiply the input signal to the selected constant value figure 3.17 [16].

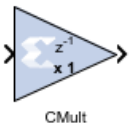

Figure 3.17 CMult

3.4.1.9. System generator:

This block could be considered as the control panel for both controlling system and simulation parameters, besides that it used calls the code generator to generate the HDL code of Simulink design. All Simulink models that consist of Xilinx Block-set must contain one System Generator block at least figure 3.18 [16].

Figure 3.18 System Generator

3.4.2. Simulink HDL coder

Simulink HDL coder is utilized to generate portable and synthesizable Verilog, in this work it used to generate the Verilog code of the both verification and plant models that designed using HDL coder blocks.

3.4.3. Vivado design suite

It produced by Xilinx, it superseded Xilinx ISE. It used to write the Pmods driver codes and to synthesis and analysis the Verilog codes that exported from Simulink XSG [17].

3.5. Supplies and measurement devices

3.5.1. Analog Discovery 2

Analog discovery 2 is a small size USB oscilloscope, has two digital oscilloscope channels, logic analyzer signal generator and variable power supply. it used as signal generator, power supply and oscilloscope to illustrate the inputs, outputs and the control signals during different stages of this study figure 3.19 shows the Analog discovery 2.

Figure 3.19 Analog discovery 2

Chapter 4

Implementation

In this chapter the implementation steps have been described. The main outlines of the implementation are given below:

- Development and verification of the DA2 driver.
- Integrating the DA2 driver with a plant model that is designed using Simulink HDL coder and XSG.
- Development and verification of the AD1 driver.
- Integrating the AD1 driver with the DA2 driver and verifying the AD1 driver operation experimentally.
- Integrating the both AD1 and DA2 drivers and a plant model to form a complete system ready to be used in HIL testing. Here the plant is modelled using two alternative approaches: In Simulink HDL coder and in XSG.

4.1. Dividing and redirecting the system clock

Sine the clock speed 30 MHz for DA2 and 20 MHz for AD1, the 100 MHz clock speed of the ZedBoard must be slowed down. For this purpose, only the Pmod driver BackBoxes has been tied to the ZedBoard clock, the rest of system are tied to divided clock in Simulink models as it shown in figure 4.1.

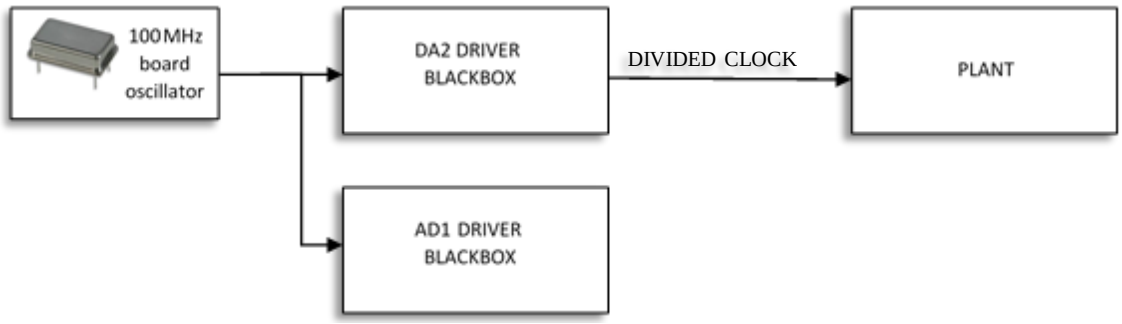

Figure 4.1 Dividing and redirecting the system clock diagram

After generating the IP Catalog using XSG system generator, the generated constraint can modify manually or in Vivado design suite. The idea is to select the board clock pin

which is Y9 to be the input clock for BlackBox of the Pmod drivers only. The board clock will be slowed by dividing it in Verilog code of DA2 and AD1 drivers that inside the BlackBox diver and sent out through pin AB6. The clock pin of the rest of system selected to be pin AB7. last step is to wiring the two pins AB6 and AB7. Figure 4.2 shows how to set the clock pins in Vivado design suite. The mentioned pins have been highlighted in yellow.

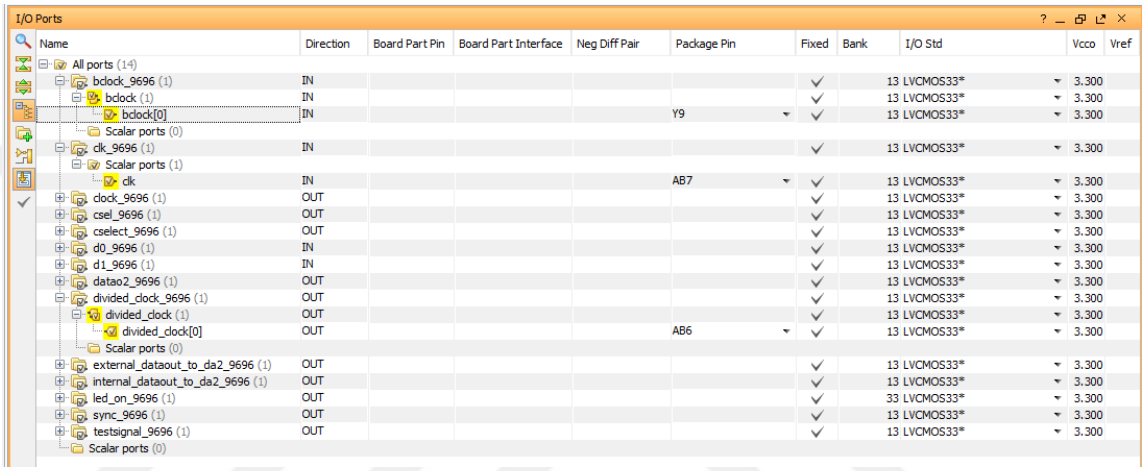

Figure 4.2 Setting the pins in Vivado

4.2. Driver Development for DA2 in Simulink Environment

The steps that followed in Driver Development for DA2:

- We have written a Verilog code to drive the Pmod DA2 and implementing it on the hardware board using Vivado design suite in order verify that it works properly.
- The Verilog code has been embedded in a Simulink XSG BlackBox block and BitBasher and serial-to-parallel blocks have been tied to its input.
- An input to DA2 driver is provided from the output of a limited-counter HDL coder block for testing purposes. The procedure was repeated also by using a XSG blocks for the limited-counter.
- The simulated output of limited counter and the waveform measured by the oscilloscope at the output of DA2 hardware, have been compared for verifying the operation of the DA2 driver.

Vivado design suite is the link between Simulink and the ZedBoard, where the models designed in Simulink are converted to HDL code by XSG. HDL code is then used in Vivado design suite to generate a bitstream file and program the ZedBoard. Figure 4.3 shows the described sequence.

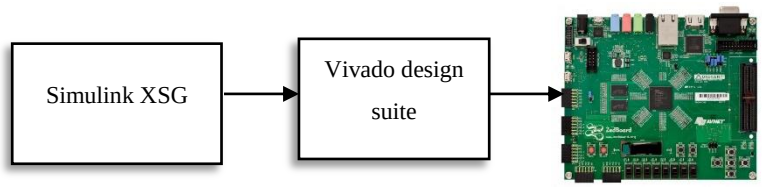

Figure 4.3 The development sequence of the driver of DA2

However, since the ZedBoard is faster than the maximum clock rate of the both AD1 and DA2 the clock has been slowed by dividing it. The Same case faced with PLANT BlackBox and the other XSG block that used in our verification model. Simulink blocks should be clock in which it operates as same frequency as the BlackBox The method which used has been described in previous section.

4.2.1. The Verilog code:

In order to interface the Pmod DA2 with ZedBoard a Verilog code has been written which feeds the data stream into DA2 using the protocol described in the above section. The operation of this Verilog code has been tested by programming the ZedBoard in Vivado 2016.4 environment. After the code validation step, the code imported to MATLAB XGS using the BlackBox in XGS.

In the next step, additional XSG blocks should be added to the DA2 driver model, in order to be able to interface the serial data to other possible XSG blocks. The black box should receive the incoming data through a BITBASHER block so as to concatenate additional bits to its output's least significant position. As mentioned before the digital data that needs to be sent to DA2 for conversion should be in 16-bit form. However, another zero should be added since one extra clock is needed to drive CS signal to high at the end of each conversion.

As mentioned in chapter 3, DA2 module starts the digital to analog conversion operation after the CSEL signal is changed from 1 to 0. In the Verilog code, a counter which

counts from 0 to 16 is used to control the status and timing of the signals that are sent to DA2.

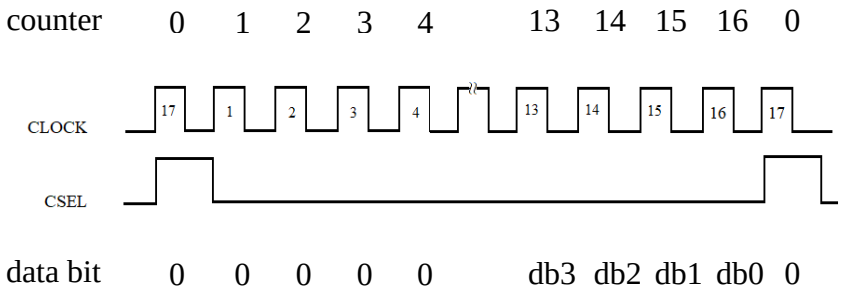

Figure 4.4 The timing diagram of the DA2 Verilog code

As shown in figure 4.4, when counter is zero, CSEL is 1. In the next clock period, CSEL goes low, activating the DA conversion and the first bit of the 16-bit stream is sent through DIN terminal. The 16-bit stream is headed by four zeros (as a required by DA2 communication protocol). Starting from the fifth clock tick, the transmission of 12-bit data to DA2 starts with the most significant first. After 16 clock ticks the transmission of the whole bit stream (the leading four zeros and the 12-bit data) to DA2 is complete. On the 17th clock period CSEL signal is set (to 1) to make DA2 convert the sent data into analog. Consequently, one clock period is reserved for setting the CSEL signal (to 1) and thereby signaling the DA2 that the bit-stream transmission is complete and conversion to analog should be performed now. After the 17^{th} clock period, the counter is reset and a new transmission-conversion cycle starts.

Two Verilog codes have been written to drive the DA2. In the first code is organized in multiple modules each doing specific job. This Verilog code is provided in APPENDIX A. However, when the code is imported to the XSG BlackBox, we found that the BlackBox accepts only the codes which consist of one single module. Therefore, another Verilog code has been written in which all instructions are written inside one module to satisfy the requirement of the BlackBox. This Verilog code is provided in APPENDIX B. The flow chart of second code is shown in figure 4.5.

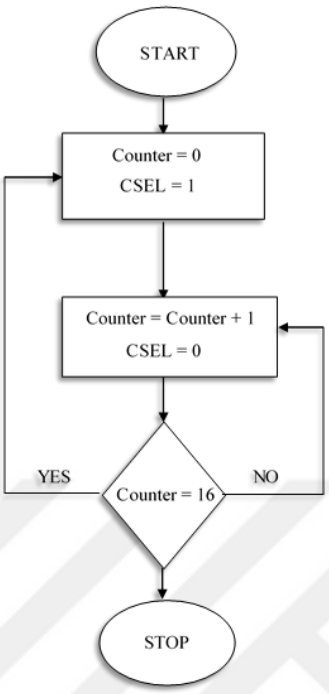

Figure 4.5 The flow chart of DA2 driver code

4.2.2. Addition of XSG Blocks to Complete the DA2 Driver

The BlackBox block which encapsulates the Verilog code described above, takes a 17-bit input in serial form. In order to match BlackBox block with other Simulink blocks, a parallel-to-serial block should be introduced in between. BlackBox takes a 17-bit word as the input which should be composed of 12 data bits provided by the plant model, and 5 leading bits which are all zeros. A BitBasher block is used to attach five zeros to the most significant position of the data word coming from the plant model. However, the BitBasher is needed only when the DA2 driver connected to simulated input. Figure 4.6 shows the model of Pmod driver.

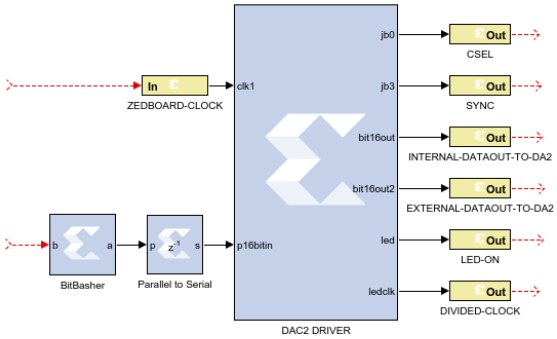

Figure 4.6 The Pmod driver

4.2.3. Verification of DA2 Driver Operation:

The Verilog code described in the previous section has been validated using Simulink XSG environment. A simple Simulink model consisting of a single limited counter block has been connected to the input of DA2 Driver block to find out whether the expected analog signal is observed at the output of the DA2 module. Two different methods have been used in this testing procedure: In the first method the limited counter has been represented by an HDL coder block and HDL code has been generated by HDL Coder. In the second method, limited counter has been represented using XSG blocks. Vivado design suite has been used to generate the bitstream file and program the ZedBoard.

The limited counter counts to 4000 and then resets itself. From equation 1

$$\left({}^{4000}/_{4095}\right) * 3.334v = 3.256v$$

Since 4000 is equivalent to a voltage of 3.256 V, we expect to observe a ramp signal with a peak value of 3.256 V at the DA2 module's output port. Figure 4.7 shows the block diagrams of HDL coder model.

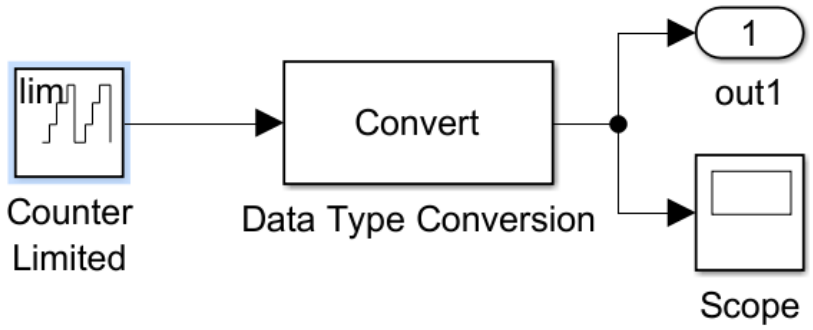

Figure 4.7 Limited counter model using HDL coder

The HDL Coder model and the XSG model are shown in figure 4.8 and 4.9 respectively.

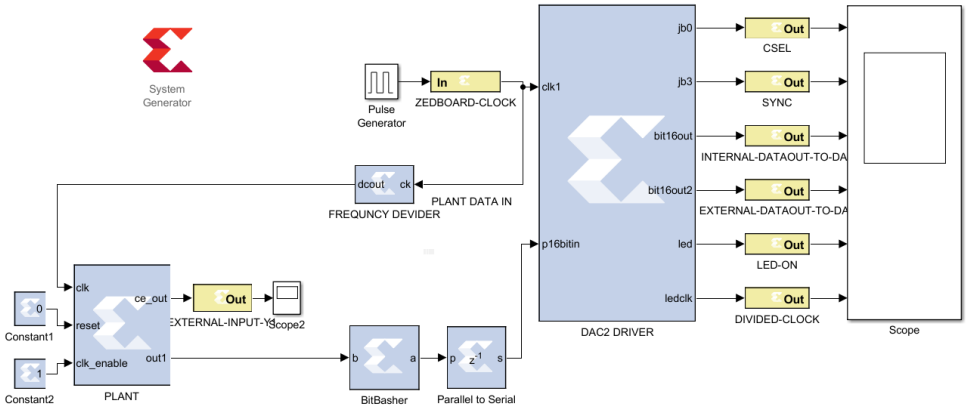

Figure 4.8 Verification model using HDL coder model inside the black box

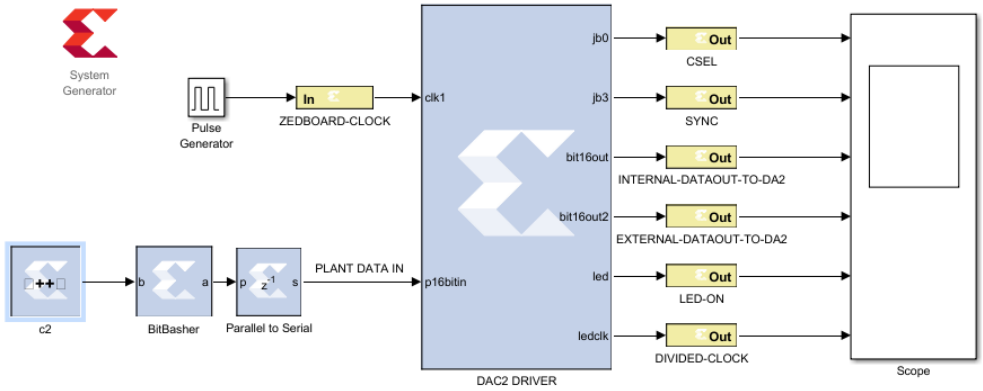

Figure 4.9 Verification model using XSG blocks

Figure 4.10 shows the simulated output of limited counter. Figures 4.11 and 4.12 show the real output signals measured at the ZedBoard output for the HDL model and the XSG model respectively. Comparison of the waveforms reveals that both of the measured waveforms match the simulated waveform. This verifies that the DA2 driver is working properly and we proceed to the next development steps.

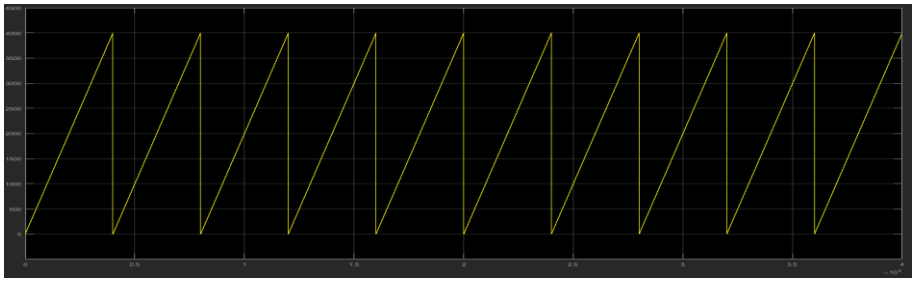

Figure 4.10 Simulink output

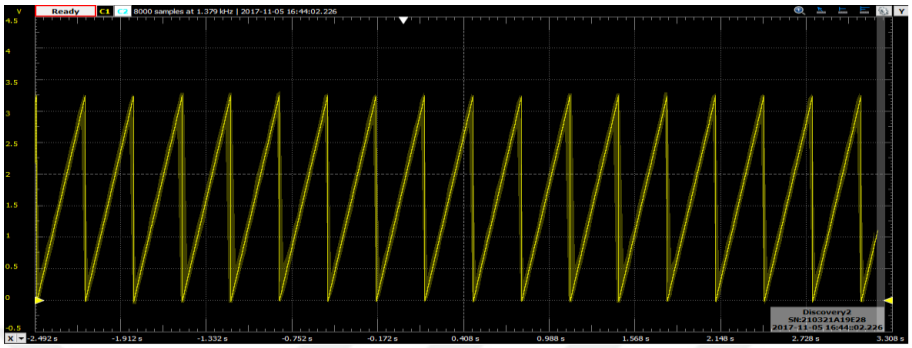

Figure 4.11 ZedBoard output of first approach

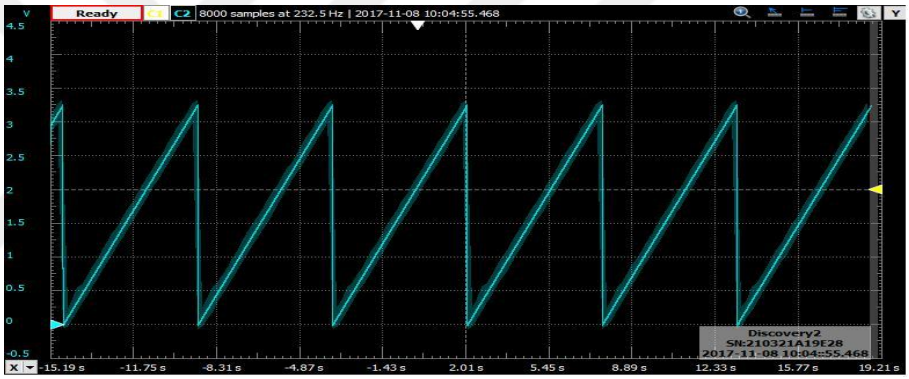

Figure 4.12 The output of second approach

4.3. The Simulink Model for the Plant to be Controlled:

Before we proceeded with the development of a driver for the ADC module, we wanted to integrate the DA2 driver that we have developed with a relatively complex Simulink model. Since we did not have a driver for connecting an ADC to a Simulink based system, this model should have an output but no input. The Simulink model that we have used for this purpose comprises two counter blocks, an adder block that adds the outputs of these counters, and a gain block with amplifies the sum signal. Counter-1 counts up to 800 (which corresponds to .651 V) and is then reset to zero. Counter 2 counts up to 1200 (which corresponds to .976 V) and then is also reset. The output of the adder is multiplied with a gain of 1.8.

We have followed two alternative paths in the development. In the first HDL coder blocks have been used in modelling the plant. Then we have implemented the same system with XSG blocks so as to compare it to the first method.

4.3.1. Modelling the Plant using HDL Coder

All blocks in this model are set to be in 12-bit resolution to match the input requirement of DA2 driver. In this case, as mentioned before, the block design has been converted to Verilog code using the HDL code generator. The Verilog code is embedded in a black box block and implemented using XSG as described in the verification section. Figure 4.13 shows the plant created using HDL block.

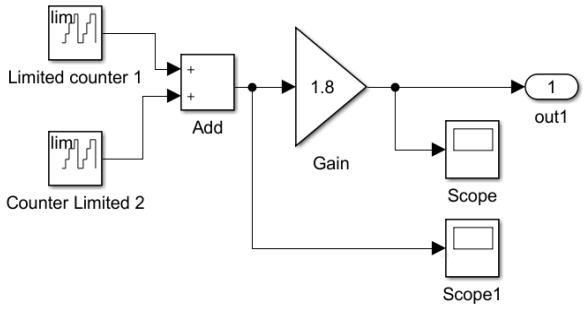

Figure 4.13 Show the block diagram of the plant.

4.3.2. Modelling the Plant using XSG

In this model the plant has been designed using XSG block only, which means that the designed model could be connected directly to DA2 driver without any code generation procedure. Figure 4.14 shows the plant created using XSG block.

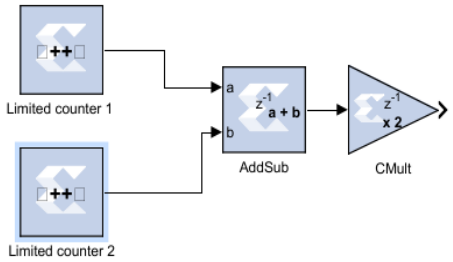

Figure 4.14 Limited adder with gain XSG block design

4.3.3. Integration of the DA2 Driver with a Plant Model:

Models have been attached to DA2 driver BlackBox in order to compare the output of simulation system with the output of HIL system. Also, the results of HIL systems that

were developed in HDL coder and XSG environment will be compared. Figure 4.15 and 4.16 show the HDL coder model design and XSG model design respectively.

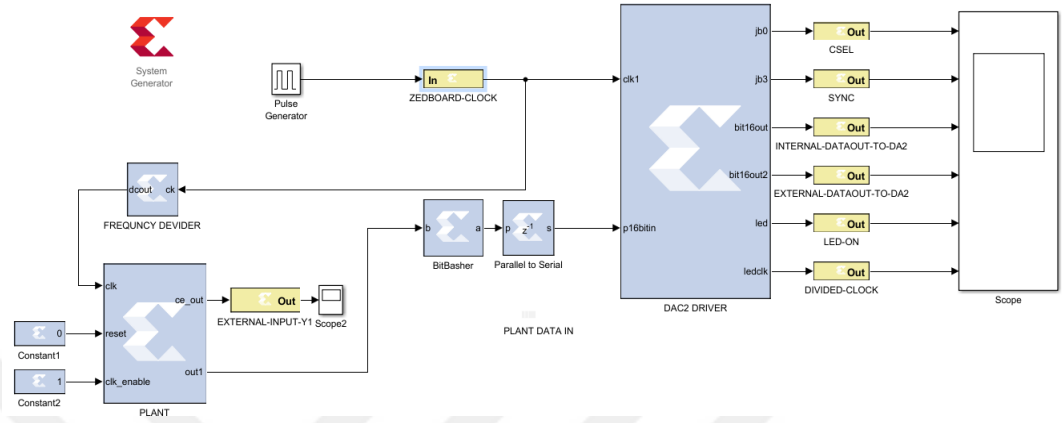

Figure 4.15 Verification model with a BlackBox containing the code generated with the HDL Coder

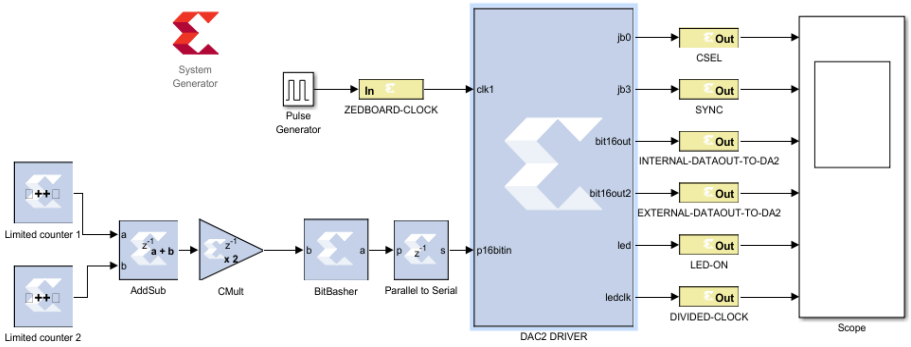

Figure 4.16 Verification model using XSG blocks

The simulation output signals of developed plant have been compared with the real output signals of ZedBoard. Figure 4.17 shows the output of the simulation output of the plant.

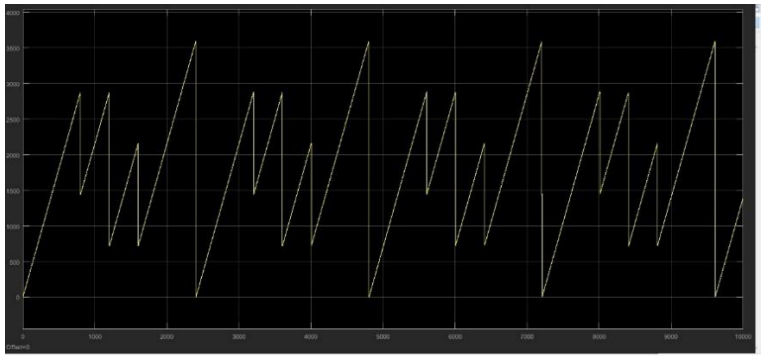

Figure 4.17 Simulation output of the plant.

Figures 4.18 and 4.19 show the signals measured at the output of the plants modeled in HDL coder and XSG respectively.

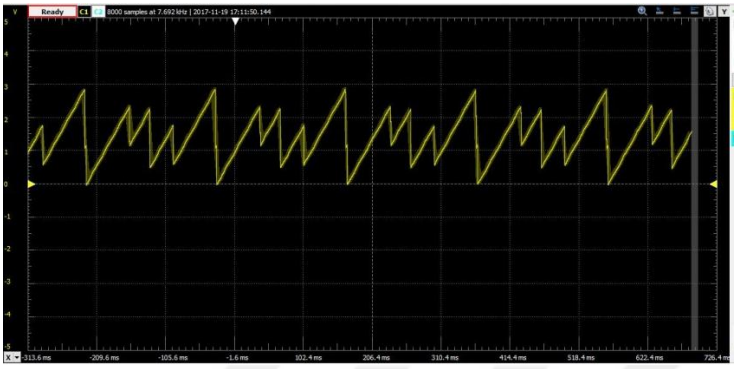

Figure 4.18 The output of plant modeled in HDL coder

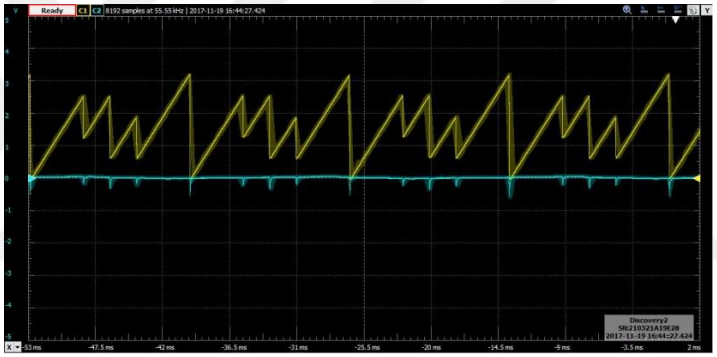

Figure 4.19 Output of plant modeled in XSG

Comparison shows that signals recorded at the output of DA2 module, closely match the simulation output.

4.4. Driver Development for AD1 in Simulink Environment

The steps that are followed in driver development for AD1 are as follows:

- We have written a Verilog code to drive the Pmod AD1 module and implemented it on the hardware board using Vivado design suite, in order verify that it works properly.
- The Verilog code has been embedded in a Simulink XSG BlackBox block that is tied to the input of DA2 driver.
- A voltage waveform has been applied to the input terminal of the AD1 driver module using a signal generator.

- The output of DA2 hardware module has been compared with the input signal of the AD1 to verify the operation of the HIL system.

4.4.1. The Verilog code:

As mentioned in chapter 3, AD1 module starts the analog to digital conversion operation after the CSEL signal is changed from 1 to 0. In the Verilog code, a counter which counts from 0 to 16 is used to control the status and timing of the signals that are sent to AD1.

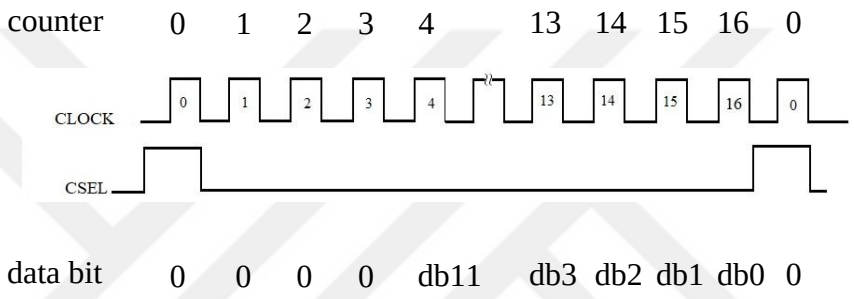

Figure 4.20 The timing diagram of AD1 driver Verilog code

As shown in figure 4.20, when counter is zero, CSEL is 1. With the falling edge of clock, analog to digital conversion is activated and the first bit of the 16-bit stream is sent out from the Pmod to the host board through D0 and D1 terminals. The 16-bit stream is headed by four zeros. Starting from the fifth clock tick, the transmission of 12-bit data from AD1 starts with the most significant first. After 16 clock ticks the transmission of the whole bit stream (the leading four zeros and the 12-bit data) to the host board is complete. On the 17th clock period CSEL signal is set (to 1) to prepare AD1 for new conversion. After the 17^{th} clock period, the counter is reset and a new transmission-conversion cycle starts. Figure 4.21 shows the flow chart of AD1 driver Verilog code. The is given in Appendix 3.

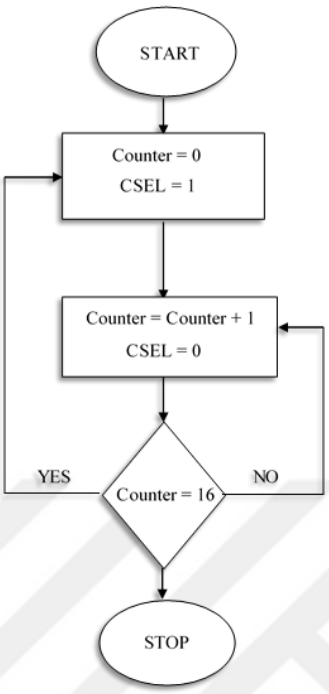

Figure 4.21 The flow chart of AD1 driver code

4.4.2. Addition of XSG Blocks to Complete the AD1 Driver

The out of AD1 driver module is parallel. However, the input of the developed plant should be in serial form. For this reason, the output of AD1driver module should be connected to parallel to serial converter as shown in figure 4.22.

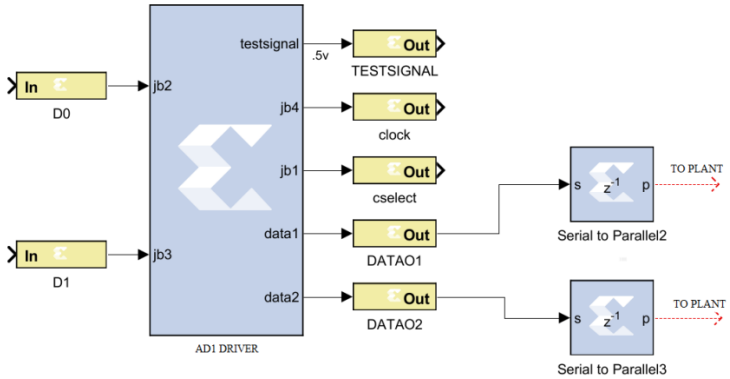

Figure 4.22 The AD1 driver block

4.4.3. Verification of AD1 Driver Operation

The AD1 driver has been verified by integrating it to Pmod DA2 driver in Simulink XSG environment. The model has been converted to Verilog code using system

generator. Vivado design suite has been used to generate the bit stream file and program the board. The signal measured at the output of the DA2 hardware module has been compared to the input signal which was applied to AD1 hardware module's input. Figure 4.23 shows AD1 Driver verification model.

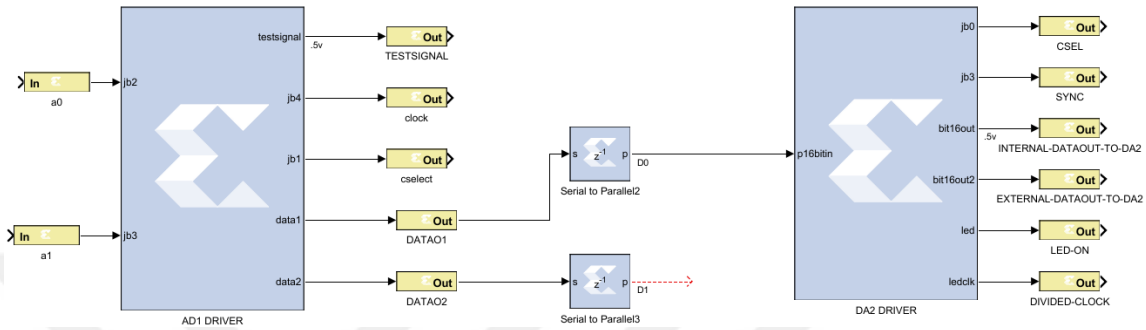

Figure 4.23 AD1 Driver verification model

4.5. Integration of DA2 and AD1 Drivers with the Plant Model

This is the final step in the development of the test platform. After verifying that the developed Pmod drivers can be used for interfacing the ZedBoard with real signals, the plant has been integrated with the AD1 driver and the DA2 driver. The structure of the HIL system is illustrated in figure 4.24. In this structure, AD1 module converts the two input waveforms into digital and the digital data representing the input signals is conveyed the Simulink model of the plant with the aid of the AD1 driver. Here the input undergoes some digital processing that represents the plant dynamics. The output generated by the plant model is transferred to the DA2 hardware module and converted in to analog with the aid of the DA2 driver.

For this particular HIL system the plant is a simple set of Simulink blocks in which an adder block adds the two inputs and multiplies the sum with a gain of two. However, the work flow that has been followed so far is applicable for any plant - no matter how complex - that can be represented by an appropriate Simulink model.

In the development of the Simulink model representing the plant, two alternative approaches have been used. The plant model has been developed using the HDL Coder approach first and then with XSG blocks only. Figure 4.24 shows the HIL system block diagram.

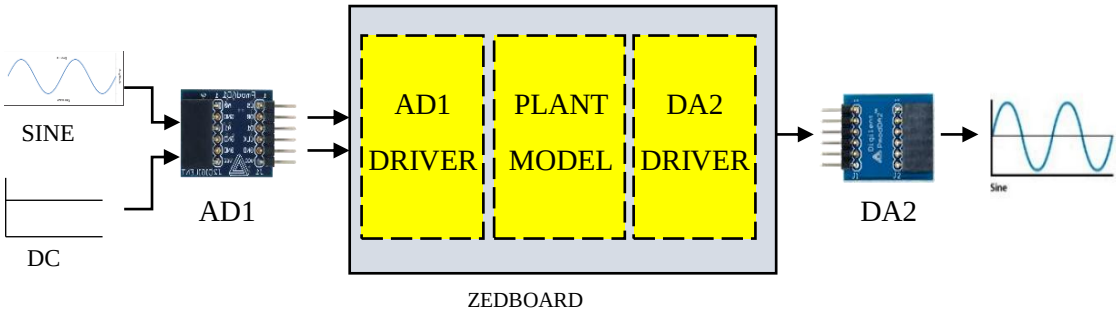

Figure 4.24 HIL system block diagram

4.5.1. Modelling the Plant using HDL coder blocks

Figure 4.25 shows the plant model where the adder block is a blackbox encapsulating the Verilog code generated by the HDL coder.

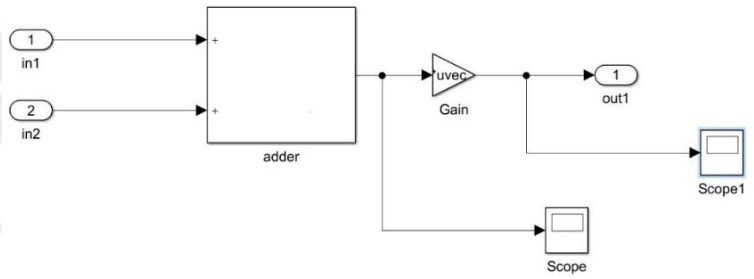

Figure 4.25 Plant model generated using HDL coder blocks

Figure 4.26 shows the block diagram of the whole HIL system in which the plant model generated with HDL Coder has been integrated with AD1 driver and DA2 driver models.

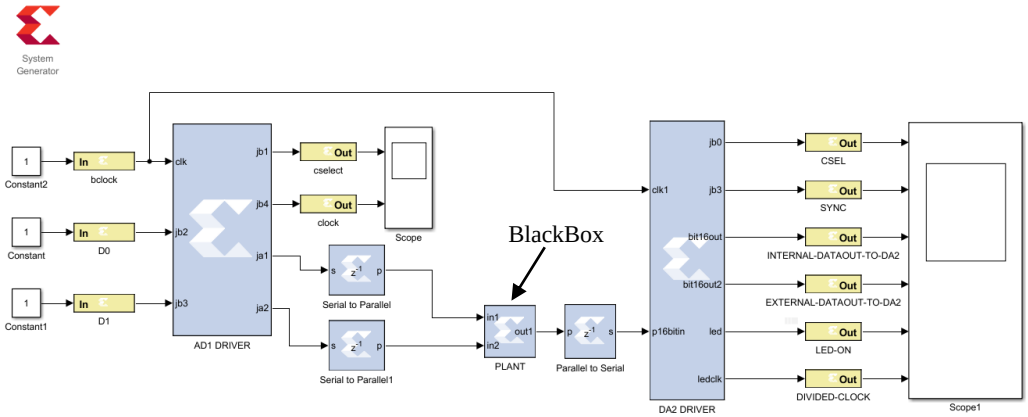

Figure 4.26 HIL system where the plant model is generated with the aid of HDL Coder

4.5.2. Modelling the Plant using XSG blocks

Figure 4.27 shows the plant model developed using XSG blocks only.

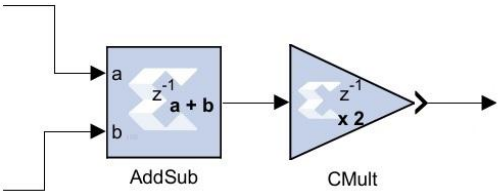

Figure 4.27 The plant using XSG blocks

Figure 4.28 shows the block diagram of the HIL system comprising the XSG model for the plant and XSG models for AD1 driver and DA2 driver.

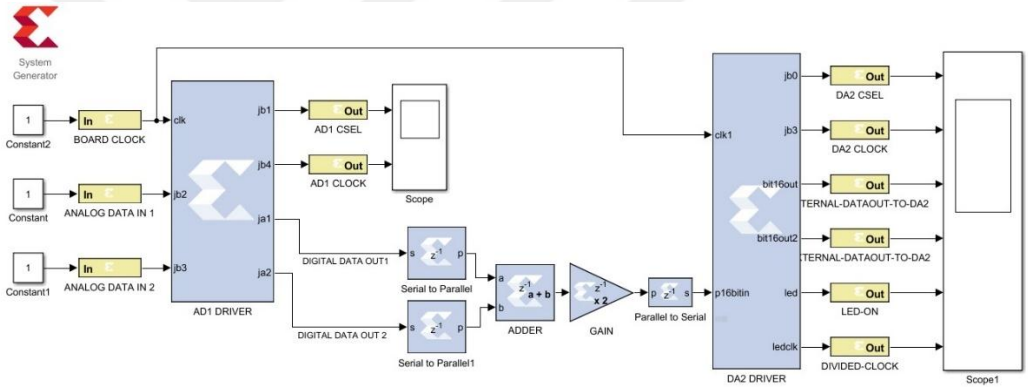

Figure 4.28 HIL System developed using XSG blocks

4.5.3. Verification of the HIL System Operation

In order to verify the operation of the HIL system a voltage waveform has been applied to the input terminal of the AD1 driver module using a signal generator. Both the input signal of the AD1 and the signal at the output of DA2 hardware module have been recorded by an oscilloscope. The recorded waveforms have been compared to the waveforms obtained in the simulation of the plant.

The inputs waveforms to the HIL system listed below. Note that since the DAC and ADC modules within the system cannot handle negative voltages, the inputs have been chosen carefully so as to avoid negative signal values both at the inputs and the output.

- ANALOG DATA IN 1: A sine wave with a peak-to-peak value of 0.5V and an offset of 5V.
- ANALOG DATA IN 2: A DC voltage of 0.5V.

Figure 4.29 shows the waveforms obtained in simulation. As expected, the output waveform is simply obtained by adding the input waveforms and multiplying the sum with two. The output waveform of figure 4.29 can be used as a basis for assessing the validity of the HIL system output that will be recorded with the oscilloscope.

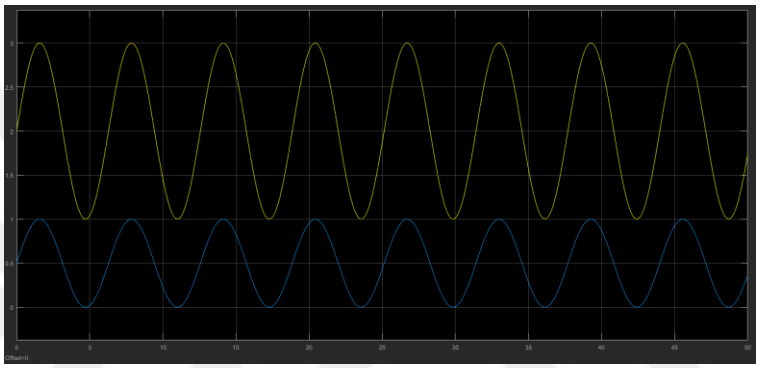

Figure 4.29 Plant simulation results: Output→Yellow, Input1→Blue, Input2→Not shown

The signal waveforms measured at the HIL system analog ports are shown in figures 4.30 and figure 4.31. The waveforms in figure 4.30 correspond to the case where the plant model was developed using HDL coder. The waveforms in figure 4.31 correspond to the plant model comprising solely XSG blocks.

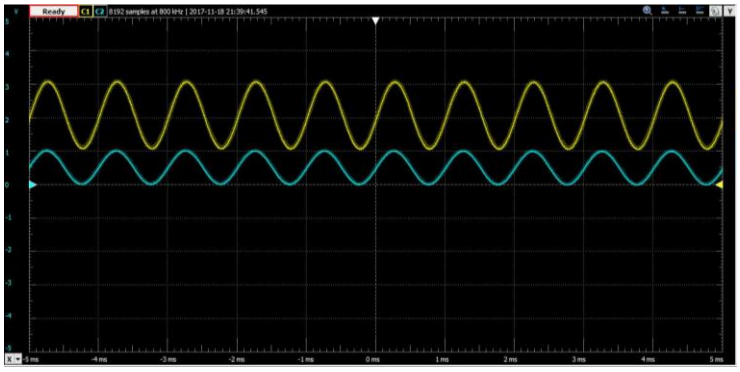

Figure 4.30 The output of HIL system where the plant is designed using HDL coder blocks. Output→Yellow, Input1→Blue, Input2→Not shown

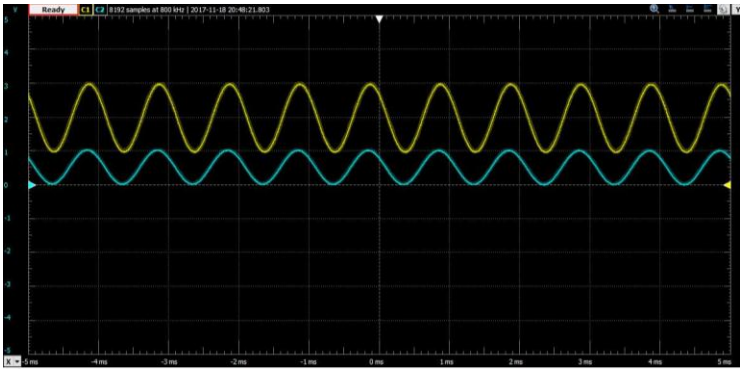

Figure 4.31 The output of HIL system where the plant is designed using XSG blocks.

Output→Yellow, Input1→Blue, Input2→Not shown

The first observation is the exact matching of figures 4.30 and figure 4.31 which shows that the HIL system produces the same results regardless of whether the plant is modeled using XSG blocks or HDL Coder blocks.

The second and the concluding observation is the close similarity of measured waveforms to the waveforms in simulation results. This shows that the HIL system can emulate the plant with high precision, producing the same results as obtained in simulation.

Chapter 5

Discussion of the Results and Conclusion

5.1. Conclusion

This study aimed to develop a HIL test platform based on a system on chip board. The proposed test platform is intended to be used for testing controller designs, when the actual plant is unreachable but a realistic assessment of the controller performance is highly desirable. The system on chip board utilized in this study has provided a high performance, light-weight and affordable platform for implementation of the HIL setup. Commercial-of-the-shelf DAC and ADC modules have been used for interfacing the analog signals to the board. In order to interface these modules to the MATLAB Simulink's user friendly high-level model development environment, drivers have been developed using both XSG and HDL Coder tools.

The main conclusion of this thesis is that based on the results of the previous chapter, the developed HIL platform has proved to be reliable and the results obtained were realistic.

Using Simulink HDL coder and XSG reduced the time needed to design and develop models for the plant under control. The possibility and ease of modeling a plant in MATLAB Simulink environment is only limited by the availability of suitable HDL coder and XSG blocks that will be used to construct the model. Unfortunately, HDL coder and XSG support only a limited subset of Simulink blocks making some complex Simulink models inappropriate for a HIL setup development work flow similar to the one followed in this study. While it is possible to model a system only with primitive and relatively simple Simulink blocks supported by HDL Coder or XSG, this partly eliminates the rapid modeling advantage offered by MATLAB Simulink environment.

5.2. Future work

As a future work we recommend developing the test platform fully using in HDL coder environment, to take advantage of ability of using most of Simulink blocks library and the ARM processors integrated in ZedBoard. The extent of performance advantages that

could be offered by the system on chip board could be explored by developing HIL platforms for complex plants operating at high sampling rates.

References

[1] Acevedo, Miguel. "FPGA-Based Hardware-In-the-Loop Co-Simulator Platform for SystemModeler." (2016).

[2] https://www.mathworks.com/help/physmod/simscape/ug/what-is-hardware-in-the-loop-simulation.html\

[3] “Hardware-in-the-Loop Simulation of an Active Heave Compensated Draw works” Sanin Muraspahic, LawkFarji, Michael Rygaard Hansen, Geir Hovland, Yousef Iskenderun and Hamid Reza Karimi)

[4] “Real-Time Hardware-in-the-Loop Test Platform for Thermal Power Plant Control Systems”, Mhiai and Gheorhhe-Daniel Andreescu

[5] "Energy storage system with supercapacitor for an innovative subway." Allègre, A. L., Bouscayrol, A., Delarue, P., Barrade, P., Chattot, E., & El-Fassi, S. (2010). IEEE Transactions on Industrial electronics, 57(12), 4001-4012.

[6] Tessari, Rita, and Cesare Fantuzzi. "Design of a packaging machine and virtual commissioning via modular hardware-in-the-loop simulations." Electrotechnical Conference (MELECON), 2016 18th Mediterranean. IEEE, 2016.

[7] Chaaban, Walid, et al. "A partially automated HiL test environment for model-based development using Simulink® and OPC technology." Information, Communication and Automation Technologies (ICAT), 2011 XXIII International Symposium on. IEEE, 2011.

[8] Zouari, Lilia, Mossaad Ben Ayed, and Mohamed Abid. "A hardware in the loop simulation for electrically driven robot manipulator." Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on. IEEE, 2015.

[9] Yoon, Myung-Hwan, Jae-min Lee, and Jung-Pyo Hong. "The simulink model of motor system for HEV using HILS (Hardware-In-the-Loop)." Electromagnetic Field Problems and Applications (ICEF), 2012 Sixth International Conference on. IEEE, 2012.

[10] Sarikan, Alper, and M. Timur Aydemir. "Real time digital simulation (RTDS) software and hardware in the loop (HIL) architecture for brushless DC motors." MELECON 2010-2010 15th IEEE Mediterranean Electrotechnical Conference. IEEE, 2010.

[11] Selvamuthukumaran, Rajasekar, and Rajesh Gupta. "Rapid prototyping of power electronics converters for photovoltaic system application using Xilinx System Generator." IET Power Electronics 7.9 (2014): 2269-2278.

[12] https://reference.digilentinc.com/reference/pmod/specification?redirect=1#introduction

[13] https://www.digilentinc.com/Pmods/Digilent-mod_%20Interface_Specification.pdf

[14] https://reference.digilentinc.com/reference/pmod/pmodda2/reference-manual

[15] https://reference.digilentinc.com/_media/pmod:pmod:pmodAD1_rm.pdf

[16] https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/sysgen_ref.pdf

[17] https://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html.

APPENDIX A

```
/*    frequency devider*/
module frequency_devider(
  input  clk1,ena  ,
  output reg  done
);
reg [24:0] count1;
always @(posedge clk1)
     begin
          if(ena)
                count1<=count1+1;
                done=count1[12];
      end
endmodule
/*    holding the initial values of sync and csel for some time*/
module wait1(
  input   clk3,ena2,reset,
  output reg  done2 ,
  output reg [4:0]count4    /// internal counter
```

```
);
reg [4:0] count5;          /// the value which will be sent to main module
  always @(posedge clk3)
   begin
        if(reset)
              begin
                   count5<=5'b00000;
                   count4<=count5;
              end
        else
             begin
                  if(ena2)
                       begin
                            count5<=count5+1;
                            count4<=count5;
                            done2=1'b0;
                       end
                  else
                       begin
                            done2=1'b1;
                            count5<=count5;
```

```
                    end
                end
  end
endmodule
/*    genetating 16 pulses and data shifting mechanism*/
module puls_counter(
  input  wire clk2,ena1,preset,
  output reg  done1 ,puls_sync,puls_csel,
  output reg [3:0] v,
  output reg [4:0]count3      /// internal counter
);
reg [4:0] count2;       /// the value which will be sent to main module
reg [15:0] n_s,p_s;
localparam [15:0] o1 =  16'b0000000000000000,   o2 =  16'b0000000000000001,
                  o3 =  16'b0000000000000010,   o4 =  16'b0000000000000100,
                  o5 =  16'b0000000000001000,   o6 =  16'b0000000000010000,
                  o7 =  16'b0000000000100000,   o8 =  16'b0000000001000000,
                  o9 =  16'b0000000010000000,  o10 =  16'b0000000100000000,
                  o11 = 16'b0000001000000000,  o12 =  16'b0000010000000000,
                  o13 = 16'b0000100000000000,  o14 =  16'b0001000000000000,
                  o15 = 16'b0010000000000000,  o16 =  16'b0100000000000000;
```

```
always @(posedge clk2)
        begin
          if(preset)
                begin
                   count2<=2'b00000;
                  count3<=count2;p_s=o1;
                end
            else
                begin
                   if(ena1)
                        begin
                            if(puls_sync) ////  to keep counting only with the positive value of puls_sync
                                begin
                                    count2<=count2+1;
                                    count3<=count2;
                                    p_s<=n_s;
                                end
                            else
                                begin
                                    p_s<=p_s; /* no change if puls_sync =0
                                end
```

```
                                        done1=1'b0;
                                        puls_sync=~puls_sync;  /// sync
                                        puls_csel=1'b0;        ///csel
                                    end
                                else
                                    begin
                                        count2<=count2;
                                        done1=1'b1;
                                        count3<=count3;
                                        puls_sync=1'b0;
                                        puls_csel=1'b1;
                                        p_s<=p_s;
                                    end
                            end
                        end
/* to shift the data bit by bit ....   */
 always @ * begin
                if(puls_sync)
                    begin   /// only if puls_sync =1
                        n_s<=p_s;
                            case(p_s)
```

```
                o1 :begin v=4'b1111;  n_s<=o2 ; end
                o2 :begin v=4'b1110;  n_s<=o3 ; end
                o3 :begin v=4'b1101;  n_s<=o4 ;end
                o4 :begin v=4'b1100;  n_s<=o5 ;end
                o5 :begin v=4'b1011;  n_s<=o6 ;end
                o6 :begin v=4'b1010;  n_s<=o7 ;end
                o7 :begin v=4'b1001;  n_s<=o8;end
                o8 :begin v=4'b1000;  n_s<=o9 ; end
                o9 :begin v=4'b0111;  n_s<=o10;end
                o10:begin v=4'b0110;  n_s<=o11 ; end
                o11:begin v=4'b0101;  n_s<=o12;end
                o12:begin v=4'b0100;  n_s<=o13 ; end
                o13:begin v=4'b0011;  n_s<=o14;end
                o14:begin v=4'b0010;  n_s<=o15  ;end
                o15:begin v=4'b0001;  n_s<=o16;end
                o16:begin v=4'b0000;  n_s<=o1;end
            endcase
        end
    else
        n_s<=p_s;  /// keep the old value
end
```

```
endmodule
module up_down_counter(
  input  wire clk4,up,down,up_down_reset,go,
  output reg  doneup ,donedown,
  output reg [15:0]count6    /// internal counter
);
reg [15:0] count7;          /// the value which will be sent to main module
reg direction;
  always @(posedge clk4)
                    begin
                        if(up_down_reset) begin count7<=16'b0000000000000000; count6<=count7; end
                        else
                              begin
                                    if(up)
                                          begin
                                              if(count7==16'b0000111111111111)
                                                          begin
                                                              count7<=count7;count6<=count7;
                                                          end
                                    else
                                          begin
```

```
                    count7<=count7+1;
                    count6<=count7;end
                end
            if(down)
                begin
                    if(count7==16'b0000000000000000)
                        begin
                            count7<=count7;
                            count6<=count7;
                        end
                    else
                        begin
                            count7<=count7-1;
                            count6<=count7;
                        end
                end
        end
            end
endmodule
```

```
/*        the main module         */
module mou (input clk,goup,godown, output wire l0,wire l1,wire l2,wire l3,wire l4,wire l5,wire j1,wire j4,wire j2,wire j3,wire
l7);
 wire devide_clock,wait1_done,puls_done,csel,sync;
 wire [4:0] c_count_to_16;
 wire [3:0 ]location;
 wire [4:0]c_wait1;
 reg e_count_to_16, e_wait1,re ;
 //reg [4:0]cout;
 reg  ps,ns;
                    //bit0         bit11
wire [15:0] dout;//=16'b1111100000110000;//2.5v
 reg led_out;
 assign l0=devide_clock;
 assign l1=wait1_done;
 assign l2=puls_done;
 assign l3=csel;
 assign l4=sync;
 assign l5=ns;
 assign l7=re;
```

```
assign j1=csel;
assign j4=sync;
assign j2=ns;
assign j3=devide_clock;
  frequency_devider (
          .ena( 1'b1),      // enabling the counter
          .clk1( clk),      // 100MHZ clock FPGA
          .done(devide_clock)// devided clock ---- outpout of the module

 );
  wait1( //to keepthe value of csel and sync 1 for some time
           .ena2( e_wait1),     // enabling the counter for 8
           .clk3(devide_clock), // devided clock ---- input to muudle
           .count4(c_wait1),    // counted value output
           .done2(wait1_done),  // indicates the the counting done output
           .reset(re)          //  to reset the module      );
puls_counter(    // to make csel 0 and sync blinkig 16 times
            .ena1(e_count_to_16),      // enabling the counter for 16
            .clk2( devide_clock),      // devided clock
            .count3(c_count_to_16),    // counted value output   output
            .done1(puls_done),         // indicates that the counting is finish   output
```

```
            .puls_csel(csel),         // csel value   output
            .puls_sync(sync),         // sync value   output
            .v(location),             // the position of data that will be shifted to output  output
            .preset(re)               //  to reset the module   input
  );
  up_down_counter( //to keepthe value of csel and sync 1 for some time
              .up(goup),
            .down(godown),    // enabling the counter for 8
            .clk4(re), // devided clock ---- input to muudle
            .count6(dout),  // indicates the the counting done output
            .up_down_reset(1'b0)              );      //  to reset the module
 always @*
begin
  if(c_wait1==5'b00111)
              begin
                    e_wait1=1'b0;
 if(c_count_to_16==5'b01111)
              begin
                    e_count_to_16=1'b0;
                    led_out=led_out;
                    re<=1'b1;
```

```
                end
    else
                begin
                    e_count_to_16=1'b1;
                    if(sync==1'b1)
                            begin
                                ns<=dout[location];
                                ps=ns;
                            end
                            else
                                ns<=ps;
                                re<=1'b0;      end
                end
    else
                begin
                    e_wait1=1'b1;
                    e_count_to_16=1'b0;
                    re<=1'b0;                  end
    end
endmodule
```

APPENDIX B

```
module sendrecive(
    input clk1,
    input  p16bitin,
    output wire jb0,   // csel signal jb0 w12
    output wire jb3, // sync signal    jb3 w8
    output reg bit16out,// data out pin jb1 w11
    output reg bit16out2,// data out pin pmod ja pin aa11
    output wire led,
    output wire ledclk    // devided clk jb2 v10
                );
    reg cselout ;
    reg [4:0] vdevider;
    reg [4:0] bitcounter;
    reg [11:0] bit12in=12'b000000000001;
    reg [16:0] bit16in;
    reg ledon;
     assign led=ledon;
     assign ledclk=ledon;
     assign jb0=cselout;
```

```
 assign jb3=ledon;
always @(posedge clk1)
 begin
      bit16in<={bit12in,5'b00000};
      vdevider<=vdevider+1;
      ledon<=vdevider[4];
      bit16out2<=p16bitin;
     if(bitcounter==5'b10000)cselout=1'b1;else cselout=1'b0;
 end
always @ (posedge ledon)
  begin
   if(bitcounter==5'b10000)begin bitcounter<=5'b00000;
end // 16 piuls
    else begin
           bitcounter<=bitcounter+1;
           bit16out<=bit16in[bitcounter];

        end
  end
endmodule
```

APPENDIX C

```
module pmodad1driverv2(
  input clk,
  input ce,
  input wire jb2,              // data input channel 1
  input wire jb3,              // data input channel 2
  output wire  jb1 ,            //chip select
  output wire  jb4,             // divided clock
  output wire  ja1,// 12 bit testrs
  output wire  ja2  // ena signal

 reg ok,ok1; ///     for extraction if 1 trans1=trans1 if 0 trans1=trans
 reg [4:0] ffdevider;
 reg [4:0] sfdevider;                // frequency divider counter
 reg [4:0]  adcsetcounter1;
 reg [4:0]  adcsetcounter2;
 wire scl;
 reg fledon,sledon;
 wire bit12test;
```

```
   assign ja1= jb2;
   assign ja2=jb3;
   assign led=fledon;
   assign jb4=fledon;
   assign jb1=scl;
   assign ledclk=fledon;
   assign scl =  ((adcsetcounter2>=5'b00000)&&(adcsetcounter2<=5'b01111)) ? 1'b0:1'b1 ;
   always @(posedge clk)
   begin
      ffdevider<=ffdevider+1;
      fledon<=ffdevider[4];
      sfdevider<=sfdevider+1;
      sledon<=sfdevider[4];
    end
always @ (posedge fledon)
              begin
            if(adcsetcounter1==5'b10111)
                                    begin
                                         adcsetcounter1=5'b00000;
                                    end
                                else
```

```
                                    begin
                                        adcsetcounter1<=adcsetcounter1+1;
                                    end

                                end
 always @ (posedge sledon)
            begin
             if(adcsetcounter2==5'b10000)
              begin
                adcsetcounter2=5'b00000;
              end
            else
              begin
                adcsetcounter2<=adcsetcounter2+1;
              end
          end
endmodule
```