NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DESIGN AND IMPLEMENTATION OF A MOTOR INCREMENTAL SHAFT ENCODER

by

Andrew M. La Valley

September 2008

Thesis Advisor:	Alexander L. Julian
Second Reader:	Roberto Cristi

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2008	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Design and Implementation of a Motor Incremental Shaft Encoder			**5. FUNDING NUMBERS**
6. AUTHOR(S) Andrew M. La Valley			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**
13. ABSTRACT (maximum 200 words)			
14. SUBJECT TERMS Motor control, Encoder, FPGA			**15. NUMBER OF PAGES** 119
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

The Department of Electrical and Computer Engineering at the Naval Postgraduate School continuously develops new design applications and searches for new ways to arm the students with the tools necessary to gain a greater understanding of advanced motor applications. One such tool is the Student Design Center (SDC). The SDC utilizes Field Programmable Gate Array (FPGA) technology for digital control of motor applications. One of the key factors in motor control is having the capability to measure the rotor position. This thesis lays the ground work for motor position control, and also focuses on the design and implementation of an electrical interface for an Incremental Shaft Encoder with the SDC. A digital algorithm was created specifically for the Incremental Shaft Encoder to interface with an FPGA in order to interpret the encoder's output signals into angular position, total degrees traveled, detection of clockwise and counter-clockwise rotation and speed estimation.

THIS PAGE INTENTIONALLY LEFT BLANK

DESIGN AND IMPLEMENTATION OF A MOTOR INCREMENTAL SHAFT ENCODER

Andrew M. La Valley
Lieutenant, United States Navy
B.S., San Diego State University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2008

Author:	Andrew M. La Valley
Approved by:	Alexander L. Julian Thesis Advisor
	Roberto Cristi Second Reader
	Jeffery B. Knorr Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Department of Electrical and Computer Engineering at the Naval Postgraduate School continuously develops new design applications and searches for new ways to provide the students with the tools necessary to gain a greater understanding of advanced motor applications. One such tool is the Student Design Center (SDC). The SDC utilizes Field Programmable Gate Array (FPGA) technology for digital control of motor applications. One of the key factors in motor control is having the capability to measure the rotor position. This thesis lays the ground work for motor position control, and also focuses on the design and implementation of an electrical interface for an Incremental Shaft Encoder with the SDC. A digital algorithm was created specifically for the Incremental Shaft Encoder to interface with an FPGA in order to interpret the encoder’s output signals into angular position, total degrees traveled, detection of clockwise and counter-clockwise rotation and speed estimation.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS, ACRONYMS, AND ABBREVIATIONS

CCW	Counter-Clockwise
CW	Clockwise
FPGA	Field Programmable Gate Array
IC	Initial Conditions
ICB	Information Capture Block
ISE	Incremental Shaft Encoder
NPS	Naval Postgraduate School
PCB	Printed Circuit Board
RPM	Revolutions per Minute
SCIM	Squirrel Cage Induction Motor
SDC	Student Design Center
VHDL	Verilog Hardware Description Language
VSC	Voltage Source Converters
ZCE	Zero Crossing Event

EXECUTIVE SUMMARY

The Department of Electrical and Computer Engineering at the Naval Postgraduate School continuously develops state of the art design tools so that the students can gain a greater understanding of advanced motor applications. One such tool is the Student Design Center (SDC). The SDC was created to familiarize the student with the basic application of solid state power design and control. Originally, the SDC made it possible for a student to make accurate predictions of voltage source converters (VSC) behavior via software simulation; these simulated results could also be tested against actual hardware components [1]. The design center is shown in Figure 1.

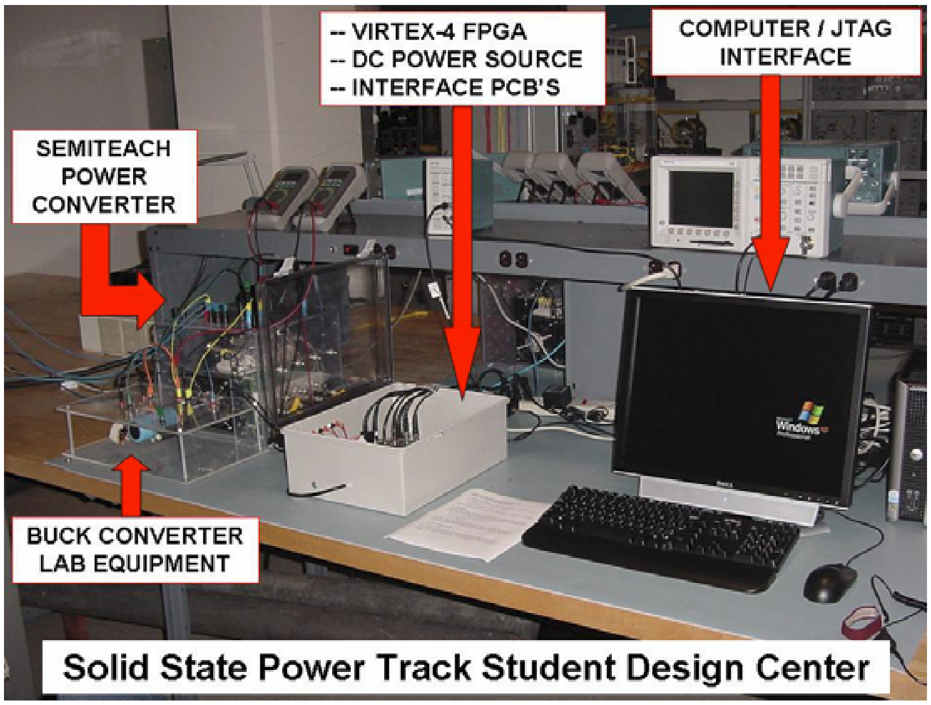

-- VIRTEX-4 FPGA
-- DC POWER SOURCE
-- INTERFACE PCB'S
COMPUTER / JTAG INTERFACE
SEMITEACH POWER CONVERTER
BUCK CONVERTER LAB EQUIPMENT
Solid State Power Track Student Design Center

Figure 1. Student Design Center [From [1]].

Before the start of this thesis, the SDC was equipped with the following hardware components: a Field Programmable Gate Array (FPGA), a Voltage Source Converter (VSC), and several other off-the-shelf components, a circuit board interface between FPGA and the power source, and a desktop computer [1]. Now, the SDC has been

upgraded with an MES20 (Type C) Incremental Shaft Encoder (ISE), which accurately measures the angular position and speed of rotation of rotating machines.

The SDC utilizes FPGA technology for digital control of motor applications. Having SDC equipped with an FPGA alone was simply not enough hardware to control the rotation of a motor. Additional hardware and software was required that could detect and measure rotor speed and angular position. Accordingly, the ISE was added to the SDC's hardware in hopes of expanding the student's educational resources in the area of motor control. The ISE can be seen in Figure 2.

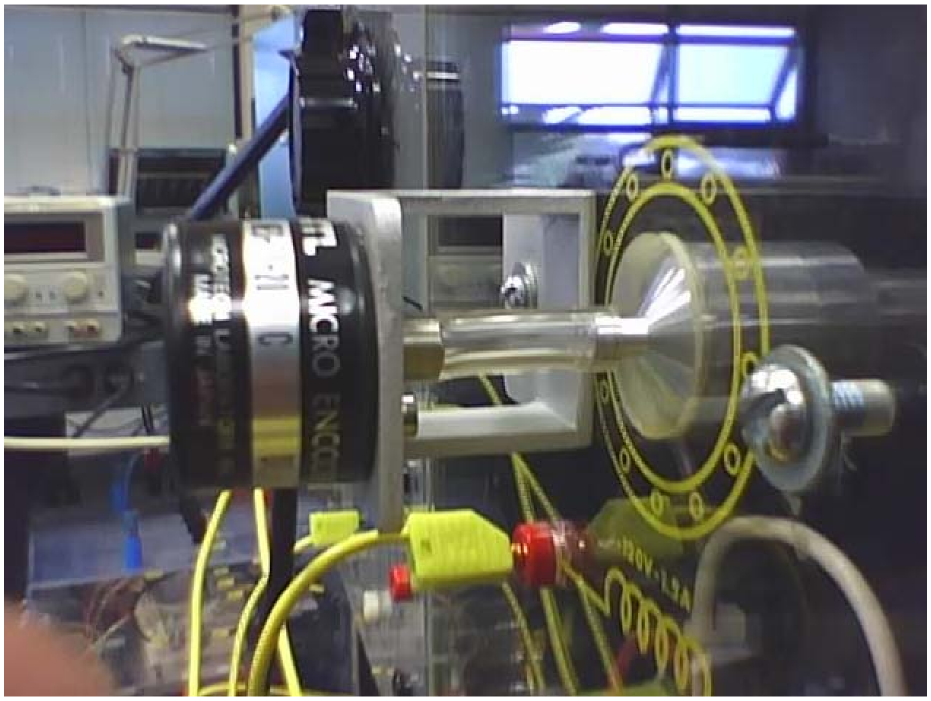


Figure 2. MES20 (Type C) Incremental Shaft Encoder mounted on a Squirrel Cage Induction Motor (SCIM).

The SDC utilizes Mathworks' Simulink software in the development of hardware control. The XILINX Foundation software produces Verilog Hardware Description Language (VHDL) code that will interface the FPGA with a hardware component assigned [1].

The main goal of this thesis was the design and implementation of an electrical interface for the ISE with the SDC via FPGA. More specifically, through the use of these programming software, a digital algorithm was created specifically for the ISE interface with an FPGA in order to interpret the encoder's output signals into angular position, total degrees traveled, detection of clockwise and counter-clockwise rotation as well as speed estimation.

A secondary objective of this thesis was to present the reader with an overview of the hardware and software required in the SDC. Specifically, it highlights current FPGA applications as well as the ISE interface with the FPGA via Simulink and XILINX Foundation simulation development.

This research will compare a simulated operation of the encoder with measured results which teaches the student that the physical operation of electronic equipment can be predicted via simulation prior to testing. Testing then is just a validation of the design that is accomplished using the simulation tools.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to take this opportunity to express my appreciation to the staff and faculty of the Naval Postgraduate School (NPS) for their dedication to the military members and recognize their relentless pursuit for excellence in higher education. Specifically I would like to thank Professor Julian for his guidance and instruction during my thesis research and throughout my time spent at NPS. Often his common–sense approach to problem solving was just the thing that would turn the light-bulb on in regards to discovering solutions to complex problems. Working with him was one of the highlights of my educational journey. It has truly been an honor and a privilege.

Special thanks to all the laboratory technicians throughout my tour of duty at NPS; specifically, Jeff Knight and Warren Rogers for there never ending zeal and dedication to higher learning.

Lastly and most importantly, I thank my wife, Jennifer, and two amazing daughters, Isabelle and Hailey. You give my life meaning and purpose. Without you three in my life, all that I have accomplished would be meaningless and empty.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

The Student Design Center (SDC) at the Naval Postgraduate School Electrical Engineering Department was originally created for the purpose of exposing students to the process of transforming performance requirements into basic design [1]. The SDC utilizes FPGA technology for digital control of motor applications. Not only does the SDC provide the student with the tools necessary to make accurate system behavior predictions, but it also exposes the students to basic power electronics design, so that they can test their simulations on actual hardware.

Undergraduate students are often introduced to micro-controller software design in introductory courses, yet typically software and hardware issues have remained relatively separated [2]. Because of the growing importance of system-level embedded design courses at the graduate level, each laboratory was specifically tailored in such a way to give students practical problems in a real-world environment while preparing for future study in product design, testing, and control.

Students use Mathworks Simulink and XILINX Foundation software to generate Verilog Hardware Description Language (VHDL) code to program the FPGA. Once the FPGA is properly formatted, it processes the inputs from the actual hardware in accordance with design parameters. Basic knowledge of digital logic design is required, but prior experience with VHDL coding is not.

B. RESEARCH OBJECTIVES

The main goal of this thesis was to design and implement an electrical interface for the ISE with the SDC via FPGA. More specifically, through the use of Mathworks' Simulink and XILINX Foundation software, a digital algorithm was created for the ISE and interfaced with an FPGA in order to interpret the encoder’s output signals into angular position, total degrees traveled, detection of clockwise and counter-clockwise rotation and speed estimation.

A secondary objective was to present the reader with an overview of the hardware and software required in the SDC. Specifically, this thesis highlighted current FPGA applications as well as ISE technology and its interface with the FPGA via Simulink and XILINK Foundation simulation development.

This research will compare simulated operation of the encoder with measured results which shows that the physical operation of electronic equipment can be predicted via simulation prior to testing. Testing then is just a validation of the design that is accomplished using the simulation tools.

C. APPROACH

The first step in developing an interface with the ISE was to analyze the ISE’s output signal and develop a simulation to reproduce that exact signal through the use of Mathworks’ Simulink and XILINX Foundation simulation software. Once this was accomplished, the simulation was then expanded to interpret the encoder’s output signals into angular position, total degrees traveled, detection of clockwise and counter-clockwise rotation as well as speed estimation. Upon achieving this milestone, the Simulink and XILINX Foundation simulation was then utilized as a template in order to generate VHDL code that was used to interface an FPGA with the hardware components in the SDC. This enabled the SDC to accurately record and detect rotational parameters, including speed estimation, in accordance with the simulated design. Lastly, the actual results obtained were then compared to simulated results in order to validate both the hardware and the software.

D. THESIS ORGANIZATION

- Chapter I introduced research goals and presented the organization of the thesis.
- Chapter II presents an overview of the SDC's hardware and software, which includes a detailed analysis of the output signal generated by the ISE as well as FPGA current technology. It describes, in detail, how the encoder generates its three output wave forms, as well as highlights how these wave forms can be utilized for specific rotational data collection.
- Chapter III discusses the FPGA interface with the SDC, as well as the ISE. It then describes how the simulation reproduced the ISE source signal.
- Chapter IV explores the design, construction, and testing of the Rotor Speed Indicator. This examination includes a comprehensive analysis of simulation construction, and also discusses simulation overall results.
- Chapter V highlights the hardware and software interface used in the analysis of the ISE.
- Chapter VI concludes this thesis with actual hardware and algorithm performance results. Future research opportunities related to total motor control are also discussed.
- Appendix *A* provides technical specification as well as MATLAB code utilized in the algorithm and simulation's development.
- Appendix *B* provides a printout of the entire XILINX Foundation model used in the algorithm and simulation's development.

THIS PAGE INTENTIONALLY LEFT BLANK

II. STUDENT DESIGN CENTER OVERVIEW

A. FIELD PROGRAMMABLE GATE ARRAY OVERVIEW

A Field-Programmable Gate Array (FPGA) is an integrated circuit that can be programmed in the field after manufacture. An FPGA is similar in principle to, but has vastly wider potential application than, programmable read-only memory chips [3]. More specifically, the FPGA is a semiconductor device that contains programmable logic components and programmable interconnects that can be programmed to perform the function of basic logic or even more complex systems, such as decoders and other mathematical functions [4]. Furthermore, these programmable interconnects allow logic blocks to be implemented as needed by either the system designer or the customer, hence the name "Field Programmable." This means that it gives the user the opportunity to reprogram in the "field," considering the logic is changeable. FPGA can also used by engineers in the design of specialized integrated circuits that can later be produced as hard-wired in large quantities for distribution to computer manufacturers and end users. A detailed description of the FPGA technology and its integration into the SDC can be seen in the thesis entitled, "Field Programmable Gate Array Control of Power Systems in Graduate Student Laboratories." by Joseph E. O'Connor [1].

B. MES20 (TYPE C) INCREMENTAL SHAFT ENCODERS (ISC) OVERVIEW

The ISE seen in Figure 3 is a digital optical encoder which is a device that converts motion into a sequence of digital pulses. By counting a single bit or by decoding a set of bits, the pulses can be converted to relative or absolute position measurements.

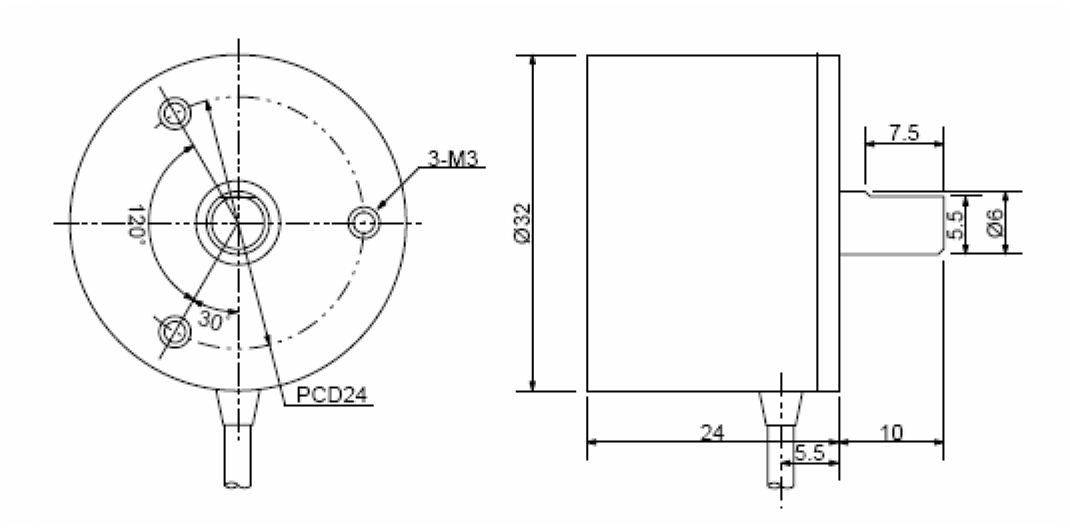


Figure 3. MES20 (Type C) Incremental Shaft Encoder physical parameters [After [5]].

There are several types of encoders on the market. These include both linear and rotary configurations but the most common type is rotary. This particular encoder is a rotary incremental encoder, which produces digital pulses as the shaft rotates, allowing measurement of the relative position of shaft. Most rotary encoders are composed of a glass or plastic code disk with a photographically deposited radial pattern organized in tracks. As radial lines in each track interrupt the beam between a photo-emitter-detector pair, digital pulses are produced [6].

Additionally, this encoder consists of three output signals or phases. The first two phases consist of two tracks and two sensors whose outputs are called *A* and *B* Phase. By counting the number of pulses and knowing the resolution of the disk, in this case 200 pulses per rotation of *A* or *B* Phase, the angular motion can be measured. The *A* and *B* Phase are used to determine the direction of rotation by assessing which phase "leads" the other. The two phases are 1/4 cycle, or 90 degrees out of phase with each other and are known as quadrature signals [7]; a generic example of quadrature signals can be seen in Figure 4.

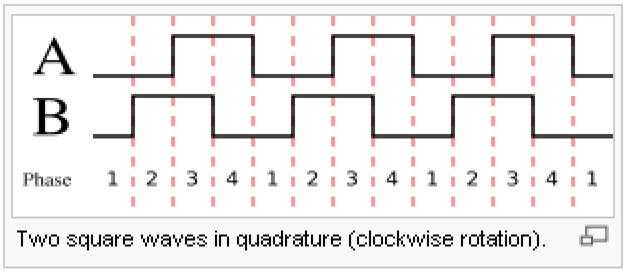


Figure 4. Generic Quadrature Output Signal. [From [7]].

Moreover, the third output channel is called the *Z* Phase; it yields one pulse per revolution, which is useful in counting full revolutions. It is also useful as a reference to define a home base or a Zero Crossing Event (ZCE). As the shaft rotates, pulse trains occur on these channels at a frequency proportional to the shaft speed, and the phase relationship between the signals yields the direction of rotation [6]. Note that the pulse width of the *Z* Phase (PWZ) can vary in accordance with equation (1). The 3 Phases CW and CCW square-wave orientation can be seen in Figure 5.

$$PWZ = P \pm 0.75P \tag{1}$$

where *P* is the pulse width.

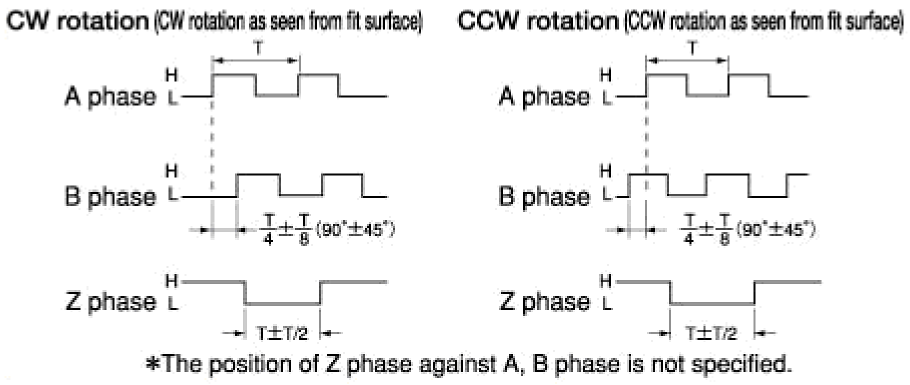

CW rotation (CW rotation as seen from fit surface)
CCW rotation (CCW rotation as seen from fit surface)
T
A phase
H
L
B phase
$\frac{T}{4} \pm \frac{T}{8}$ (90°±45°)
Z phase
T±T/2
*The position of Z phase against A, B phase is not specified.

Figure 5. ISE CCW and CW Output [From [5]].

C. CHAPTER SUMMARY

This chapter provided a detailed overview of the SDC and its hardware components; namely the FPGA and the ISE technology. It also described, in detail, how the encoder generated its 3 output waveforms, and highlighted how these waveforms were utilized for specific rotational data collection. Chapter III will discuss the FPGA interface with the SDC as well as the ISE. It will also underscore the development of the simulation that recreates expected performance of the ISE source signal, as well as discuss the simulated reproduction of the 16 possible states the encoder generates. Lastly, it will conclude with a review of simulated output results.

III. ISE SIMULATED INPUT SIGNAL RECREATION

A. SIMULATED SOURCE SIGNAL DEVELOPMENT

The first step in developing an interface with the ISE was to analyze the device's output signal and develop a simulation to reproduce that source signal in its entirety through the use of Simulink and Foundation simulation software. Equipped with the ISE square-wave architecture, the 3 square-wave outputs were reproduced by using a combination of Pulse Generators, Multiplexers, a Step Function Generator, and a Switching Block. The circuit architecture can be seen in Figure 6.

In order to construct both the CW and CCW square-wave source signal that mirrored the signals produced in the ISE, 6 separate Pulse Generators provided 6 square-wave inputs that were used to drive the Simulink simulation. Specifics on how the simulation reproduced CW and CCW rotation are explained in detail in the following sections.

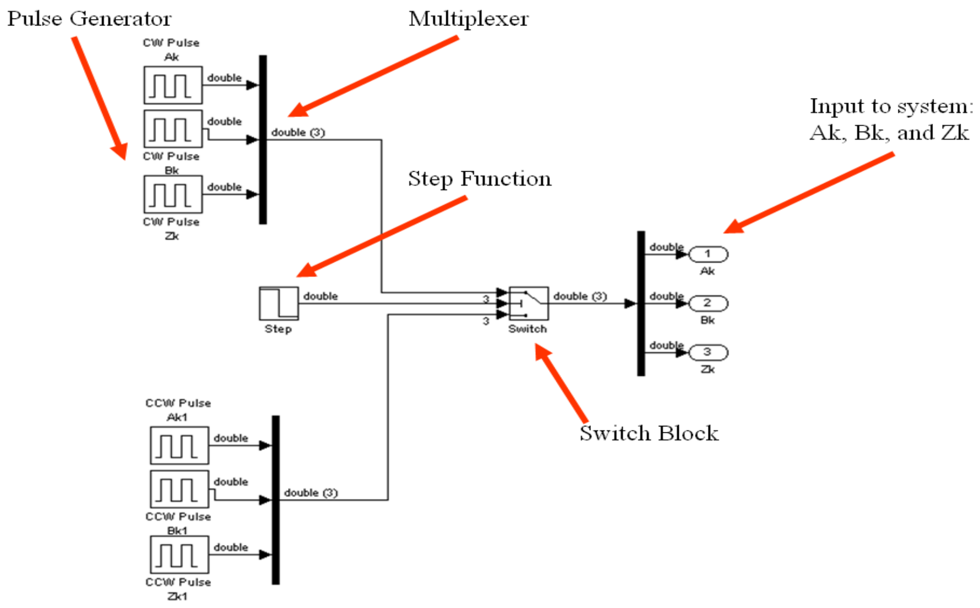


Figure 6. ISE Simulated Source Architecture.

1. Simulated Clockwise Rotation Development

In regards to CW rotation, in order to reproduce the *A* Phase square-wave, a time based pulse was utilized with the wave amplitude set to a value of 1, the period of the wave set to was a initially set at t_square = $2x10^{-4}$ seconds, and the pulse width set to 50%. For the *B* Phase reproduction, the same parameters were applied, with one exception; this time, the *B* Phase was given a phase delay set to t_square/4. Essentially, this forced the *B* Phase to lag the *A* Phase by 90 degrees. Lastly, in either the CW or CCW rotation, the *Z* Phase was reproduced by keeping the *Z* Phase square-wave signal high, or a value of 1, for 99.6% of the time. This enabled the *Z* Phase to simulate a positive high signal that would pulse low once for every 200 pulses of the *A* Phase or *B* Phase. The simulated CW rotation source signal can be seen in Figure 7. Note that in Figure 7, the *A* Phase is leading the *B* Phase by 90 degrees; the *Z* Phase displayed a constant high until pulsing low at approximately 0.004 seconds. Also note that the *Z* Phase pulse width is in accordance with equation (1).

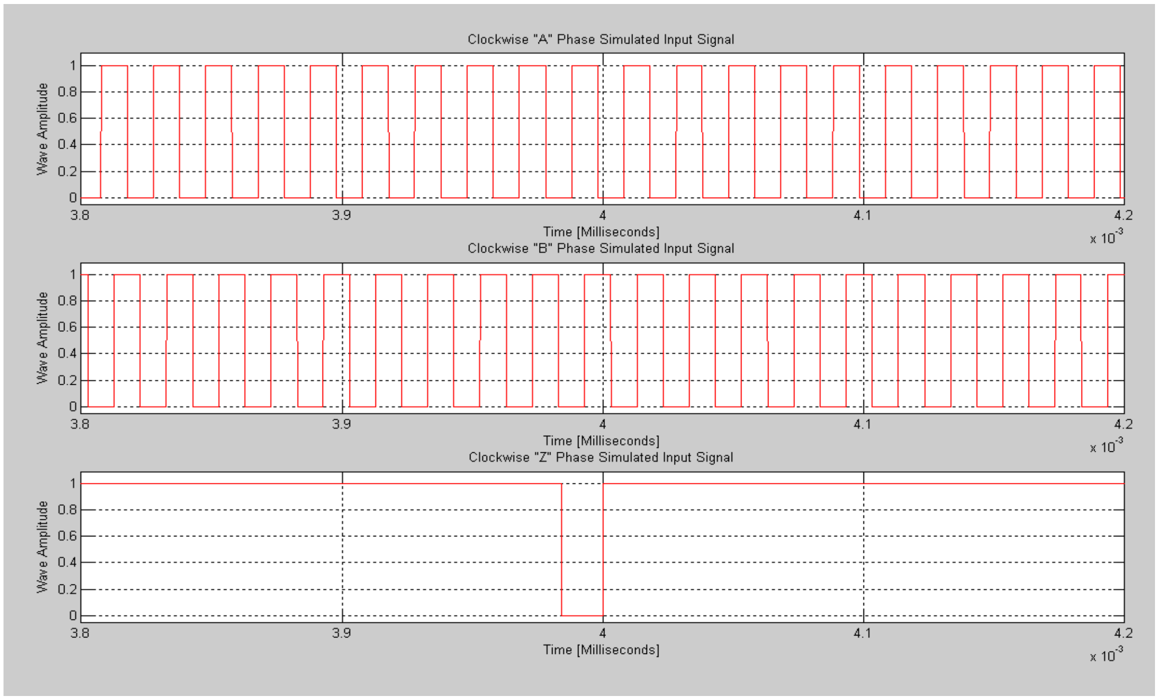


Figure 7. Clockwise *A*, *B* and *Z* Phase Simulated ISE Input Signals.

2. Simulated Counter-Clockwise Rotation Development

For reproduction of CCW rotation, the B Phase had to lead the A Phase by 90 degrees. This was accomplished simply by applying the phase delay of t_square/4 on the A Phase instead of the B Phase; this effectively simulated a CCW rotation. This source signal simulated output results can be seen in Figure 8. Note that in Figure 8, the A Phase is now lagging the B Phase by 90 degrees; the Z Phase again displays a constant high signal until pulsing low at approximately 0.004 seconds.

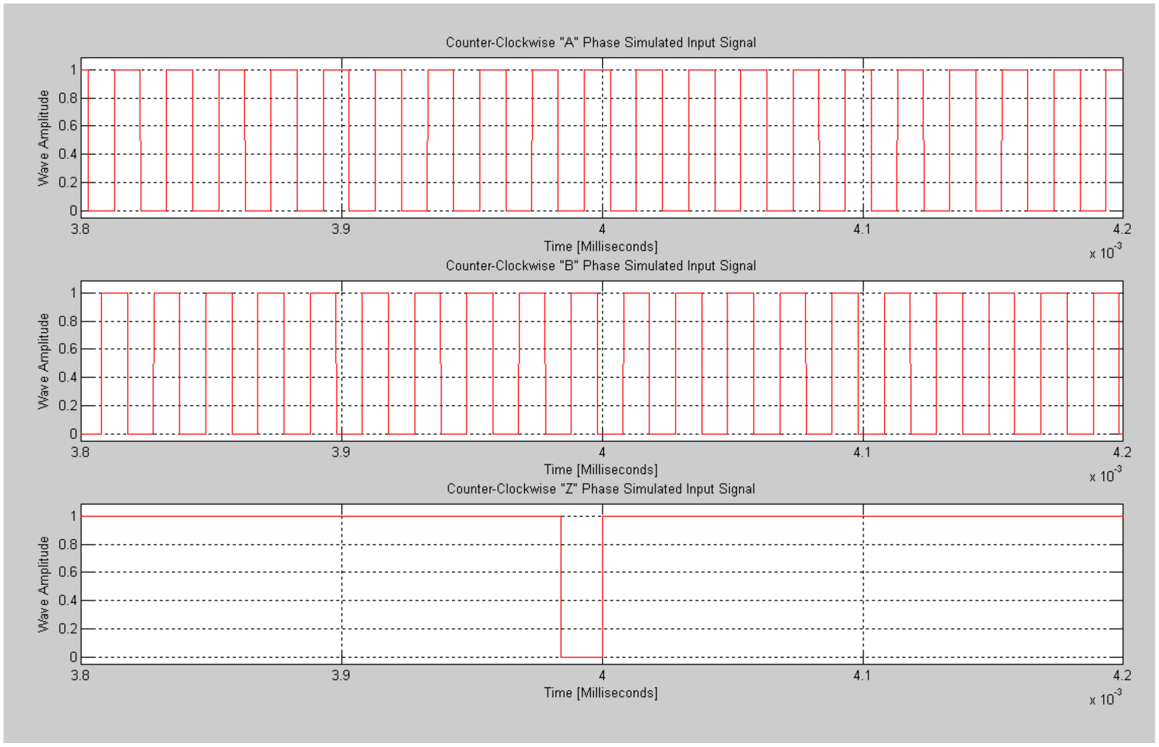


Figure 8. Counter-Clockwise *A*, *B* and *Z* Phase Simulated ISE Source Signal.

B. SIMULATION SOURCE INPUT DEVELOPMENT

Next, combining all 6 input signals into a single vector output was accomplished via a Multiplexer Block. These output signals could then be utilized to drive the simulation in either the CW or CCW direction. However, to facilitate proper testing, it was essential to provide the simulation with a method to change the inputs from CW to the CCW direction all in the same simulation period. As a result, using a Step Function

Block in combination with a Switch Block, the simulation could do just that; it could change rotation direction input instantaneously. For example, the Step Function Block provides a step between two definable levels at a specified time. If the simulation step time parameters were set to 0.002 seconds, then once the simulation time reached 0.002 seconds the initial value assigned would shift to the final value, thus changing the output from one definable level to another. Either definable level can then be used to drive the switching block that ultimately changes the source input it reflect CW or CCW operations.

Furthermore, in regards to the Switch Block seen in Figure 6, a user can select the conditions under which the first input is passed with the “Criteria for passing” which is the first input parameter. Then a user can make the block check whether the control input is greater than or equal to the threshold value, strictly greater than the threshold value, or nonzero. If the control input meets the condition set in the “Criteria for passing”, then the first input is passed; otherwise, the third input is passed. Lastly, the middle port on the switch block is called the control port. This control port is then driven by the step function’s output, and in turn, the output of the switch block is the input to the simulation.

C. LOOKUP TABLE DEVELOPMENT

Once the source signal was recreated in simulation, the computer required the capability to interpret these input signals into something more meaningful than just square-wave detection. Recall that the goal of this thesis was to develop a way for the computer to discern between CW and CCW rotation, detect angular position change, be able to record total degrees traveled, as well as accurately measure the speed of rotation. This was accomplished partly through the use of the following Lookup Table (LUT) shown in Table 1.

After analyzing the 3 square-waves produced by the ISE, it became evident that there were 16 possible states that could be generated. These states can be seen in Table 1.

State	A_k	B_k	Z_k	A_{k-1}	P_k	N_k	0_k
1	0	0	0	0	0	0	0
2	0	0	0	1	0	1	1
3	0	0	1	0	0	0	0
4	0	0	1	1	0	1	0
5	0	1	0	0	0	0	0
6	0	1	0	1	1	0	1
7	0	1	1	0	0	0	0
8	0	1	1	1	1	0	0
9	1	0	0	0	1	0	1
10	1	0	0	1	0	0	0
11	1	0	1	0	1	0	0
12	1	0	1	1	0	0	0
13	1	1	0	0	0	1	1
14	1	1	0	1	0	0	0
15	1	1	1	0	0	1	0
16	1	1	1	1	0	0	0

Table 1. The 16 State Table.

1. Determining the States

In reference to Figure 9 and in regards to both CW and CCW rotation, the red vertical lines indicate the previously sampled signal (A_{k-1}). The blue vertical lines indicate the current sample of the *A*, *B* and *Z* Phases (A_k, B_k, Z_k). With this in mind, if A_{k-1} was previously 0, or low, and the current sample of the *A* Phase is 1, or high, B_k is low, and Z_k high, then the device is rotating in the CW direction.

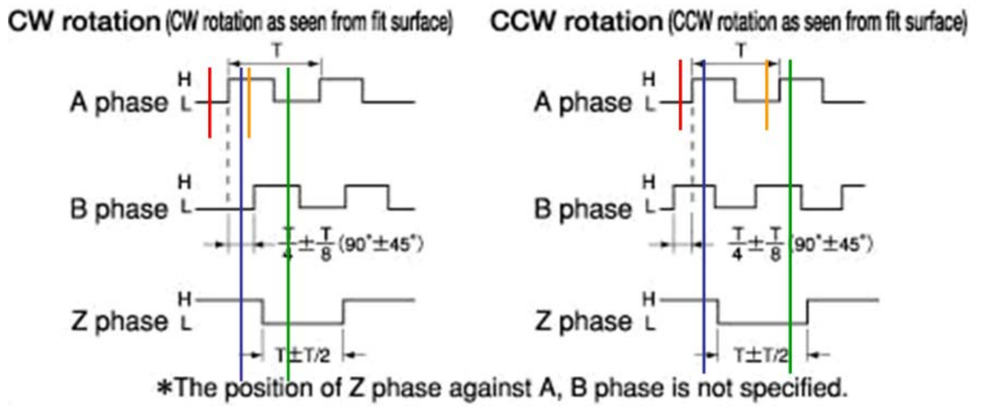

CW rotation (CW rotation as seen from fit surface)
CCW rotation (CCW rotation as seen from fit surface)
A phase
B phase
Z phase
H
L
T
T/4±T/8 (90°±45°)
T±T/2
*The position of Z phase against A, B phase is not specified.

Figure 9. MES20 (Type C) ISE Square-Wave Output [After [5]].

On the other hand, if the A_{k-1} signal was low, and the current sample of A_k, B_k, Z_k are all high, then the device is rotating in the CCW direction. The orange vertical lines indicate the previously sampled signal (A_{k-1}). The green lines indicate the current sample of the *A*, *B* and *Z* Phases (A_k, B_k, Z_k). In this case, when A_k goes from high to low, B_k is high, and Z_k is low, it indicates that CW rotation with a ZCE has occurred. A ZCE indicates that 200 pulses of the *A* or *B* Phase have occurred, thus indicating that 1 revolution of 360 degrees has also occurred. Similarly, if A_k goes from low to high, B_k is high, and Z_k is low, then the device is rotating in the CCW direction with a ZCE detected.

Furthermore, once the 16 possible states were determined and arranged as a lookup table, each individual state was then given a 3 bit binary word that represented the output of that particular state. The most significant bit in the 3 bit binary word indicates whether or not a CW incremental step has occurred and is labeled P_k. For example, if the P_k bit is 1, then a CW incremental step has occurred; if the P_k bit is 0, then no step has been detected. The middle bit of the 3 bit word indicates whether rotation in the CCW has occurred, and was labeled N_k. Lastly, the least significant bit was labeled 0_k; this bit indicates whether a ZCE has occurred in either CW or CCW direction.

2. Lookup Table Functionality

In reference to Table 1, direct your attention to state number 6. State 6 or [0101] is represented by a binary 5, or [101]; meaning that P_k is 1, N_k is 0, and 0_k is 1. A binary 5 indicates that a CW step has occurred because the P_k bit is 1. It also indicated that a ZCE has also occurred considering that the 0_k bit is 1. Similarly, state 11 is [1010]. The output for this state is a binary 4, or [100]; meaning that P_k is 1, N_k is 0, and 0_k is 0. This binary 4 indicates that only a CW step occurred and this time there is no ZCE detected considering 0_k bit is 0. This same pattern holds true for both CCW rotation operations as well as ZCE detection.

D. ISE PHYSICAL ORIENTATION

As seen in Figure 10, the orientation of the encoder became a concern during the development of the model. Specifically the issue was that the encoder faced away from the user and toward the motor it is connected to. This meant that the encoder would register CW rotation when the motor is actually rotating CCW and vise versa. With this in mind, it became evident that rotation detection in the simulation needed to be completely opposite to that of the actual rotation. This was accomplished by simply switching the P_k and N_k output of the lookup table in Table 1 above. More specifically, by swapping out these two binary states, it would correctly represent the rotation of the motor and not the encoder. With the above information in mind, a new state table was generated and can be seen in Table 2.

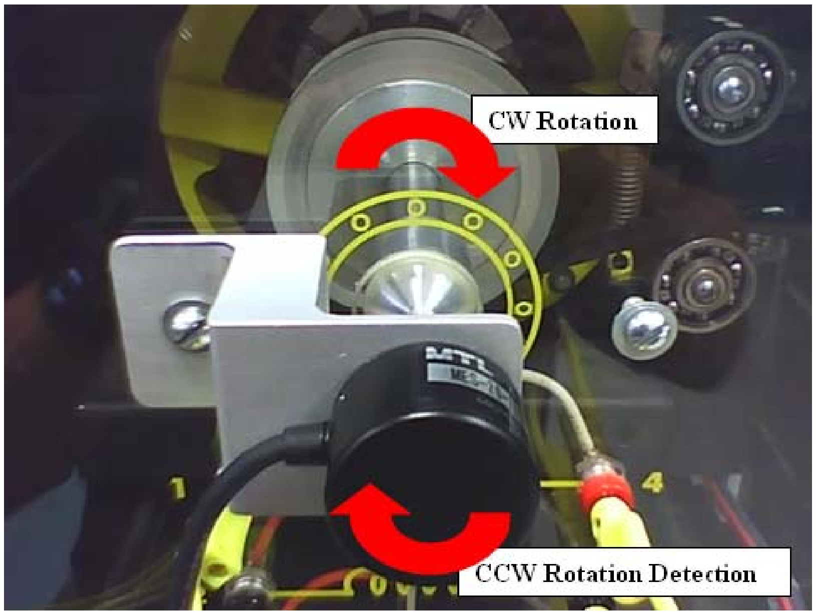


Figure 10. ISE mounted on a Squirrel Cage Induction Motor.

State	A_k	B_k	Z_k	A_{k-1}	P_k	N_k	0_k
1	0	0	0	0	0	0	0
2	0	0	0	1	1	0	1
3	0	0	1	0	0	0	0
4	0	0	1	1	1	0	0
5	0	1	0	0	0	0	0
6	0	1	0	1	0	1	1
7	0	1	1	0	0	0	0
8	0	1	1	1	0	1	0
9	1	0	0	0	0	1	1
10	1	0	0	1	0	0	0
11	1	0	1	0	0	1	0
12	1	0	1	1	0	0	0
13	1	1	0	0	1	0	1
14	1	1	0	1	0	0	0
15	1	1	1	0	1	0	0
16	1	1	1	1	0	0	0

Table 2. Modified 16 State Table.

E. SIMULATION REPRODUCTION OF THE 16 POSSIBLE STATES

Next, the simulation needed the capability to reproduce these 16 physical states when the situation called for it. Therefore through the architecture seen in Figure 11, this goal was achieved. The output of the signal source had to be data type "Boolean." That was accomplished through the use of the Gateway In Blocks. These blocks convert Simulink integer, double and fixed point data types into the System Generator fixed point type. Each block defines a top-level input port in the HDL design generated by System Generator. In this case, they also force the output to be of type Boolean as well as provide a digital input link to the physical hardware that generates the input signal; namely the ISE input signal.

Recall that the simulation required 4 input signals in order to determine the current state, not just the 3 inputs you would expect considering that the encoder only produces 3 square-waves. In particular, the simulation needed the previous sample of the *A* Phase as well as the current sample of the *A*, *B*, and *Z* Phases. Consequently, a Delay Block with a latency of 1 was positioned on the *A* Phase source input signal in order to provide the simulation with the previous state of the *A* Phase, A_{k-1}, as well as the current state of the *A* Phase. Next, all 4 inputs were united through the use of a Concatenation Block. The Concatenation Block has a number of *n* ports, where *n* is some value between 2 and 1024, inclusively, and has 1 output port. The first and last input ports are labeled "hi" and "low," respectively. The input to the "hi" port will occupy the most significant bits of the output and the input to the "lo" port will occupy the least significant bits of the output. Lastly, the output of the Concatenation Block labeled "Concat" was then evaluated against a LUT defined in accordance with Table 2 and via the Read-Only Memory (ROM) Block labeled ROM1. The output of the ROM1 Block is a 3 bit word representing the output of the LUT.

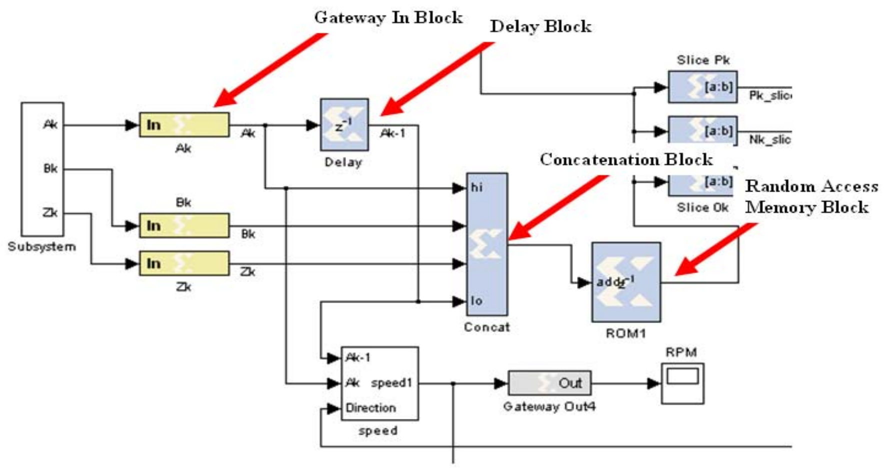


Figure 11. Simulink®/ XILINX® Signal Generation Architecture.

F. SIMULATION M-CODE BLOCK INPUT GENERATION

Now that the 16 states can be recreated in simulation, and the LUT has been defined, the next step in the simulation development was creating the ability to output the correct values of P_k, N_k and 0_k which corresponded to the current state observed. In the previous section, recall that the Concatenation Block provided the input to a ROM1 Block. Recall that each word was associated with exactly 1 address in Table 2. For this simulation, the ROM1 Block generated a specific output in the form of a 3 bit binary word by referencing the LUT which is defined in the initial conditions file and in accordance with Table 2. This 3 bit binary output was then separated bit-by-bit through the use of the Slice Blocks. These Slice Blocks allowed the simulation to literally slice off a sequence of bits from output data and create a new data value; in this case, either a 1 or a 0 as the output. The output data type of the Slice Block is an unsigned value with its binary point set at zero. Again, bear in mind, that the simulation output at this stage of operations is only a 3 bit word, and consequently, the Slice Block labeled P_k observes only the most significant bit, Slice Block N_k looks at the middle bit, and Slice Block 0_k observes the least significant bit. It is the 3 separated bits that are now available to drive the M-Code Block embedded MATLAB™ software. Physical configuration can be seen in Figure 12.

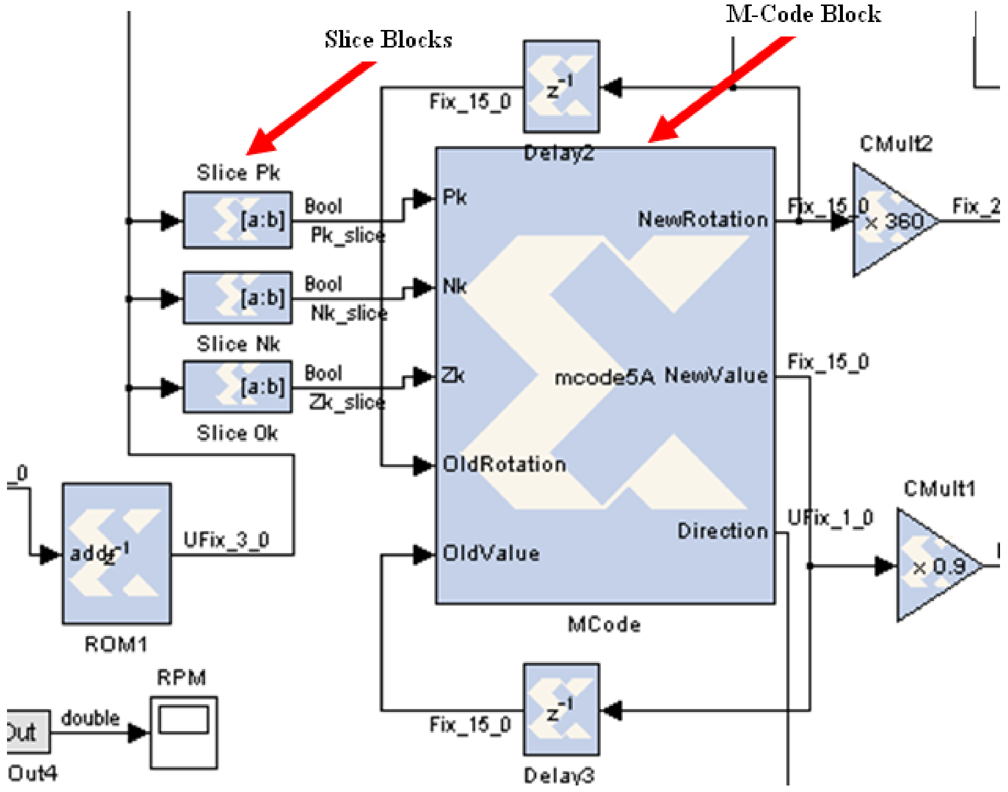


Figure 12. Slice Blocks/M-Code Block Interface Architecture.

G. M-CODE BLOCK ARCHITECTURE/FUNCTIONALITY

By selection of the Boolean output feature in the Slice Block parameter box, the output data was forced to be type Boolean. The individual Boolean output bits were now available to drive the M-Code Block which houses MATLAB code. The M-Code Block is essentially a container for executing a user-supplied MATLAB function within Simulink. The M-Code Block executes the embedded code and calculates the block’s outputs during a simulation. The same code is then translated in a straightforward manner into equivalent behavioral VHDL code when hardware is generated. The block's simulink interface is derived from the MATLAB function signature, and from block mask parameters. In regards to the M-Code Block physical attributes, there is 1 input port for each parameter to the function, and 1 output port for each value the function returns. Port names and ordering correspond to the names and ordering of parameters and return values. In this instance, there are 5 input ports, labeled P_k, N_k, 0_k, OldRotation, and OldValue. Also note that there are 3 output ports labeled NewRotation, NewValue, and the Direction; physical architecture can be seen in Figure 13.

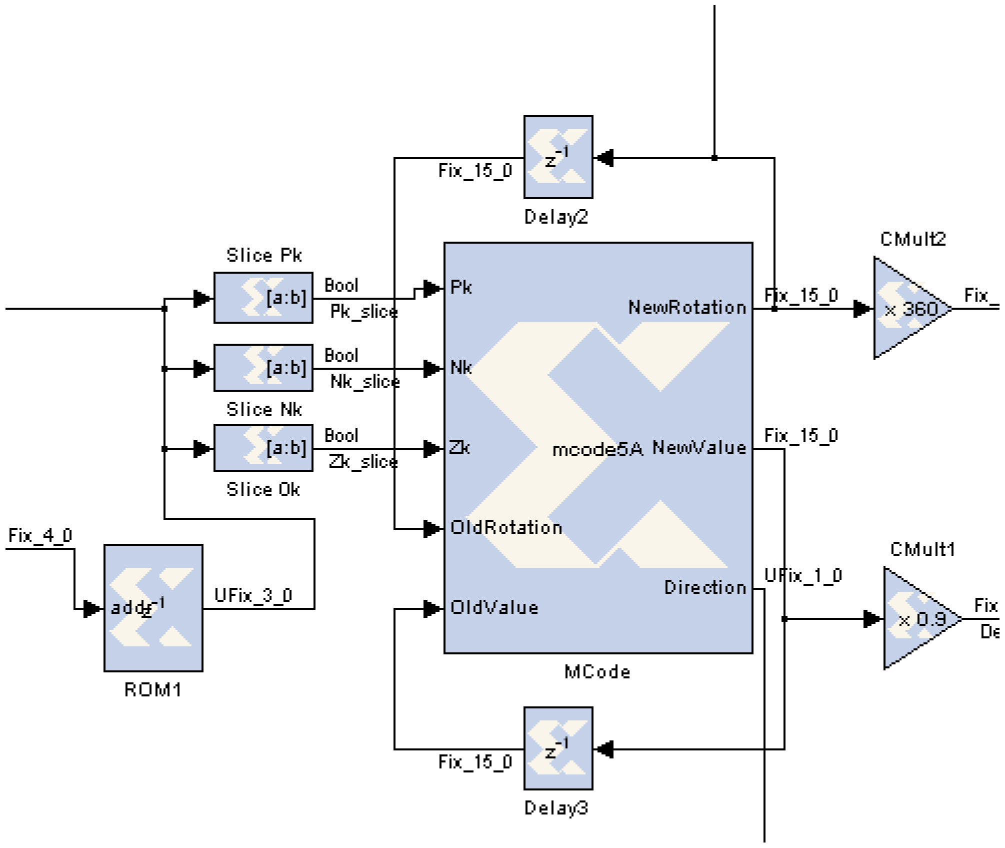


Figure 13. M-Code Block Physical Architecture.

In order to make the MATLAB function imbedded in the M-Code Block operate properly while using XILINX Foundation software, an important issue had to be resolved first. Namely, the use of “While” loops and “For” loops in Foundation software was not an option. More specifically, XILINX Foundation software is just not developed for that particular application, and consequently will not function in that capacity. Therefore, the simulation needed a method to get the old data collected passed on as new data in order to produce meaningful end-product. As a result, two physical feed back loops were constructed; both equipped with a Delay Block with a latency of 1 in order to simulate a “While” loop or a “For” loop type of operation in a XILINX Foundation software format. This allowed the simulation to pass vital data from the previous status to the next round of samples taken. These input loops were labeled NewRotation and NewValue. Note that there is an additional output from the M-Code Block, labeled Direction; however, this output was not delayed at this point in the simulation, but rather used as the input to the Speed Indicator Block. The Direction Bit functionality will be discussed in detail in Chapter IV.

H. M-CODE BLOCK MATLAB CODE

Now that the simulation has the ability to pass previous data to the current data, and the fact that the output from the LUT has been separated into 3 separate Boolean values, namely; P_k, N_k ,and 0_k, The simulation is ready to drive the MATLAB embedded code in accordance with the 3 operational flow charts shown below. Note that each flow chart is operating simultaneously during the simulation.

1. Pk Bit Operational Flow Chart

In reference to Figure 14, once the 3 separate Boolean values P_k, N_k ,and 0_k are inputted to the M-Code Block, the program first looks to see if the P_k bit is either a 1 or 0. If the input is a 1, then the ISC must be rotating in the CW direction and therefore the half-pulse counter is incremented by 1, as well as the Direction Bit is set to 0. As a direct result, the Direction Bit output is then sent as an input to the Speed Indicator Block. On the other hand, if the P_k input is a 0, then the ISC must either be in a static condition, or CCW rotation is occurring; in either case, no action will occur.

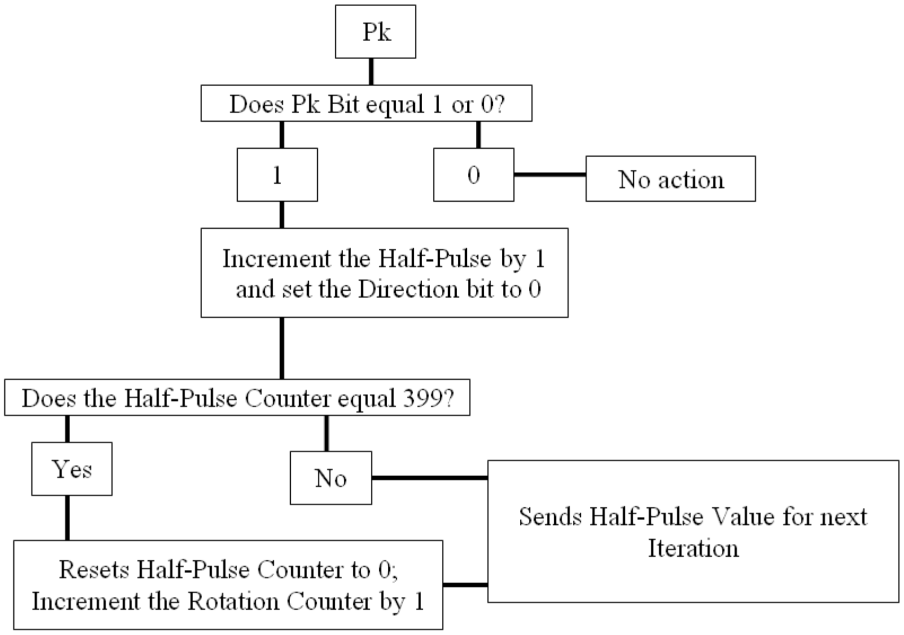

Pk
Does Pk Bit equal 1 or 0?
1
0
No action
Increment the Half-Pulse by 1 and set the Direction bit to 0
Does the Half-Pulse Counter equal 399?
Yes
No
Sends Half-Pulse Value for next Iteration
Resets Half-Pulse Counter to 0; Increment the Rotation Counter by 1

Figure 14. CW P_k Bit Operational Flow Chart.

Next, the simulation will check to see if the half-pulse counter is equal to 399; if so, the half-pulse counter will be reset to 0 and passed on for further simulation processing, as well as the rotation counter will be incremented by 1. Conversely, if the half-pulse counter is between 0 and 399, then the value is simply passed on for further processing without incrementing the rotation counter.

2. Nk Bit Operational Flow Chart

In regards to CCW rotation operations, Figure 15 displays the operational flow chart for N_k bit operations. The N_k bit operations mirror that of P_k operations, however with a couple of differences. Much like the P_k bit operations, the simulation will check to see if the input bit, N_k is a 1 or 0. If the bit is a 1, then the half-pulse counter is decremented by 1, and the Direction Bit is set to 1 and this value is sent to the Speed Indicator Block. If the input is a 0, the ISC is either in a static state or the motor is rotation in the CW direction, and again no action is taken.

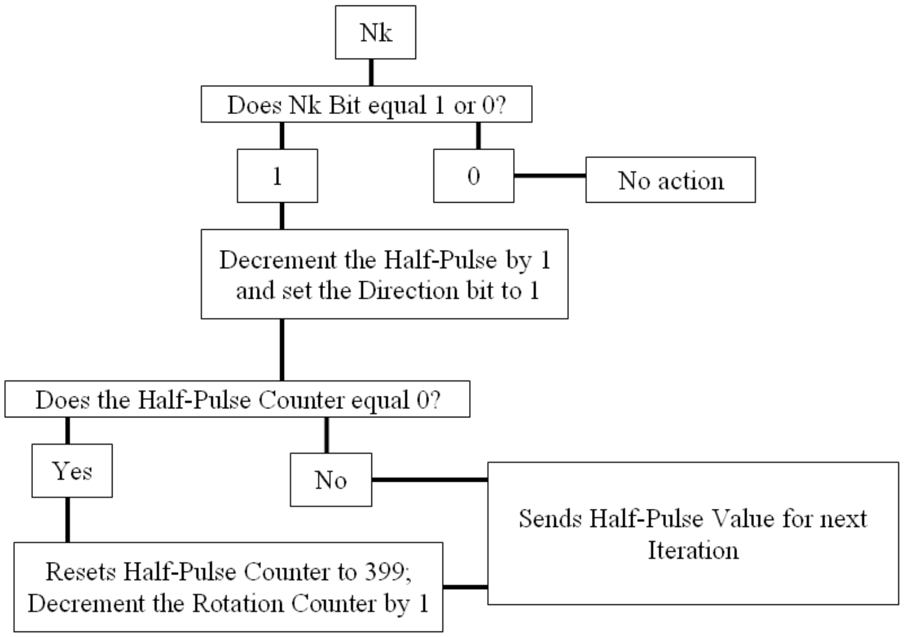

Nk
Does Nk Bit equal 1 or 0?
1
0
No action
Decrement the Half-Pulse by 1 and set the Direction bit to 1
Does the Half-Pulse Counter equal 0?
Yes
No
Sends Half-Pulse Value for next Iteration
Resets Half-Pulse Counter to 399; Decrement the Rotation Counter by 1

Figure 15. CCW N_k Bit Operational Flow Chart.

Moreover, the half-pulse counter is then compared to 0; if the value of the counter is equal to 0, then the counter value is reset to 399. Essentially, this reset to 0 will ensure that the half-pulse counter will never take on a negative value, which is a desired property for proper simulation performance. Lastly, the rotation counter is decremented by 1, and half-pulse counter data is passed on for further simulation processing.

3. 0k Bit Operational Flow Chart

In reference to Figure 16, the program will monitor the 0_k bit. Specifically, it will check to see if the 0_k is 1 or a 0. If a 1 is observed, it simply indicates that 200 full pulses, or 400 half-pulses, have occurred. In other words, 360 degrees of rotation has occurred. On the other hand, if a 0 is observed, then the simulation is allowed to continue normal CW or CCW rotation operations. The simulation requires this last bit to function properly in its current design parameters; meaning all possible states defined in Table 2 must be taken into account. Yet, the 0_k bit pulse width variance proved to be too inconsistent during development to provide accurate rotational data. As a direct result, the decision to exclude the 0_k bit's input when determining rotations count, and half-pulse count was reached. MATLAB code for these flow charts can be seen in Appendix *A*.

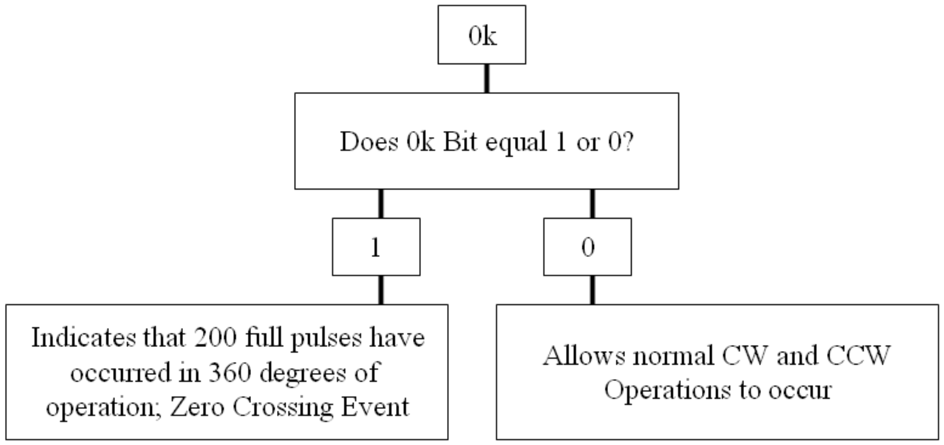

0k
Does 0k Bit equal 1 or 0?
1
0
Indicates that 200 full pulses have occurred in 360 degrees of operation; Zero Crossing Event
Allows normal CW and CCW Operations to occur

Figure 16. 0_k Bit Operational Flow Chart.

I. DETERMINING TOTAL DEGREES TRAVELED

In this section of the simulation, recall that the M-Code Block has two feedback loops that provide the rotation count and the half-pulse count. Also recall that for every 360 degrees of rotation there are 400 half-pulses produced. Therefore, by utilizing a Multiplication Block with a value of 0.9, and multiplying the half-pulse count value by this multiplication block output, the half-pulse count value is converted to degrees traveled. Also note that with a maximum half-pulse count goes from 0 to 399; that means all 400 half pulses are taken into consideration and convert up to 359.1 degrees before the reset to 0. Additionally, the total degrees traveled can also be determined by multiplying the NewRotation Feedback Loop value by a value of 360. This converts the rotations count to degrees of angular position.

Last but not least, the total degrees traveled can be calculated by adding the NewRotation feedback output value to the degrees traveled outputs via the Addition Block. The Physical architecture of the simulation can be seen in Figure 17 and 18, as well as the output waveforms of both feedback loops are seen in the 3 graphics displayed in Figure 19.

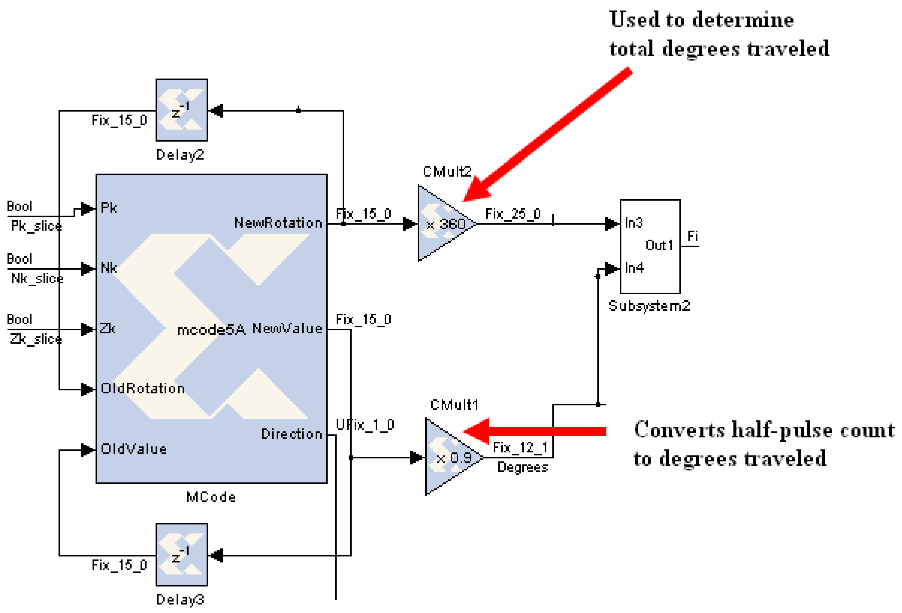


Figure 17. Total Degrees Travel Architecture.

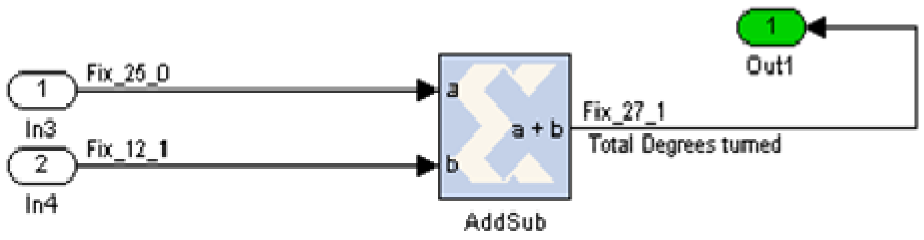


Figure 18. Block Labeled Subsystem2 in Figure 17.

J. SIMULATION OUTPUT RESULTS

From Figure 19, note that from time 0 to 0.001 seconds this particular simulation produced over 6 rotations in the CW direction. When the input was changed to CCW, the simulation produced an additional 12 rotations. This result was confirmed via the middle graphic, which indicated that 2236 total simulated degrees where traveled from 0 to 0.001 second. In other words, the simulation produced 6 full 360 degrees of rotations with an additional 76 degrees in the CW direction. Additionally, in the CCW direction, the simulation produced 12 full 360 degrees of rotations with an additional 169 degrees traveled. Also note that in the middle graphic shows that at approximately 0.002 seconds, the simulated degrees traveled became a negative value; this means that the simulation returned a negative degree traveled count. Lastly, in the bottom graphic, the simulation produced and displayed up to 359.1 degrees of rotation before being reset to 0 for each direction of rotation. In either direct of rotation, the degrees of rotation remained a positive value. This is the exact simulation performance expected and desired, and in accordance with the 3 operational flow charts previously discussed.

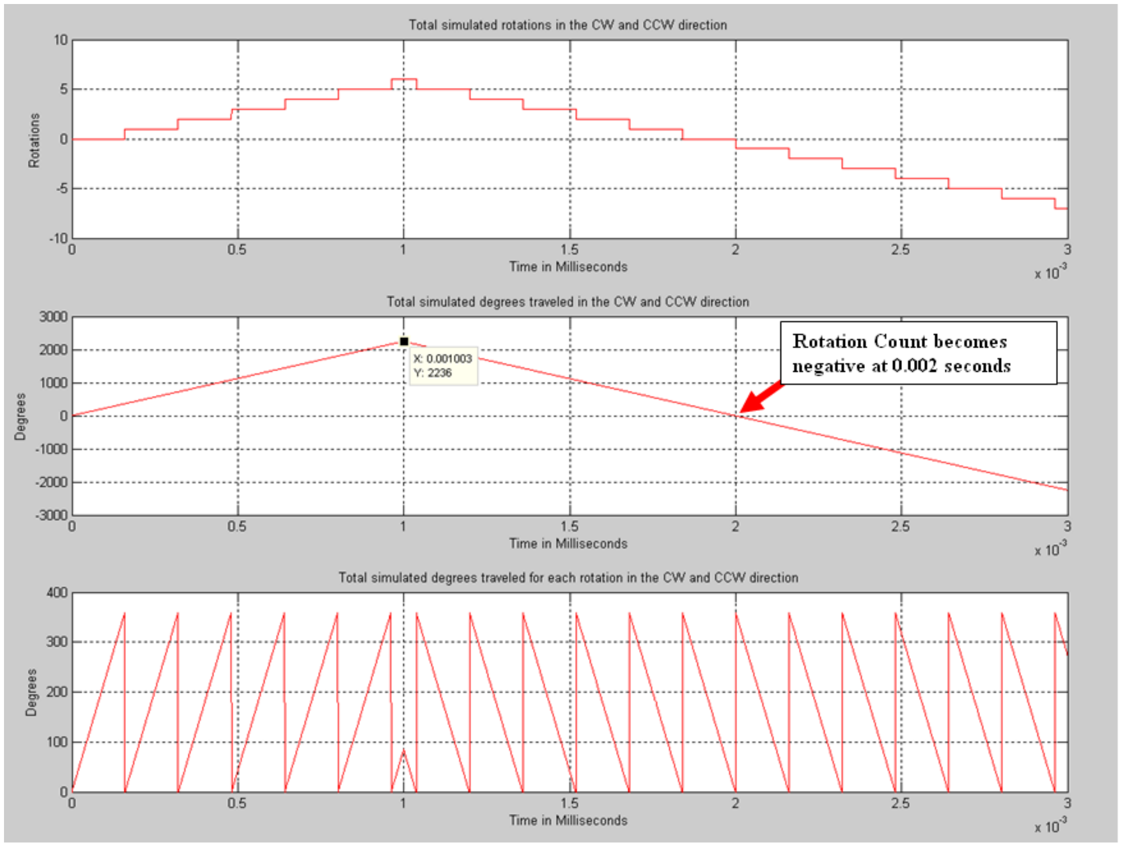


Figure 19. Top Graphic; Total Simulated Rotation Count. Middle Graphic; Total Simulated Degrees Traveled. Bottom graph - Total Simulated Degrees Traveled for each Rotation.

K. CHAPTER SUMMARY

This chapter discussed the FPGA interface with the SDC as well as the ISE in great detail. It went on to emphasize the development of a simulation which mirrored the performance of the ISE source signal, and discussed the simulations reproduction of the 16 possible states. Next, the simulation had to be setup in such a manner that the encoder's output signal reflected the motor's revolution data and not the encoder's data. Lastly, it concluded with a review of the simulation output results and confirmed the validity of the simulation model. Chapter IV will explore the development and implementation of the Rotor Speed Indicator.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ROTOR SPEED INDICATOR

A. TACHOMETER OVERVIEW AND INPUT SIGNAL DEVELOPMENT

The most basic type of incremental encoder is a tachometer which has just one output and is most often used in unidirectional applications that track only position or speed information. Specifically, a tachometer is described as an instrument that measures the rotation speed of a shaft or disk, as in a motor or other machine. The device usually displays the revolutions per minute (RPM) on a calibrated analog dial, but digital displays are increasingly common [8].

In order to create a tachometer using the ISE, both the A_k and A_{k-1} input signals were utilized. Recall that the A_k signal is a square wave with set amplitude of 1 and a variable period relative to the speed of rotation. Also recall that the A_{k-1} signal is the previous sample of the A_k signal, sampled just one clock cycle earlier. From Figure 20, one can see that there are 3 inputs to the Speed Indicator Block, two of which are taken directly from A_k and A_{k-1}, and one input is supplied by the M-Code Block and labeled Direction.

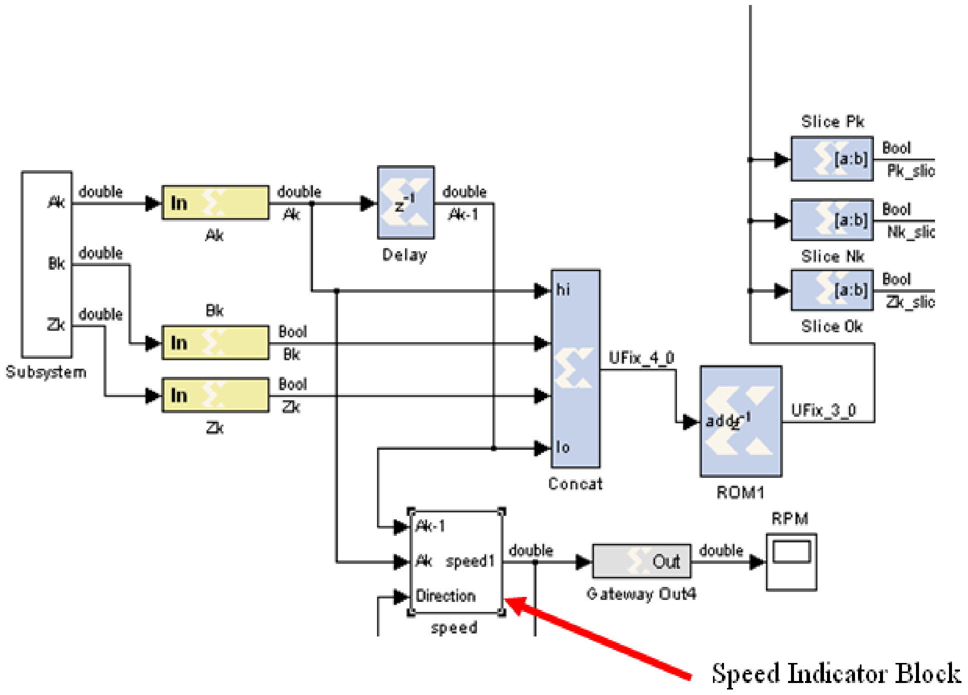


Figure 20. Speed Indicator Block.

In order to understand how this tachometer functions, we will break down the Speed Indicator Block into small describable sections and cover each section in great detail. Figure 21 displays the output wave form produced by the ISE for both the CW and CCW rotation. Note that the solid red line represents the A_{k-1} sample; the dotted blue line represents the current signal sampling point or A_k. Lastly, for clarification purposes, the arrows were added to the figure to indicate the direction of sampling.

In regards to the CW rotation the encoder will produce a 1 or H, followed by a 0 or L. In the CCW direction it will produce a 0 followed by a 1. When either pattern is observed, it simply indicates that rotation has occurred.

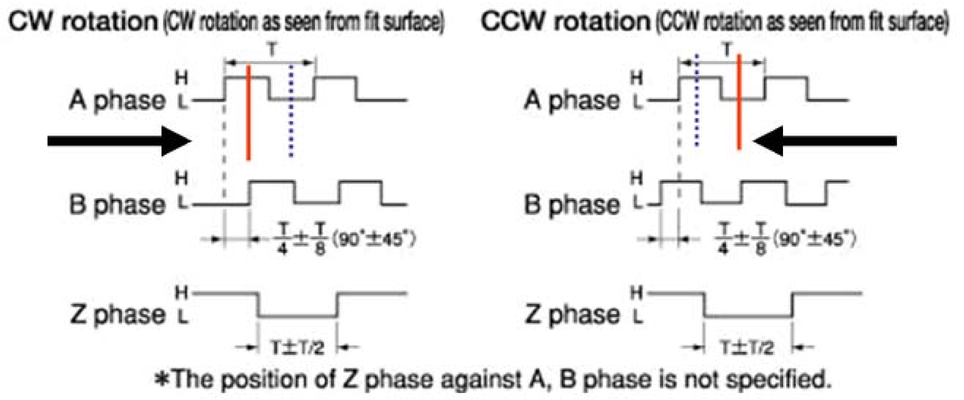

CW rotation (CW rotation as seen from fit surface)
CCW rotation (CCW rotation as seen from fit surface)
T
A phase
H
L
B phase
H
L
T/4 ± T/8 (90°±45°)
Z phase
H
L
T±T/2
*The position of Z phase against A, B phase is not specified.

Figure 21. ISE A Phase Input Signal. [After [5]].

B. SPEED INDICATOR BLOCK SIMULATION DEVELOPMENT

In reference to Figure 22, we next examine the internal working of the Speed Indicator Block. As mentioned earlier, the inputs to this block are Direction, A_k and A_{k-1}. First let's focus attention on the inputs A_k and A_{k-1}; both inputs are fed directly into the logical XOR Logic Block with its time delay set to zero. The output of the XOR Block is either a 1 for the input 01 or 10, or a 0 for the input 00 or 11. In layman's terms, every time the signal goes from H to L or L to H, a 1 will be output downstream of the XOR Block, otherwise a 0 will be the output, thus indicating that the motor has rotated one-half pulse width. Note that a total of 400 half-pulses per rotation is produced using the ISE and for each full pulse produced there is a rotation of 1.8 degrees.

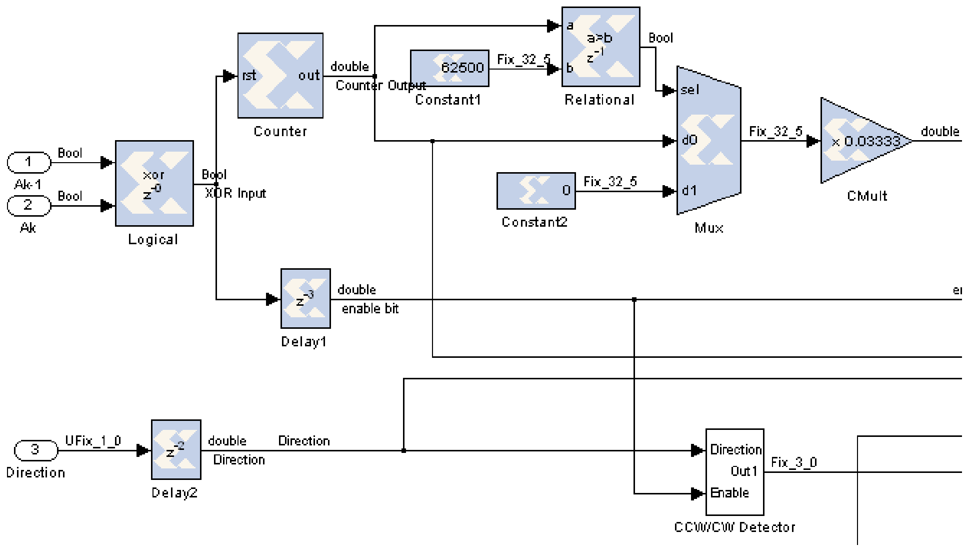


Figure 22. (Partial) Speed Indicator Block Diagram.

Next, looking down stream of the XOR block one can see that the output is then tallied via a Counter Block. This block implements a free running or count-limited type of an up, down, or up/down counter. The counter output can be specified as a signed or unsigned fixed point number. The output of the counter, in relation to the XOR output, can be seen in Figure 23.

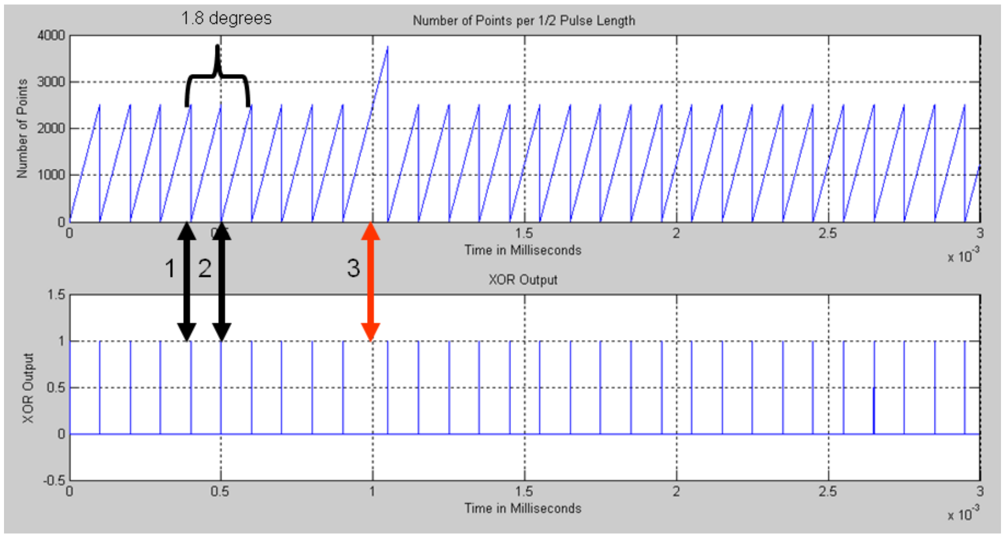


Figure 23. XOR Number of points/ ½ Pulse Length/Time; XOR Output/Time.

In Figure 23, the red double-headed arrow number 3 indicates that the simulation changed directions from CW to CCW and record some additional clock cycles before a reset to zero. Also note the position of the two black double headed arrows numbered 1 and 2; these arrows indicate that the counter counted up to 2499 clock cycles, then resets to 0 each time the XOR output became 1.

1. Zero Revolution Detector

In Figure 24 the area boxed by the dotted red border is the section of the simulation that checks to see if the device has a rotation rate greater then or less then 10 revolutions per minute. Why 10 revolutions per minute? A zero rotation point had to be established that was near a zero but not actually zero. So the value of 10 RPM was specifically selected to represent zero rotations. With this parameter in place, if the simulation was less then 10 revolutions per minute, then output of the Multiplexer Block (Mux) would be considered zero, and consequently no rotation will be registered. On the other hand, if the rotation is greater then 10 revolutions per minute, then the output of the counter will be the output of the Mux Block, and its value will be passed on for further processing.

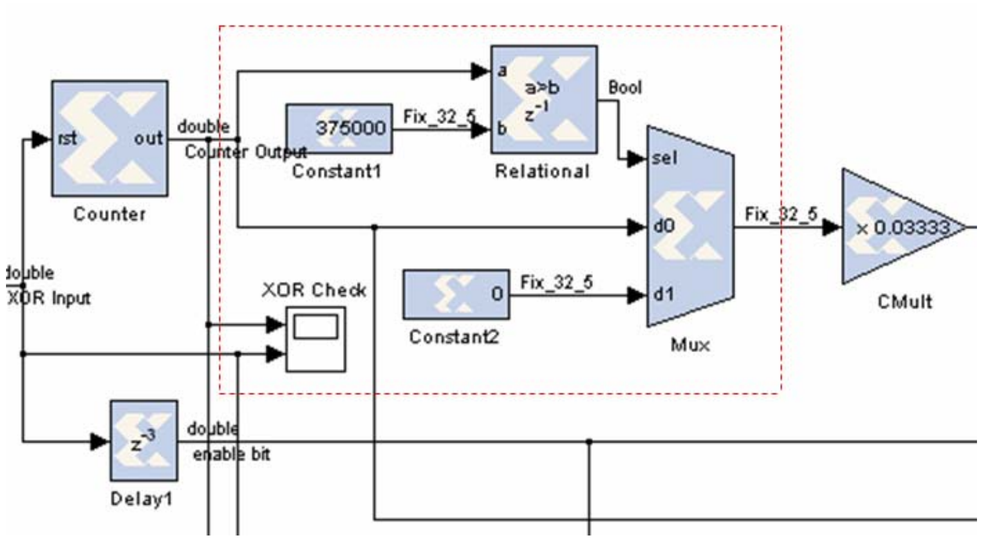


Figure 24. Speed Indicator Rotation Check.

How does the above operation occur? Through the use of a Relational Block, the simulation was able to compare two different values and output one value in accordance with the specific rational operator used. In this case, the simulation compared a constant value (channel “a”) against the counter output (channel “b”) using the rational operator a>b. For example, for 10 revolutions to occur in one minute at a t_square value of 0.0002, the number of clock cycles would exceed 375000. Therefore, the simulation compares the value of 375000 to that of the output of the counter. If a>b, then the output will be a Boolean 1, and trigger the Mux block to output the constant value assigned to channel d1; in this case a 0 would be passed. On the other hand, if a<b, then the answer is false, and a Boolean 0 would force the Mux block to output the counter value assigned to channel d0; and consequently the counter value would be passed.

2. Multiplication Operations

Down stream of the Mux Block is a Multiplication Block, labeled CMult. This block was added in order to scale down the input value supplied by the Mux block by 0.0333. Scaling down the Mux output was done to ensure that the new Mux output value did not exceed the LUT called “Reciprocal” maximum value. Values of LUT Reciprocal can be seen in Figure 25 and the MATLAB code can be found in the Initial Conditions File seen in Appendix *A*.

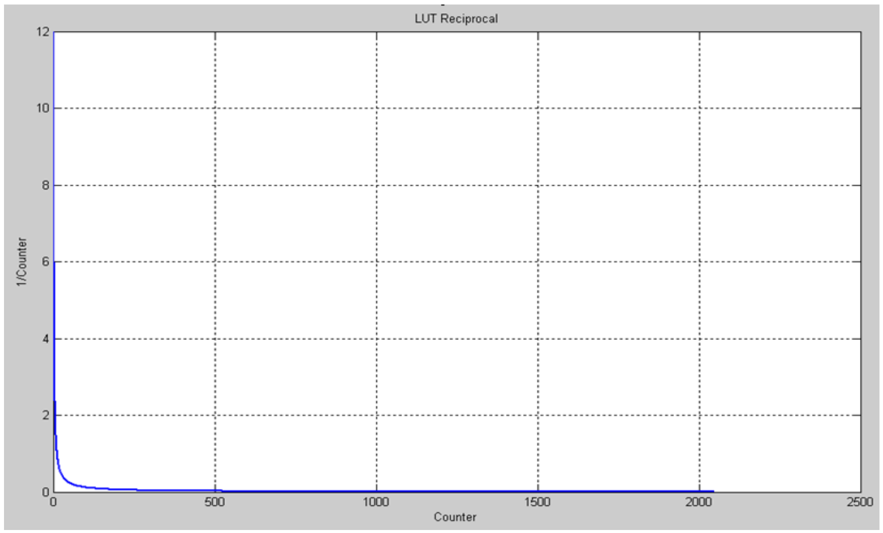


Figure 25. Look Up Table "Reciprocal."

Furthermore, the Multiplication Block (CMult) implements a *gain* operator, with output equal to the product of its input by a constant value. In this simulation, the model used a Multiplication Block set to a 16 bit word length with the binary point set at the 14th bit; this provided the precision required for the value 0.03333 to be used. The circuit architecture can be seen in Figure 26.

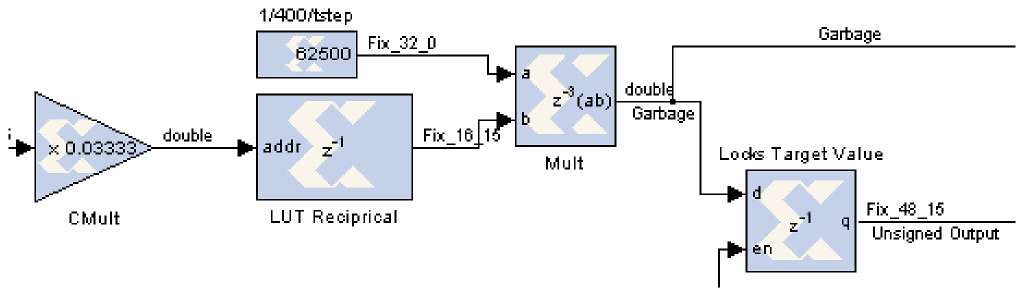


Figure 26. CMult /LUT Reciprocal Block.

In regards to the LUT Reciprocal, this block was specifically implemented due to the fact that performing a division operation is just too difficult a process for the computer system to perform effectively. Consequently, by providing the scaled down reciprocal version of the Mux output, then multiplying the LUT output values by the constant value 625000, the program was able to effectively process the total degrees traveled. Note that the Mux output reduction by 0.0333 is later restored to its actual value via the LUT's base equation. This result was then scaled from total degrees traveled per second to RPM by multiplying the output by 0.1667. Simulated results can be seen in Figure 27.

In Figure 27 (top graphic) note that 99.9% of the data output by the Multiplication Block is inaccurate. Therefore, a Register Block was implemented in conjunction with Delay Block that was finally linked to the XOR output. This effectively allowed the simulation to "activate" the Register Block at precisely the correct moment in time to capture the data that was considered correct (bottom graphic); this allowed accurate RPM data to be passed downstream for further processing. The black arrows in Figure 27 highlight the points where the data is considered accurate. It is at these points alone that the data is captured and passed. The remaining data seen in Figure 27 (top graphic) appears to be wrapping due to numerical saturation. Consequently, this data is not considered accurate and can not be used. Specifics on how the program coverts degrees traveled to RPM will be discussed in the following section.

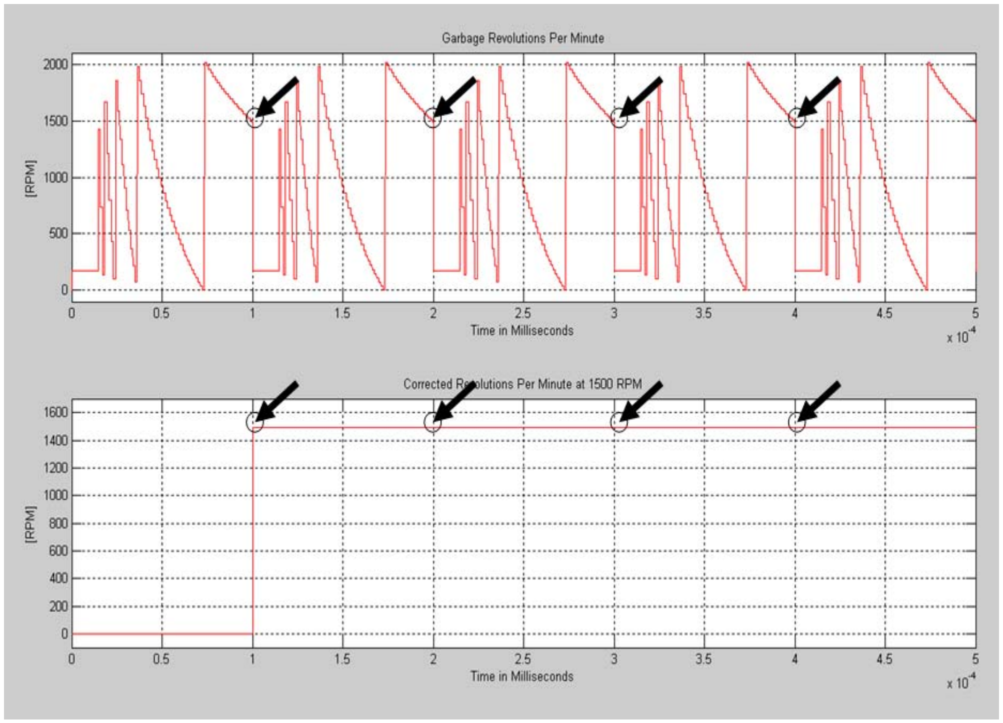


Figure 27. Garbage RPM (top)/Corrected RPM Output (bottom).

3. Converting Raw Data to RPM

Now that the total amount of degrees traveled is a known quantity, by inserting an additional Multiplication Block with a value of 0.1667 downstream of the Register Block, Degrees per Second was scaled to reflect RPM. Originally this simulation was set up to produce an output of 1500 RPM. However the actual value obtained via simulation was 1488 RPM. As a result, there was a 0.8 % simulated speed estimation error produced. This error could be due to rounding error in the program, yet the error is small enough to be considered a good estimation of the actual simulated speed. Results can be seen in Figure 28. From Figure 28, note that during time of transition from CW to CCW or vice versa, speed estimation is not considered accurate and should not be considered.

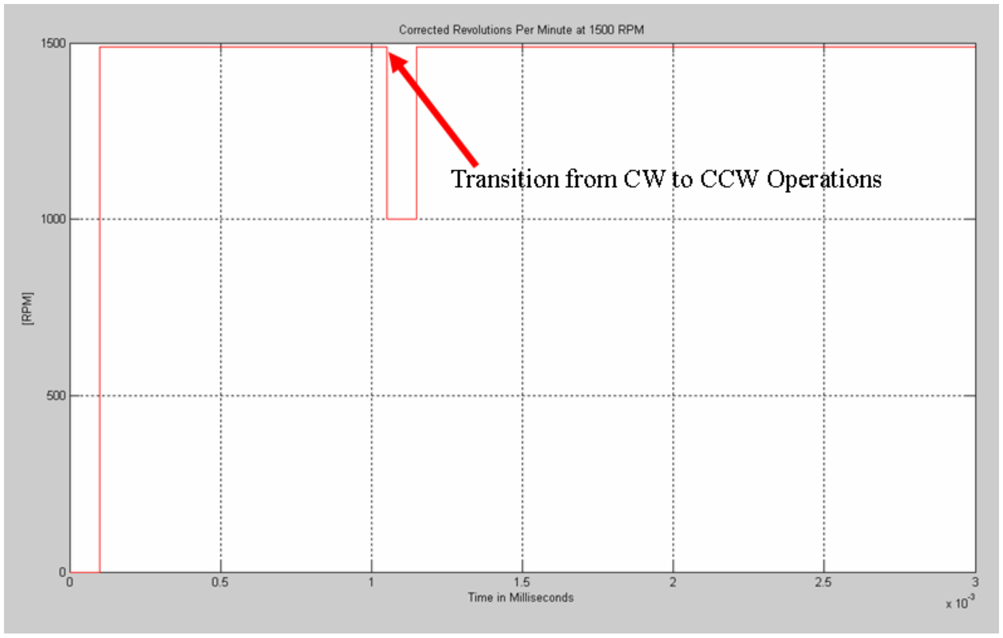


Figure 28. Speed Indicator Simulated Output Results at 1500 RPM.

4. Direction Bit Operation

At this point in the simulation, the speed of a device can be accurately measured with a high degree of precision. However, the direction of the device still can not be determined without additional programming; therefore, a Direction Bit was added to the simulation. Recall that one of the three inputs to the Speed Indicator Block was the Direction Bit. This bit was designed to input a 0 if the device was rotating in the CW direction and a 1 if the device was rotating in the CCW direction. Simulation architecture can be seen in Figure 29 and highlighted by the red arrow.

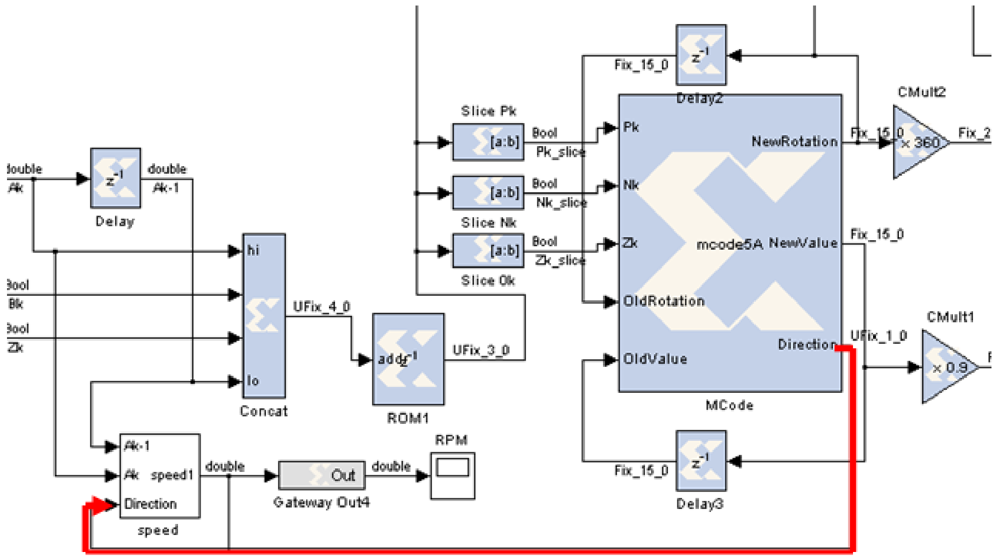


Figure 29. Speed Indicator /M-Code Block Interface.

Now, referring to Figure 30, the input labeled Direction is fed directly into the CCW/CW Detector Block. This input utilizes a Delay Block with a latency of 2 in order to ensure correct timing of the simulation inputs and also synchronizes its input with that of the XOR Output Enable Bit, which also feed into the CCW/CW Detector Block.

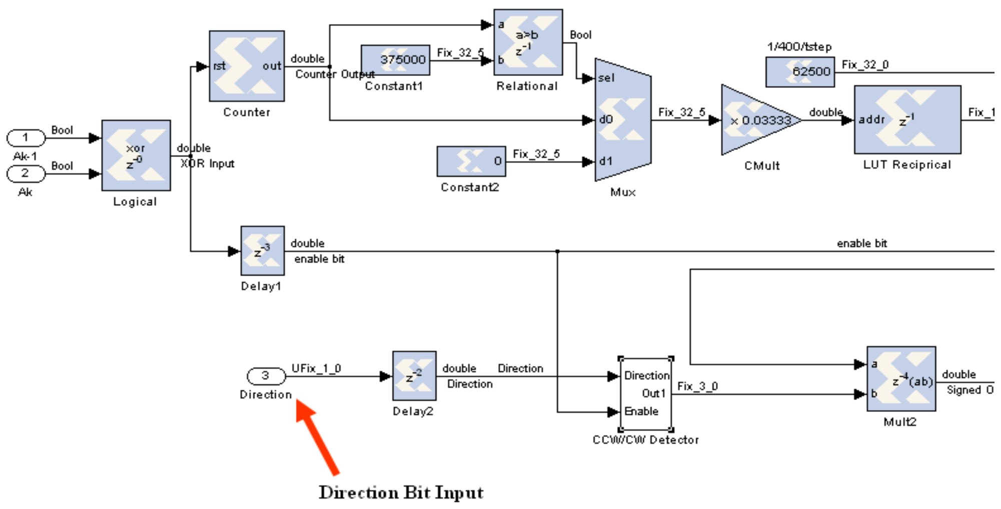


Figure 30. Direction Bit Input Interface with the CCW/CW Detector Block.

5. CCW/CW Detection Block

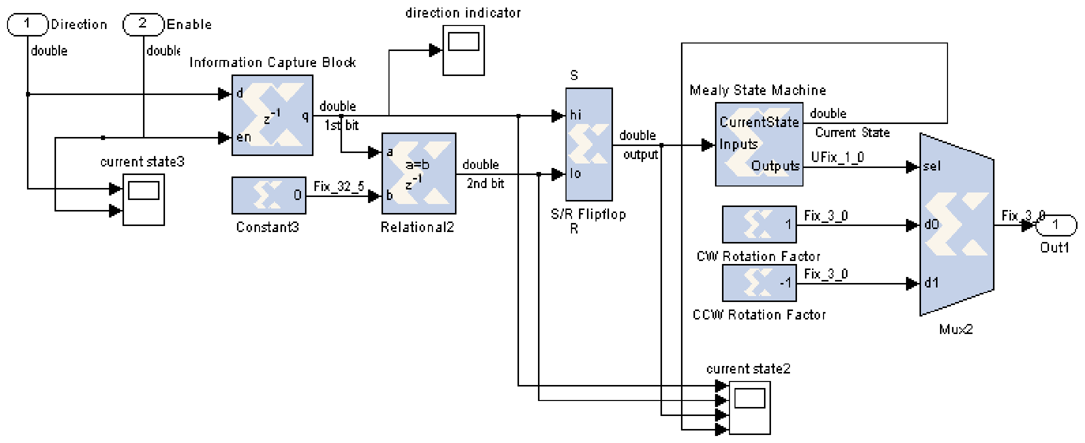


Figure 31. CCW/CW Detector Block Physical Architecture.

From Figure 31, one can see that the Direction Bit and the XOR Enable Bit are the inputs to the Register Block, labeled Information Capture Block (ICB); again a Register Block was utilized to captures the data at precisely the correct moment in time when the input is meaningful to the program. Once this data is collected and allowed to pass, the output of the ICB is compared to 0 via the Rational Block with the logic a=b. Essentially what this means is that if the output of the ICB is equal to 0, then the constant 0 is passed; otherwise a 1 is passed; it is simply a true or false output. The output of the Rational Block is then compared in an S/R Flip Flop. This particular S/R Flip Flop was implemented by a Mealy State Machine. The truth table can be seen in Table 3 below. This particular truth table has its current state values on the left side of the table, and the values for S and R along the top row as two concatenated bits.

S R	00	01	10	11
0	0	0	1	0
1	1	0	1	0

Table 3. Truth Table for SR Flip Flop.

Since the S/R flip flop is implemented with a Mealy State Machine, the output is a function of the current state and the inputs. The inputs are the values for S and R, and the current state is either 0 or 1. For example, if the current state is 0 and the current value of the output is 0, and the input for S is 1 and R is 0, then the output will become 1, as seen above, and the next state will be state 1. As long as the R input bit is not a 1, the output will remain a 1, and the current state will remain one. Simulation architecture can be seen in Figure 32.

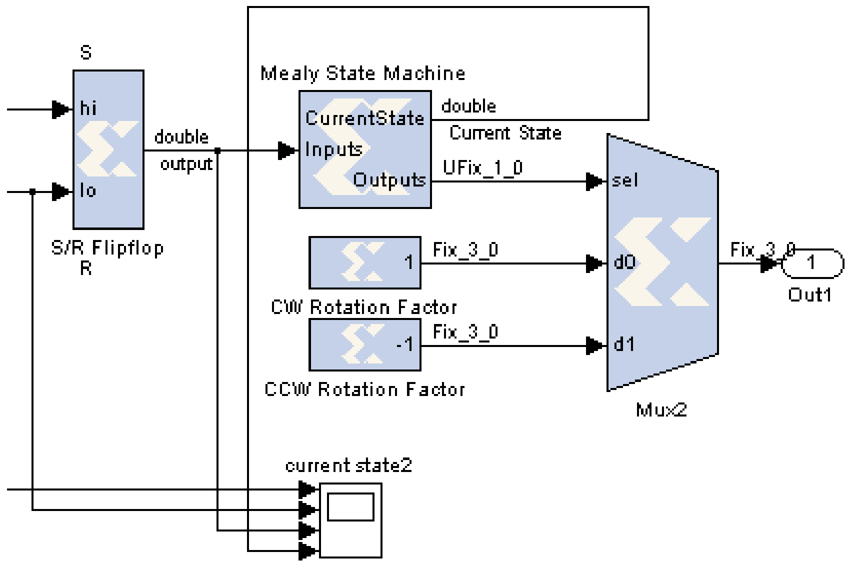


Figure 32. Mealy State Machine Direction Architecture.

As previously mentioned, the output of the Mealy State Machine is either a 1 or 0. Therefore, utilizing a Bus Multiplexer (Mux 2), the simulation will either output the user defined constant value of 1 or a negative 1 depending on the output of the Mealy State Machine. In turn, the output of Mux2 will be used to calculate and display positive or negative rotation of the device by multiplying the Mux2 output by the CMult2 Block previously mentioned. As a direct result, the simulation can now display positive or negative RPM. The Physical architecture can be seen in Figure 33.

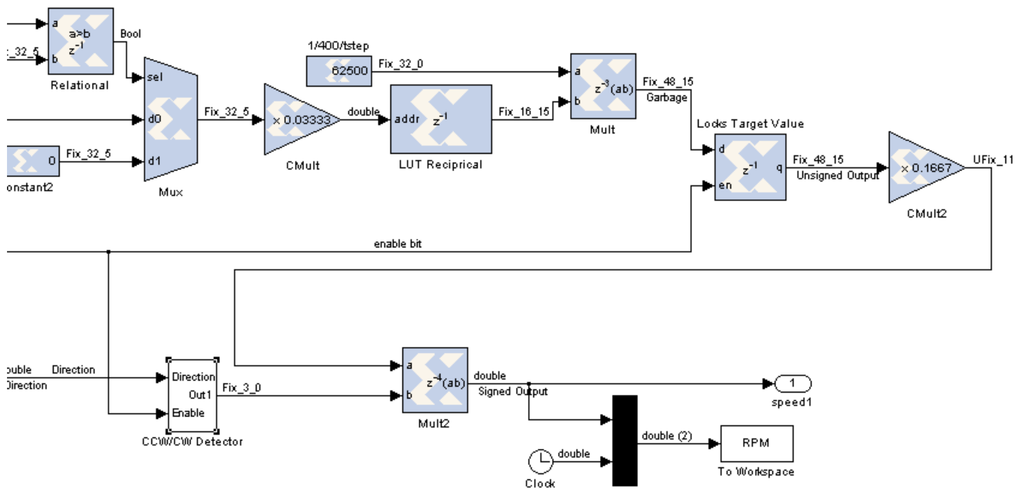


Figure 33. Speed Indicator Block Output.

C. CHAPTER SUMMARY

This chapter presented an overview of a tachometer and how the *A* Phase produced by the ISE was utilized as an input in the construction of the Rotor Speed Indicator. The next topic discussed was how a tachometer was reproduced in simulation, which highlighted specific areas of simulation development, such as Zero Revolution Detection, as well as signal pre and post-processing. Lastly, it explained how the simulation was developed in order to detect and register both CW and CCW rotation. Chapter V describes the digital interface between the hardware and software utilized in signal processing.

THIS PAGE INTENTIONALLY LEFT BLANK

V. HARDWARE AND SOFTWARE INTERFACE

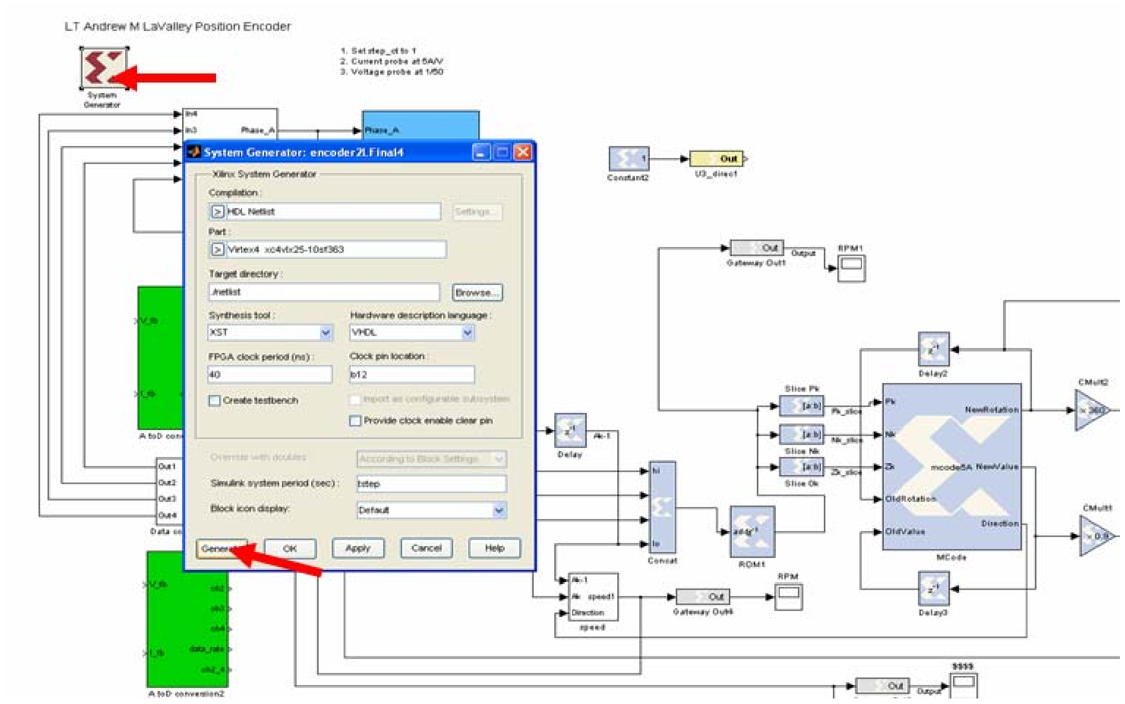
A. HARDWARE AND SOFTWARE INTERFACE INTRODUCTION

The SDC utilizes Simulink software for modeling power electronics systems as well as running simulations to test engineering power designs. Simulink provides an environment for multi-domain simulation and Model-Based Design for dynamic and embedded systems. Moreover, it provides an interactive graphical environment equipped with a customizable set of block libraries that let the user design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing. Lastly, Simulink enables model analysis and provides the diagnostics tools necessary to ensure model consistency and identifies modeling errors prior to hardware setup and testing [9].

As mentioned earlier, XILINX Foundation software generates VHDL code once the simulation or model is perfected. In turn, the software is then used to generate code that can be used to program the FPGA. Again, being proficient in VHDL programming is not a requirement; however understanding the procedures that make it possible to interface the hardware with the software are essential for signal processing and will be discussed in greater detail in the following section.

B. GENERATING VHDL CODE USING ISE FOUNDATION

Once the model is operating as designed it will then be used as a design template for VHDL code generation. The first step in generating VHDL code is to simply click the System Generator Block located in our XILINX Foundation software model. This will open up the System Generator User Interface Menu Block. Screen shot can be seen in Figure 34.



System Generator: encoder2LFinal4

Xilinx System Generator

Compilation : HDL Netlist

Part : Virtex4 xc4vlx25-10sf363

Target directory : ./netlist

Synthesis tool : XST

Hardware description language : VHDL

FPGA clock period (ns) : 40

Clock pin location : b12

Create testbench

Import as configurable subsystem

Provide clock enable clear pin

Override with doubles : According to Block Settings

Simulink system period (sec) : tstep

Block icon display: Default

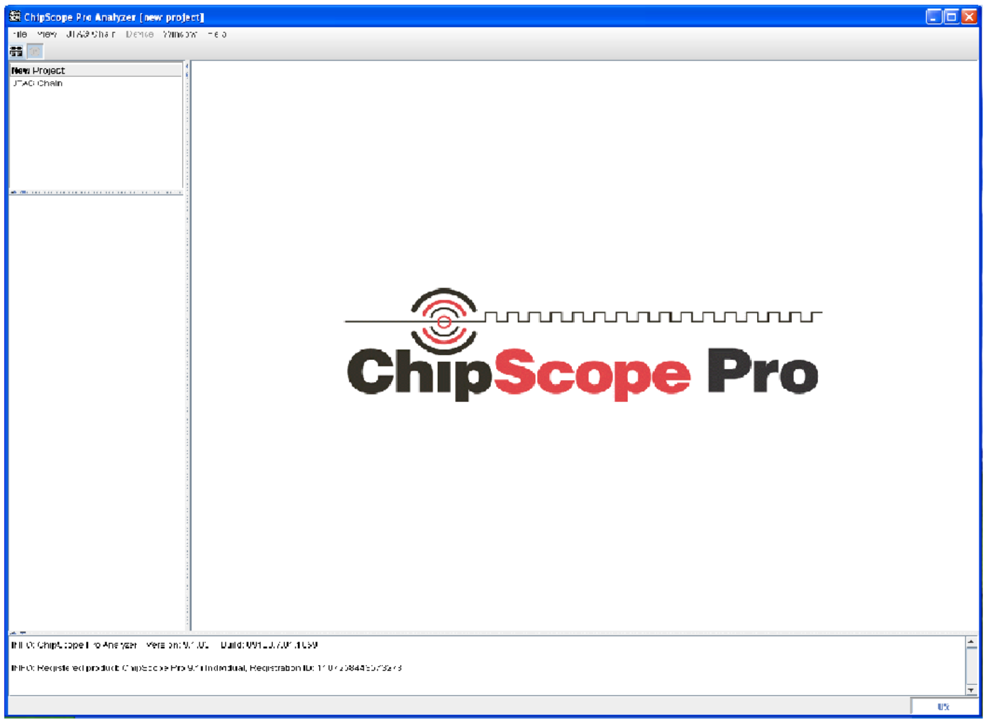
Figure 34. System Generator Block and Menu.

From the System Generator User Interface Menu Block enter the required data fields, and then click the “Generate” button. The Model is then compiled, and made available for further processing. Furthermore, using ISE Foundation software, the newly compile file is then utilized to generating the program file associated with the particular model developed. More specifically, once synthesized, a programming file is generated using the ISE Program Navigator software by opening the newly compiled file from a drop down list, then clicking on Generate Program File. This automatically starts the process that will generate and configure the device. Once this process is complete, the programming file is generated and the FPGA is then programmed.

C. CHIPSCOPE™ INTERFACE

Next, the user can control a converter remotely through ChipScope™ Pro embedded software. More specifically, ChipScope Pro is a tool that inserts a logic analyzer, bus analyzer, and virtual I/O low-profile software cores directly into your design. It allows the user to view any internal signal or node, including embedded hard or soft processors at or near system operating speed. Additionally, through the utilization of ChipScope Pro Logic Analyzer, the program can take the recorded data and not only view its contents but analyze it for further post-processing.

ChipScope Pro is initially opened from the ISE Foundation Window. The control screen shot is shown in Figure 35.



```
INFO: ChipScope Pro Analyzer   Version: 9.1.0[illegible]   Build: [illegible]
INFO: Registered product: ChipScope Pro 9.1 Individual, Registration ID: [illegible]
```

Figure 35. ChipScope™ Pro Start up Control Screen.

Once the ChipScope™ Pro software is up and running, the user must first establish communications with the hardware by providing a connection between the JTAG Chain and the FPGA. Figure 36 shows how the simulation interfaces with the ChipScope™ software via the ChipScope™ Interface Block.

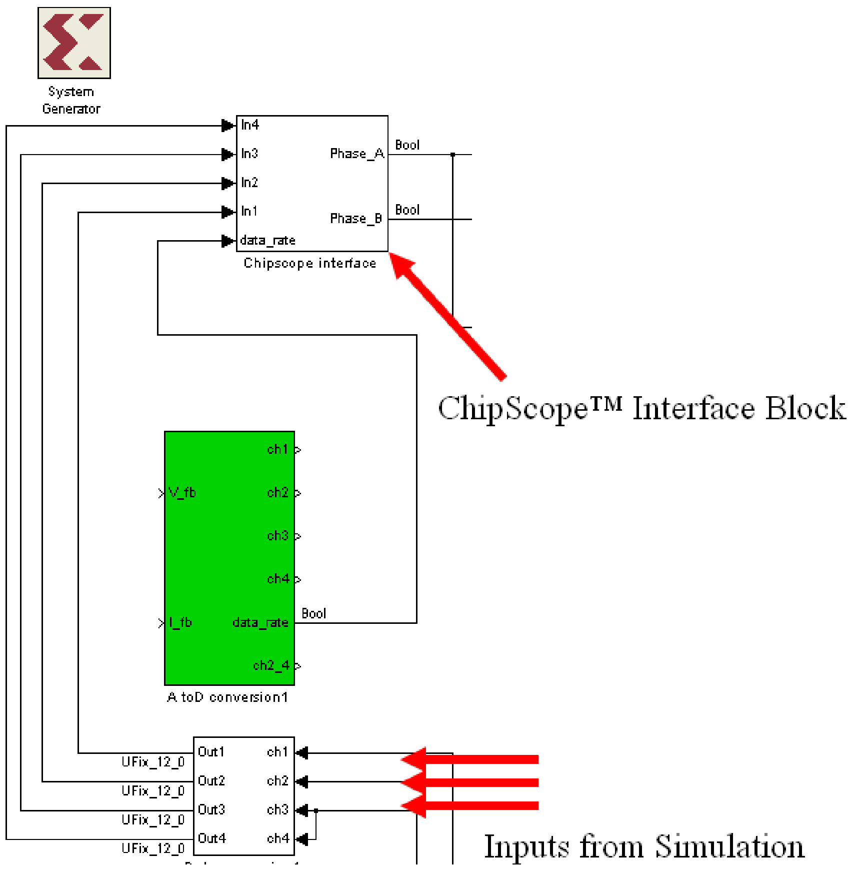
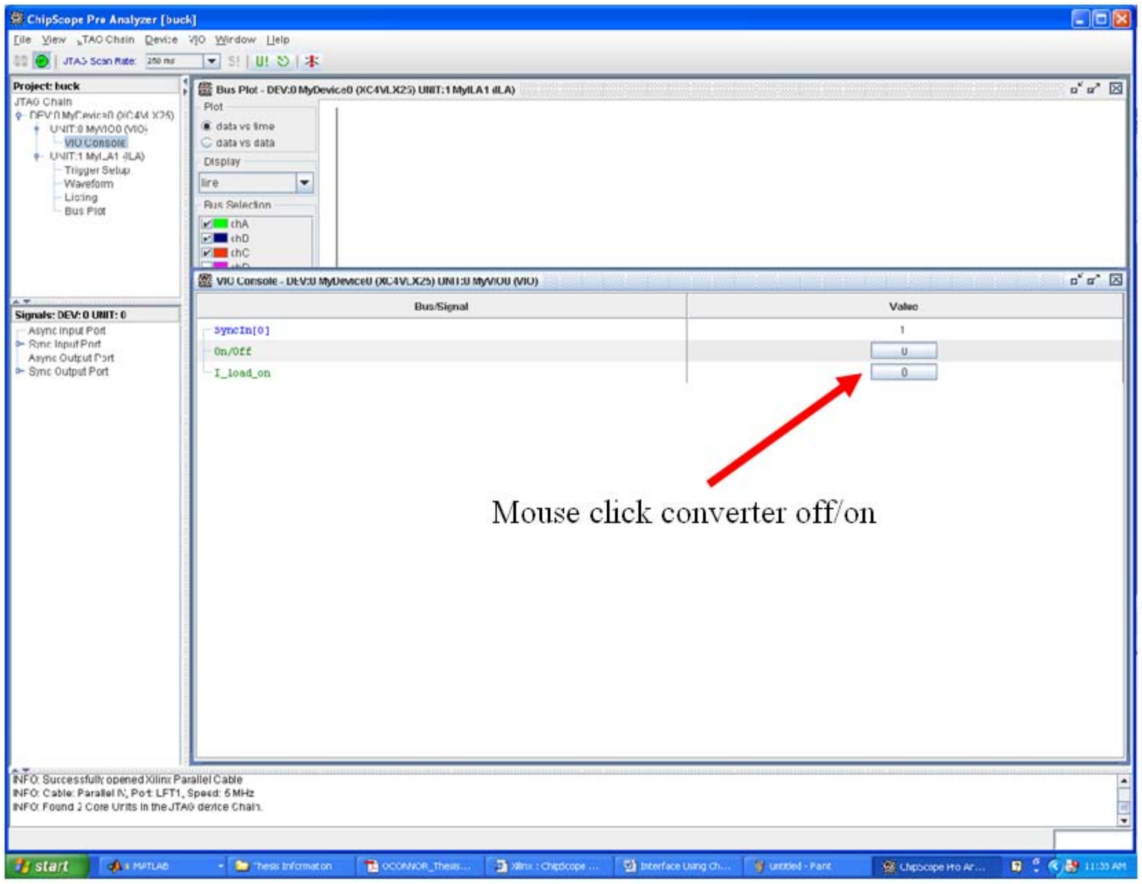


Figure 36. ChipScope™ Interface Block/ Input from Simulation.

As shown in Figure 37, under the drop down menu “File,” select the file that you wish to view as well as the FPGA programming you wish to utilize. Next, select “Devices” tab then highlight the particular device you are interested in, and then click on Configure. From this window select the newly generated VHDL code you wish to utilize then hit the “Ok” button. The device is now ready to read via ChipScope™ Pro.

Once this interface is complete, the user can utilize special feature to process input and output data. For example, the VIO Console Interface Screen is an application that allows the user to manually control the hardware. The screen shot of the VIO Console Interface page can be seen in Figure 37.



Bus/Signal	Value
SyncIn[0]	1
On/Off	0
I_load_on	0

```
INFO: Successfully opened Xilinx Parallel Cable
INFO: Cable: Parallel IV, Port: LPT1, Speed: 5 MHz
INFO: Found 2 Core Units in the JTAG device Chain.
```

Figure 37. VIO Console Interface Screen Shot.

In Figure 37, note that, in this case, the buck converter can be either toggled on or off with a simple click of the mouse. With this feature in place, the user can control the FPGA as well as the VSC digital control process via ChipScope Pro software. Additionally, this interface allows for a detailed digital analysis of input and output signals without instruments. Last but not least, it provides the user with a tool to conduct bit-by-bit evaluation when a more detailed analysis is required.

Yes, the VIO interface page in ChipScope Pro software is a powerful tool, but the use of the VIO Console Interface will not be necessary for analysis of the input signals in this particular case. Instead, the Bus Plot feature was utilized for all signal post-processing. From Figure 38, notice that ChipScope Pro was able to display all 3 of the output signals generate from the ISE. In this particular instance, a Squirrel Cage Induction Motor (SCIM) was used to produce a CW rotation at 1400RPM. The green waveform represents the actual RPM the encoder is rotating during this finite window of time. The red waveform represents the Degrees Traveled per Rotation, and, lastly, the blue waveform represents Total Degree Traveled. Note that the blue waveform is a 12 bit binary word; therefore it can only reach a maximum value of 4096 before it rolls over to 0 due to numerical saturation. Moreover, these three waveforms will then be passed on for further processing and display.

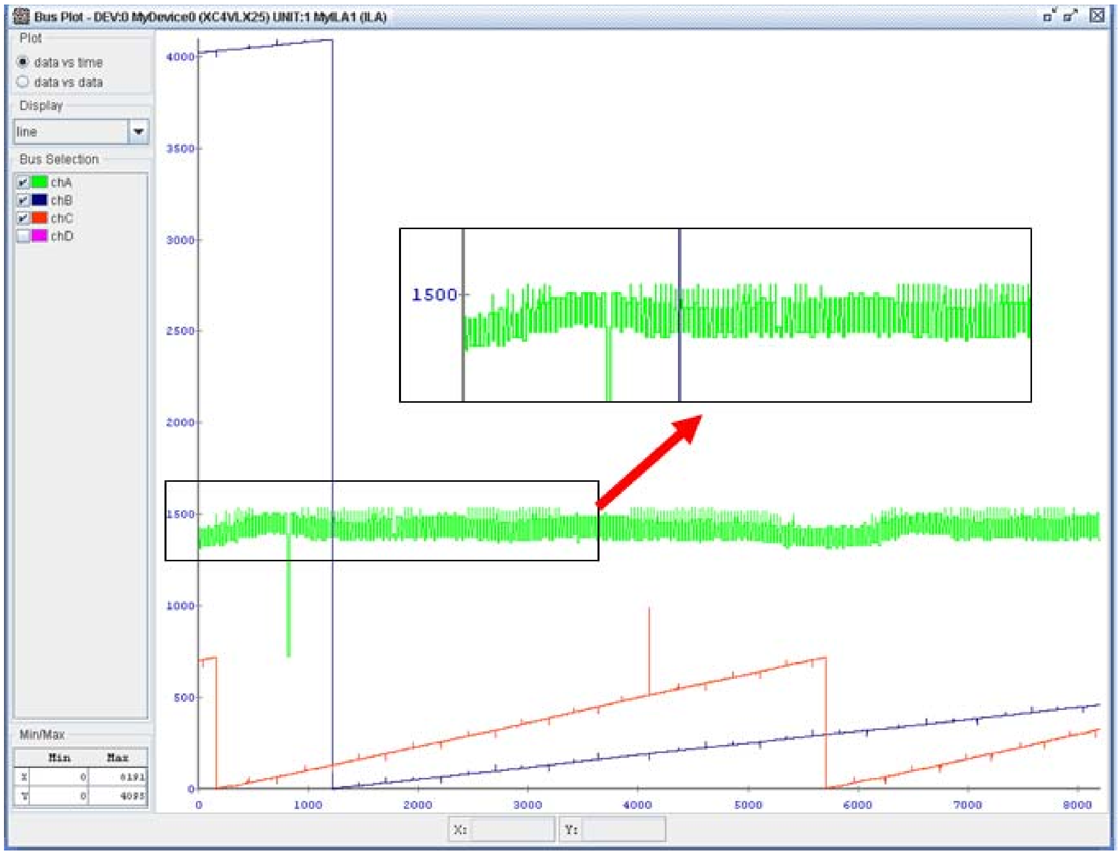
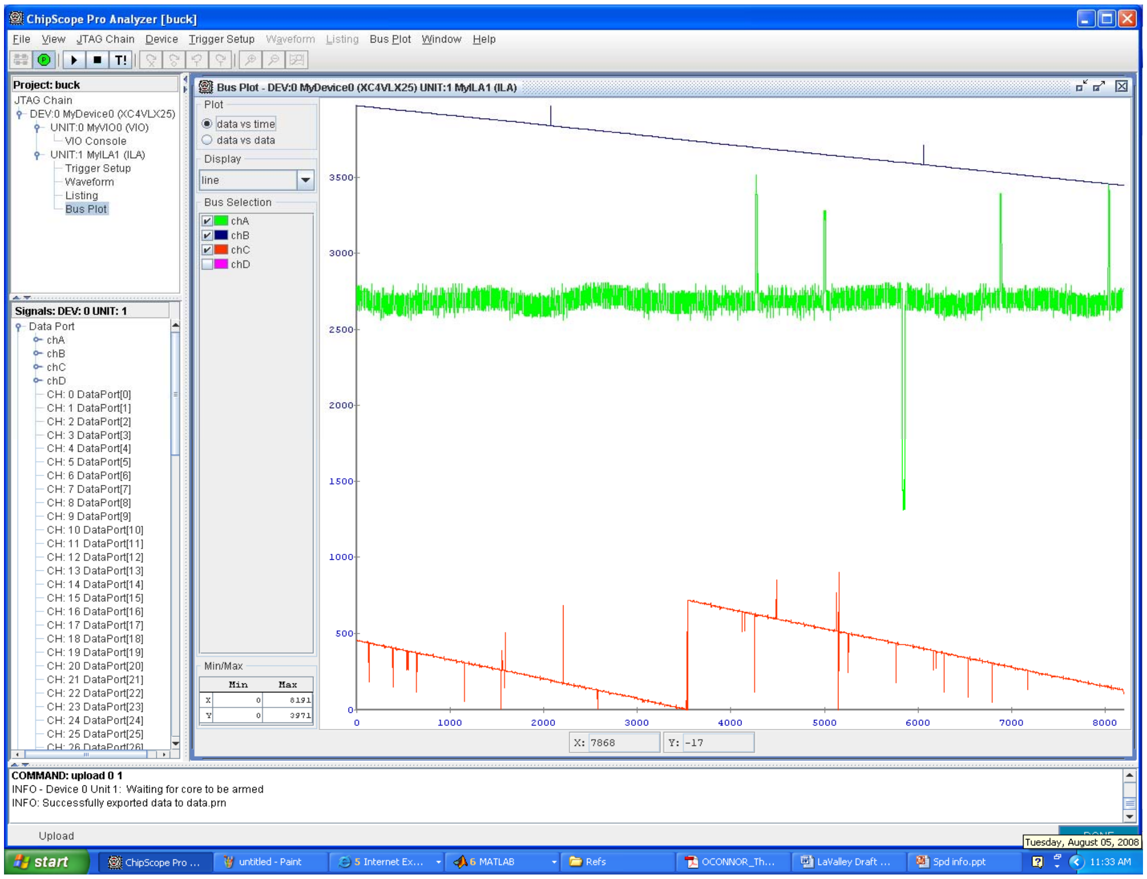


Figure 38. Bus Plot of a Squirrel Cage Induction Motor at 1400RPM (CW).

When ChipScope Pro was used to record and analyze data from a CW rotating motor, ChipScope output display worked excellent. Yet, when the motor rotated in a CCW direction, erroneous data was observed. The problem did not stem from the design of the simulink software; if that were the case, one would expect similar results in the simulated runs. Instead, every simulation performed as expected and consequently the fault had to be in ChipScope Pro software. As it turned out, ChipScope Pro does not possess the capability to accurately process 2’s compliment binary numbers. With this in mind, additional processing of the output signal was required. For example, Figure 37 displays the screen shot of the output of a Squirrel Cage Induction Motor (SCIM) traveling in the CCW direction at approximately -1400RPM. If ChipScope Pro possessed the capability to process 2's compliment binary numbers, then we would expect the green waveform to represent -1400RPM; yet again, this was clearly not the case and a rotation rate of approximately 2700RPM was recorded.



Plot: ● data vs time ○ data vs data

Display: line

Bus Selection:
- ✓ 🟩 chA
- ✓ ■ chB
- ✓ 🟥 chC
- ☐ chD

	Min	Max
X	0	8191
Y	0	3971

X: 7868 Y: -17

```
COMMAND: upload 0 1
INFO - Device 0 Unit 1:  Waiting for core to be armed
INFO: Successfully exported data to data.prn
```

Figure 39. Bus Plot of a Squirrel Cage Induction Motor at -1400RPM (CCW).

To correct this error when taking CCW measurements, the data collected via ChipScope Pro software was exported to an external program for supplementary signal processing. More specifically, the data was exported into a MATLAB m-file that was specifically created to evaluate each and every data point collected, then converted the 2's compliment data into a correct rotation values in either CW or CCW direction. The

MATLA code for this application can be seen in Appendix *A*. The results can be seen in figure 40. Note in Figure 40 that the post-processed value of 2700RPM is now correctly reflecting the actual CCW rotation observed at -1400RPM.

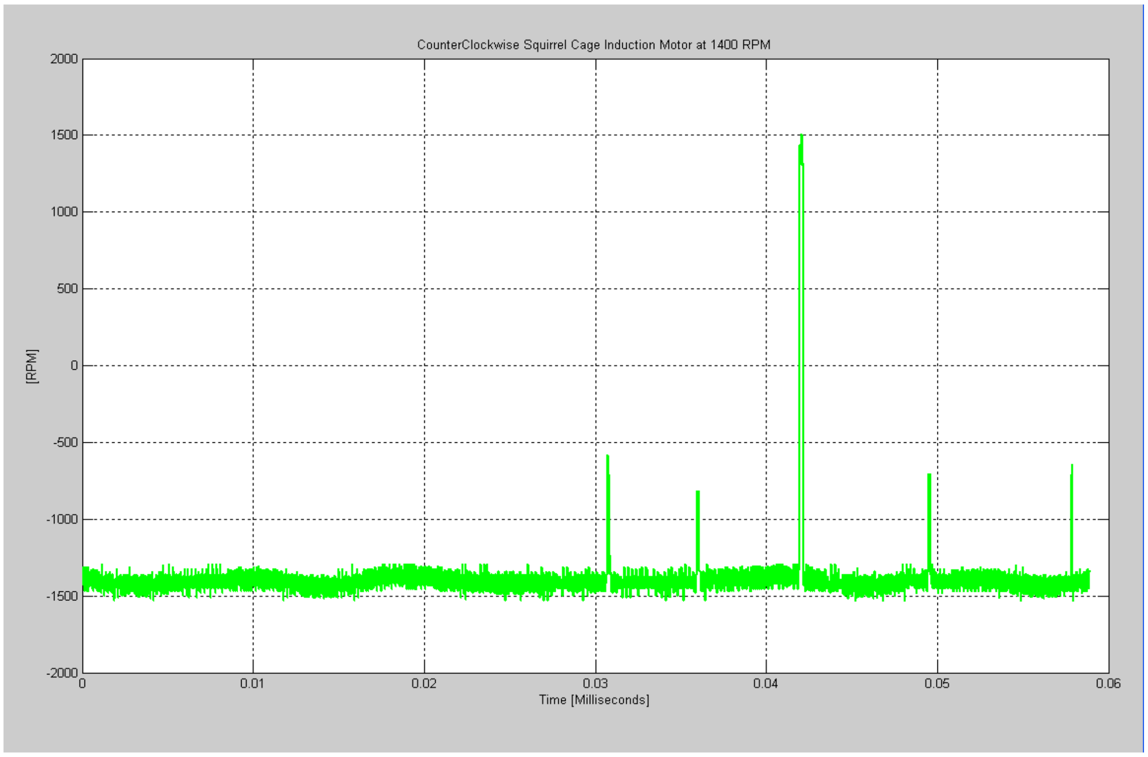


Figure 40. Counter-Clockwise Squirrel Cage Induction Motor at -1400 RPM.

D. CHAPTER SUMMARY

This chapter gave a brief overview of the Mathworks' Simulink® simulation software and highlighted its involvement in the computer algorithm development. Next, it revisited XILINX Foundation software application, and explained how this software generates VHDL code using ISE Foundation Software. It then went on to explain how ChipScope™ Pro Software was utilized in analysis of the ISE's data, as well as its interface with an FPGA. Lastly, this chapter was concluded by pointing out some of ChipScope Pro's limitations in regards to recording, analyzing, and displaying rotational data, as well as how these limitations were overcome. Chapter VI will discuss the ISE output results as well conclude this thesis.

VI. RESULTS AND CONCLUSION

A. ISE PHYSICAL PERFORMANCE RESULTS

In order to measure the quality of the algorithm developed for the ISE, a series of high and low speed operational tests were conducted in both the CW and CCW rotation directions in order to assess the overall validity of the encoder's algorithm as well as its implementation in regards to interface between hardware and software. A Squirrel Cage Induction Motor (SCIM) was utilized in the rotation of ISE after considering its excellent flexibility in terms of variable speed and direction control.

1. High Velocity Clockwise Rotation Results

The first test was analyzed using ChipScope Pro Software. The SCIM was calibrated to rotate at rate of 1400 RPM in the CW direction using the Biddle Hand Tachometer. After the speed of rotation was verified, the Bus Plot feature in ChipScope Pro produced the following results which are seen in Figure 41. The blue waveform represents the total degrees traveled had a positive slope, which indicated a CW rotation with an increasing value. Also, total degrees traveled reset to 0 upon reaching its 12 bit limitation of 4096 degrees traveled. Next, the green waveform representing the speed of rotation, clearly displayed a CW rotation rate of approximately 1400 RPM. Lastly, the red waveform representing degrees traveled per rotation also had a positive slope, thus verifying a CW rotation.

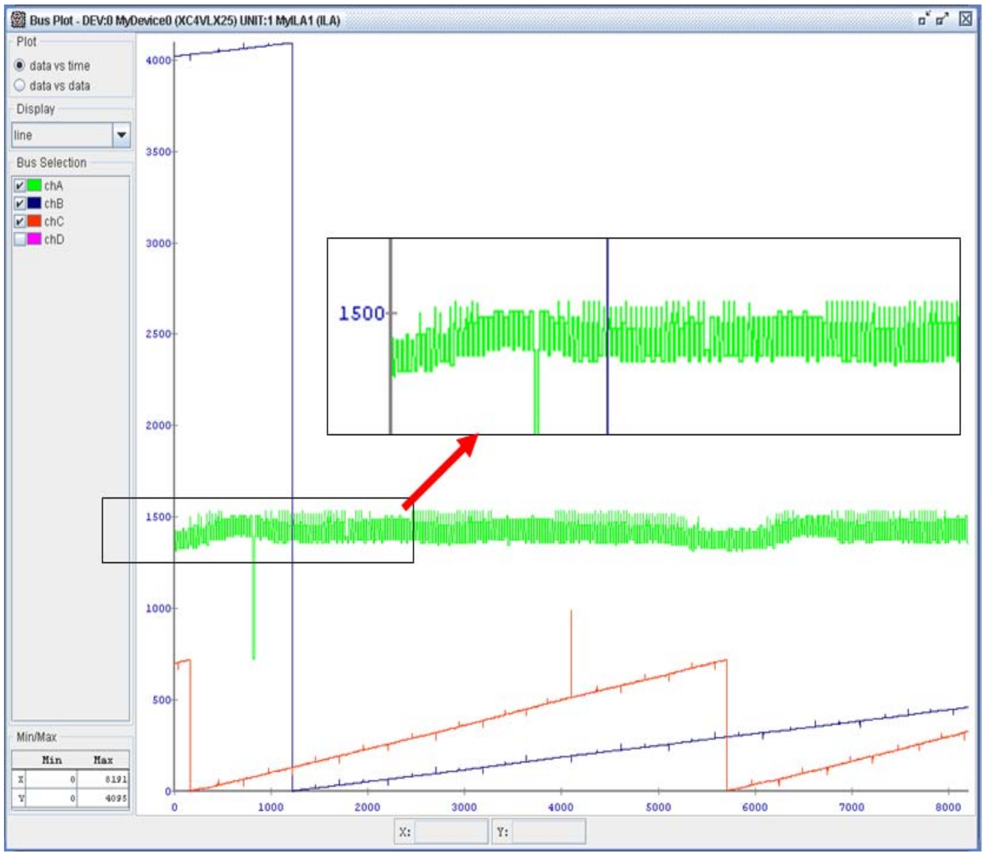


Figure 41. ChipScope™ Pro Bus Plot (CW Rotation at 1400RPM).

Next, these results where verified via a MATLAB plot. Figure 42 displays a CW rotation of the SCIM at approximately 1400 RPM. Figure 43 displays the degrees traveled per rotation. Note that the slope of the waveform is again positive and also reset to 0 when the value reached a maximum of 359.1 degrees traveled. Lastly, the time for 1 rotation was approximately 0.042 seconds. Therefore, using equation (2) the speed was then verified to be approximately 1428 RPM.

$$\left(\frac{1\text{ revolution}}{0.042\text{ seconds}}\right)\left(\frac{60\text{ seconds}}{1\text{ minute}}\right)=1428.6\text{ RPM} \tag{2}$$

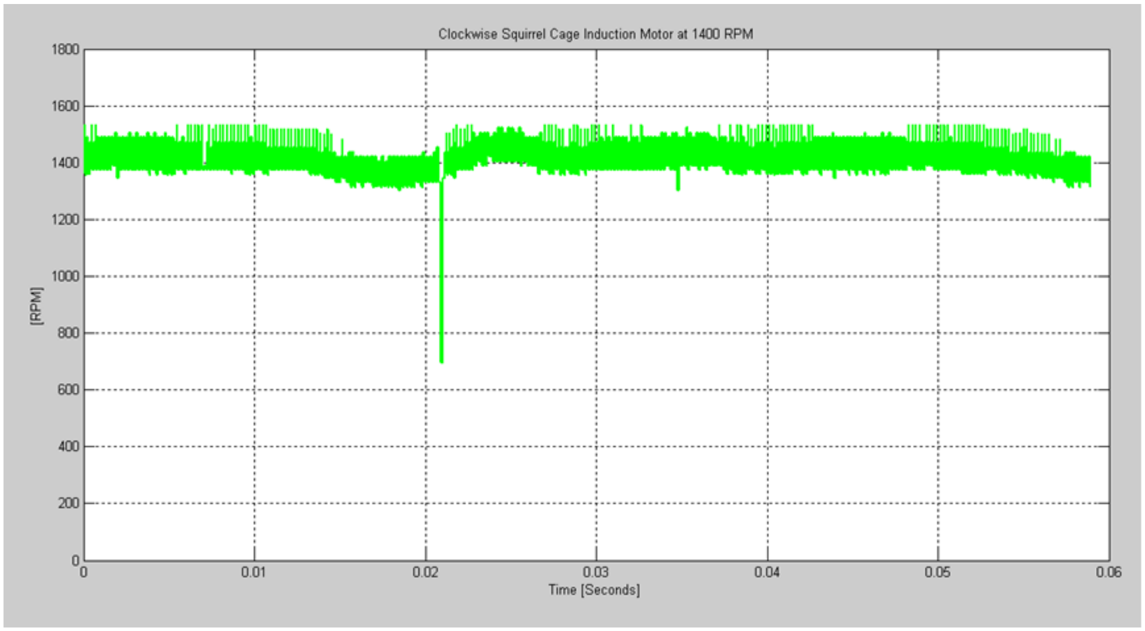


Figure 42. Squirrel Cage Induction Motor (CW Rotation at 1400RPM).

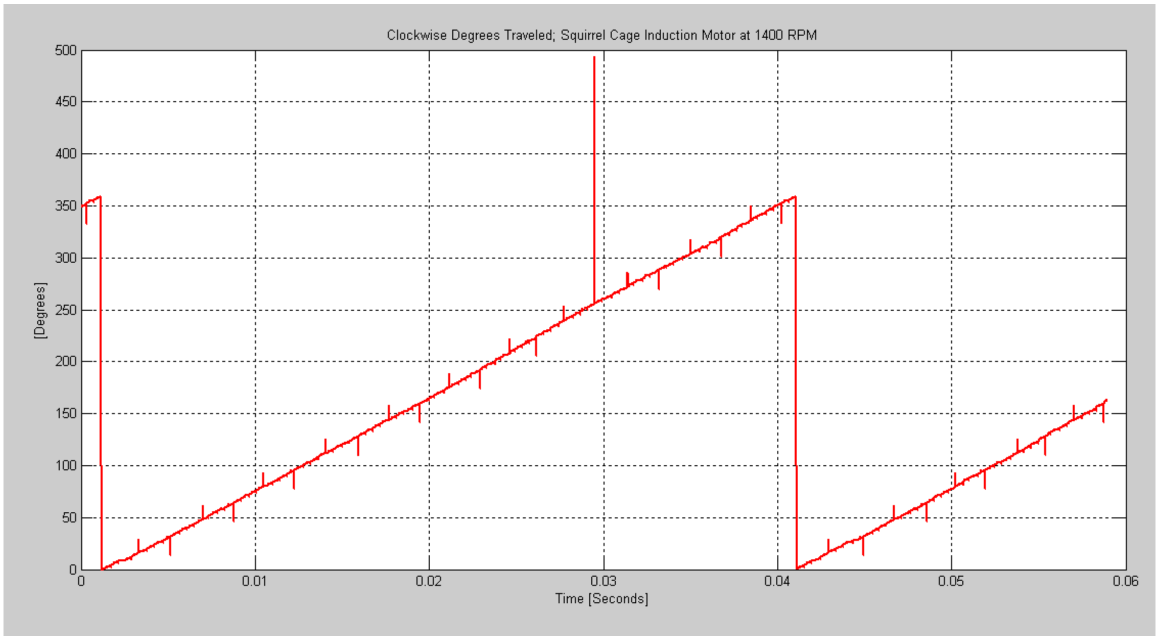


Figure 43. Degrees Traveled Squirrel Cage Induction Motor (CW Rotation at 1400RPM).

2. High Velocity Counter-Clockwise Rotation Results

In this test, a CCW rotation of the SCIM was setup again with a rotation rate of -1400RPM. The speed of rotation was verified via Biddle Hand Tachometer as well as all rotation data collected via ChipScope Pro software. As seen in Figure 44, this time both the total degrees traveled (blue waveform) and degrees traveled per rotation (red waveform) had a negative slope; this indicated a decreasing value in terms of total degrees traveled as well as degrees traveled per rotation. Moreover, similar to the CW rotation test, the total degrees traveled reset, but this time it reset at 0 to its maximum value of 4096. This reset was due to the rollover of its 12 bit binary word. Finally, the speed of rotation (green waveform), as expected, gave faulty data; therefore additional post-processing was utilized in order to display the actual rotation values achieved. Results of speed of rotation can be seen in Figure 45.

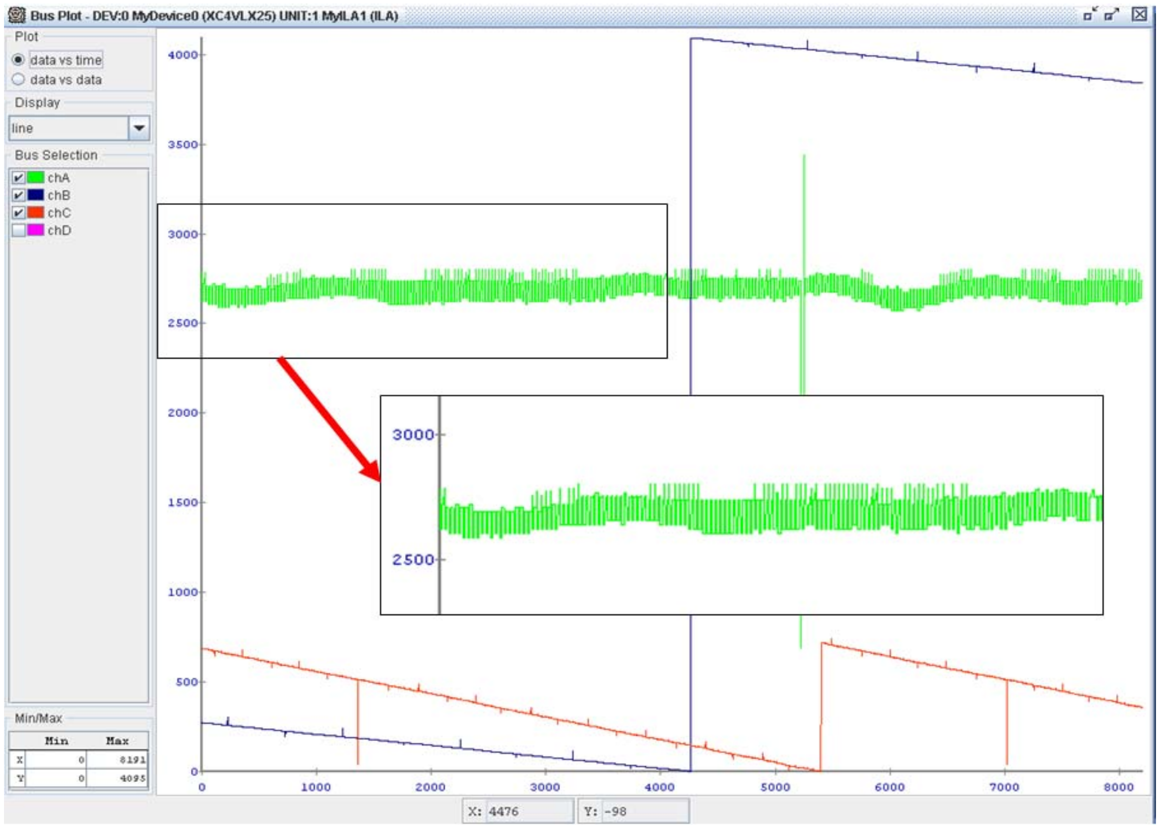


Figure 44. ChipScope™ Pro Bus Plot (CCW Rotation at -1400RPM).

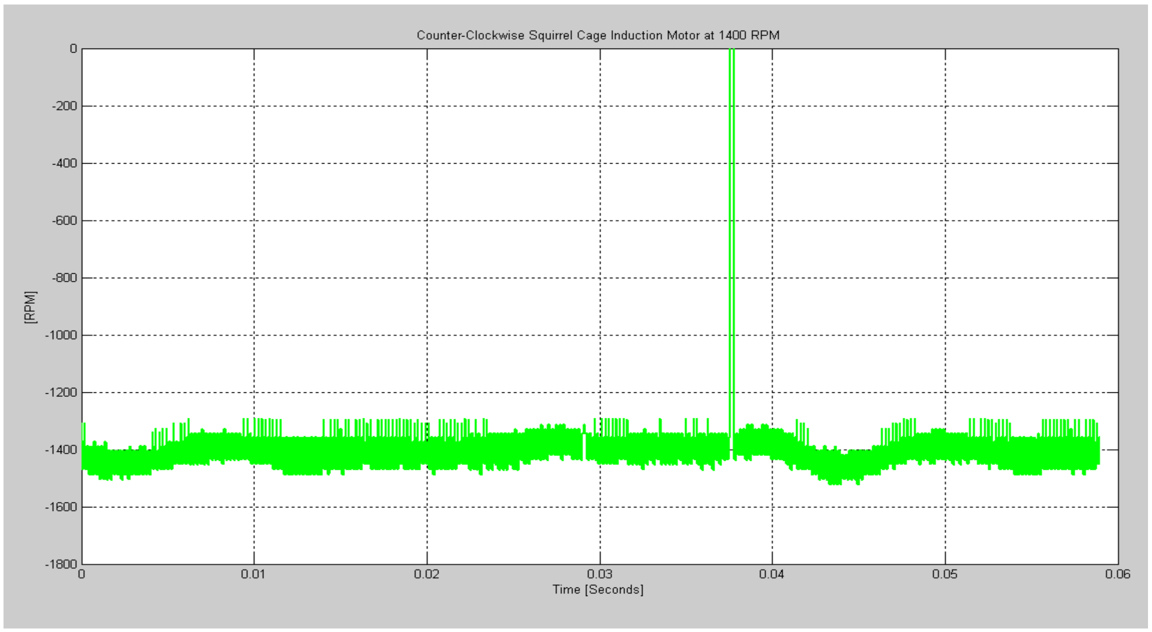


Figure 45. Squirrel Cage Induction Motor (CCW Rotation at -1400RPM).

Figure 45 displays the post-processed speed of rotation data. Note the rotation rate was approximately -1400RPM; the negative value indicated that a CCW rotation occurred. Furthermore, Figure 46 displays the degrees traveled per rotation in the CCW direction. Again, there is a reset to 359.1 degrees once the degrees archived a value of 0 degrees.

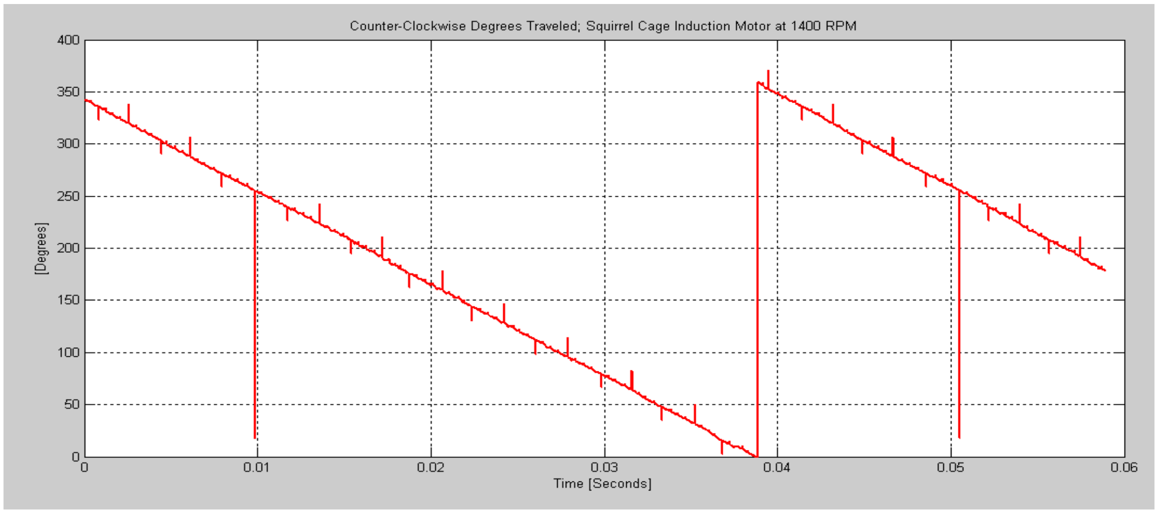


Figure 46. Degrees Traveled Squirrel Cage Induction Motor (CCW Rotation at -1400RPM).

As a point of interest, each waveform observed appears to have some noise associated with each of the output waveform displayed. Yet, this noise does not seem to affect the overall quality of the output signal generated. On the other hand, this noise should not be ignored; therefore a digital filter may help to minimize this output noise and should be considered for follow-on algorithm upgrades.

3. Low Velocity Clockwise Rotation Results

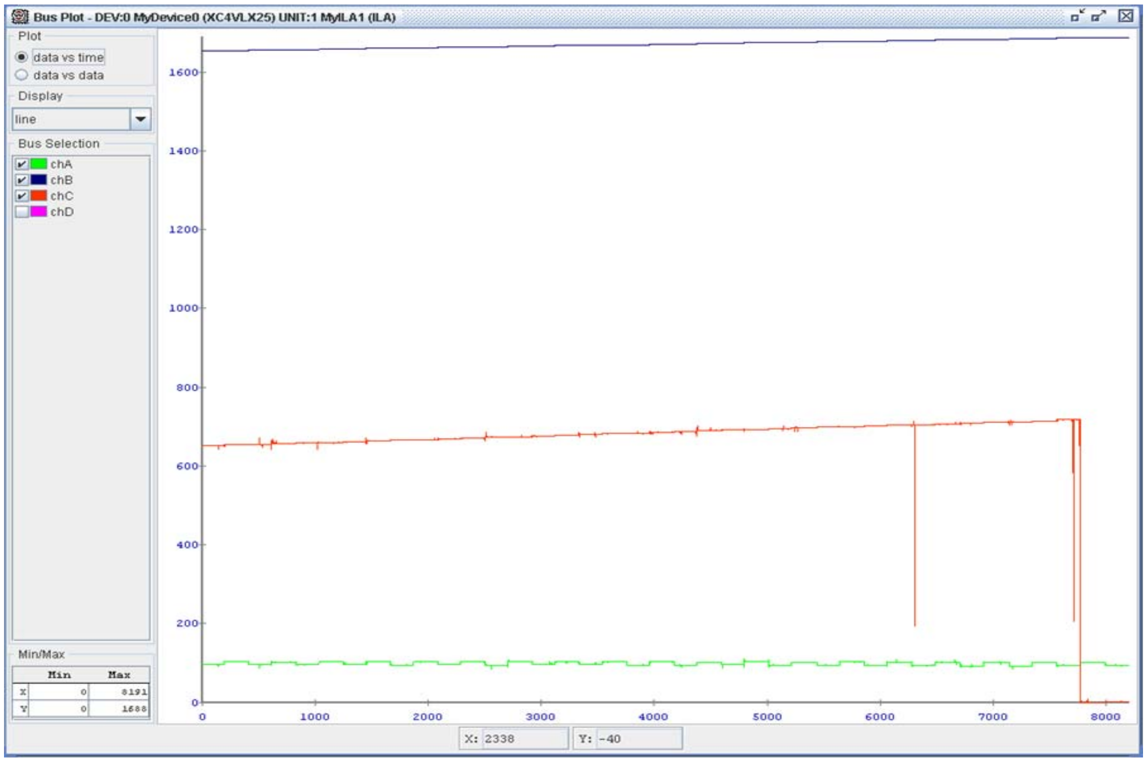


Figure 47. ChipScope™ Pro Bus Plot (CW Rotation at 100RPM).

Next, the encoder was run at a decreased speed of 100 RPM in the CW direction. ChipScope Pro software was again utilized to detect and record the results. As seen in Figure 47 above, the total degrees traveled (blue waveform) displayed a positive slope and thus increased in value. Additionally, the degrees traveled per rotation (red waveform) also increased in value and again verified that CW rotation occurred. Lastly, the speed of rotation (green waveform) correctly reflected 100RPM. Again, these results were post-processed using MATLAB for verification purposes and then displayed in Figure 48 and 49. As seen in Figure 48 and 49, similar results were obtained.

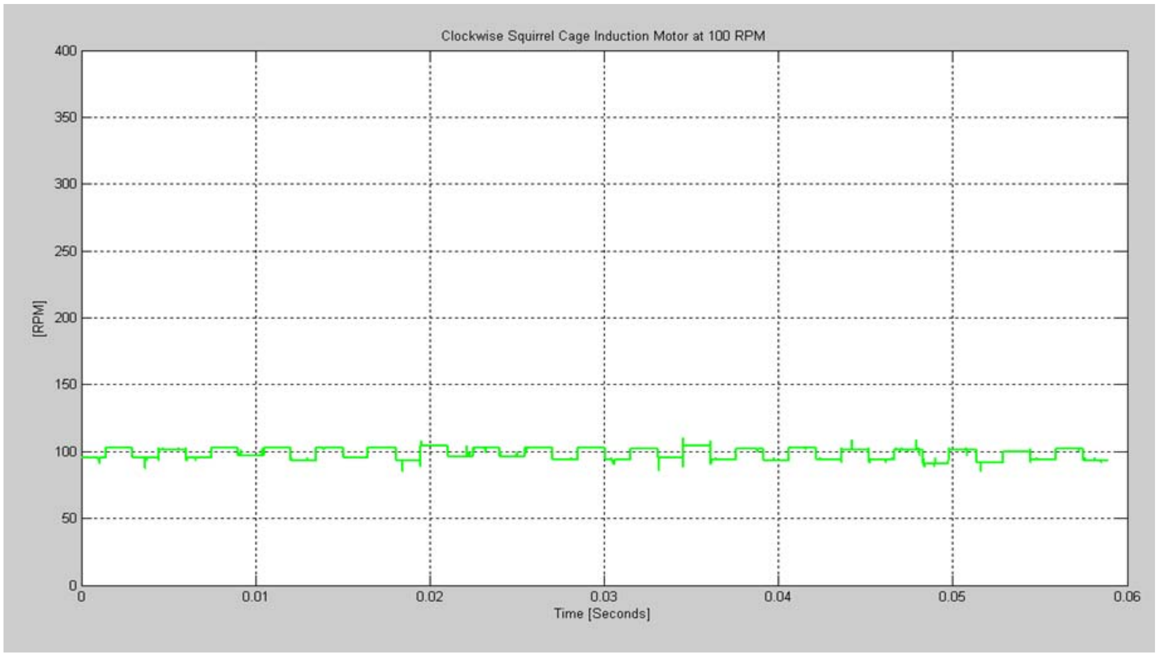


Figure 48. Squirrel Cage Induction Motor (CW Rotation at 100RPM).

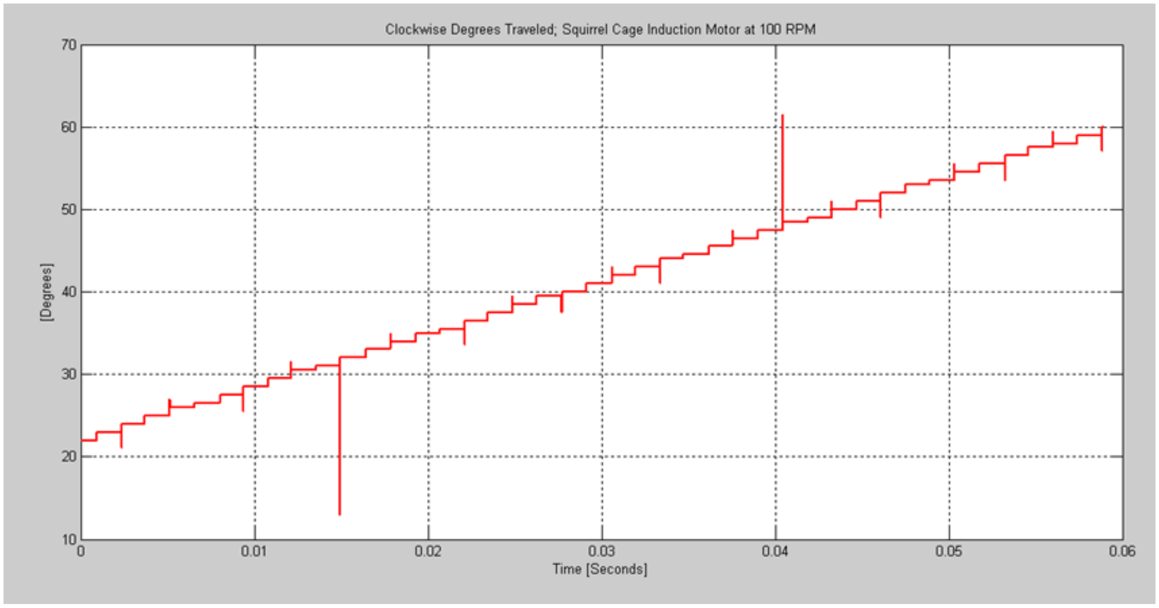
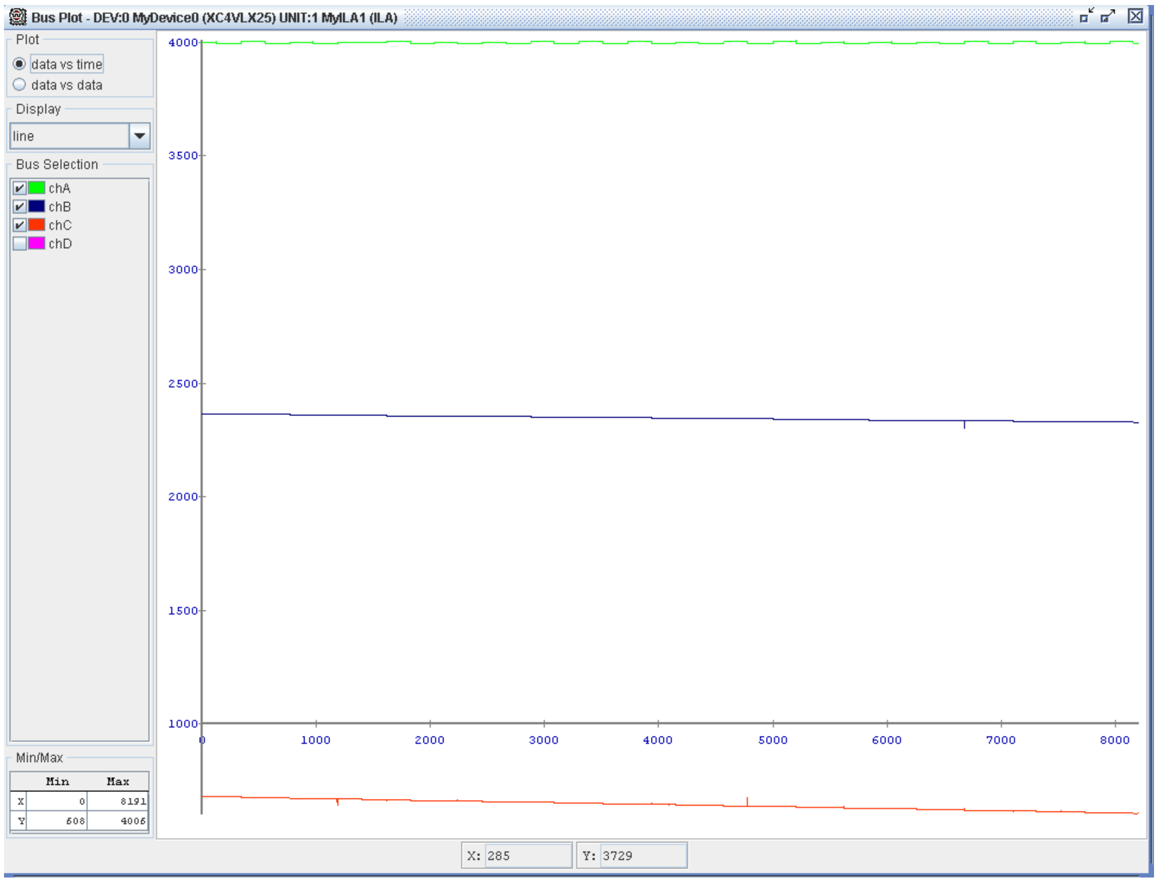


Figure 49. Degrees Traveled Squirrel Cage Induction Motor (CW Rotation at 100RPM).

4. Low Velocity Counter-Clockwise Rotation Results

Similar to low velocity CW rotation test, the encoder was run at the speed of -100 RPM, but this time in the CCW direction. Like all previous tests, ChipScope Pro software was utilized to detect and record the results. As seen in Figure 50, the total degrees traveled (blue waveform) as well as the degrees traveled per rotation (red waveform) both displayed a negative slope. Lastly, the speed of rotation (green waveform) required additional post-processing to correctly reflect actual results. These results can be seen in Figure 51 and 52.


Bus Plot - DEV:0 MyDevice0 (XC4VLX25) UNIT:1 MyILA1 (ILA)

Plot
[● data vs time]
○ data vs data

Display
line

Bus Selection
✓ 🟩 chA
✓ 🟦 chB
✓ 🟥 chC
☐ 🟪 chD

Min/Max

	Min	Max
X	0	8191
Y	608	4006

X: 285 Y: 3729

Figure 50. ChipScope™ Pro Bus Plot (CCW Rotation at -100RPM).

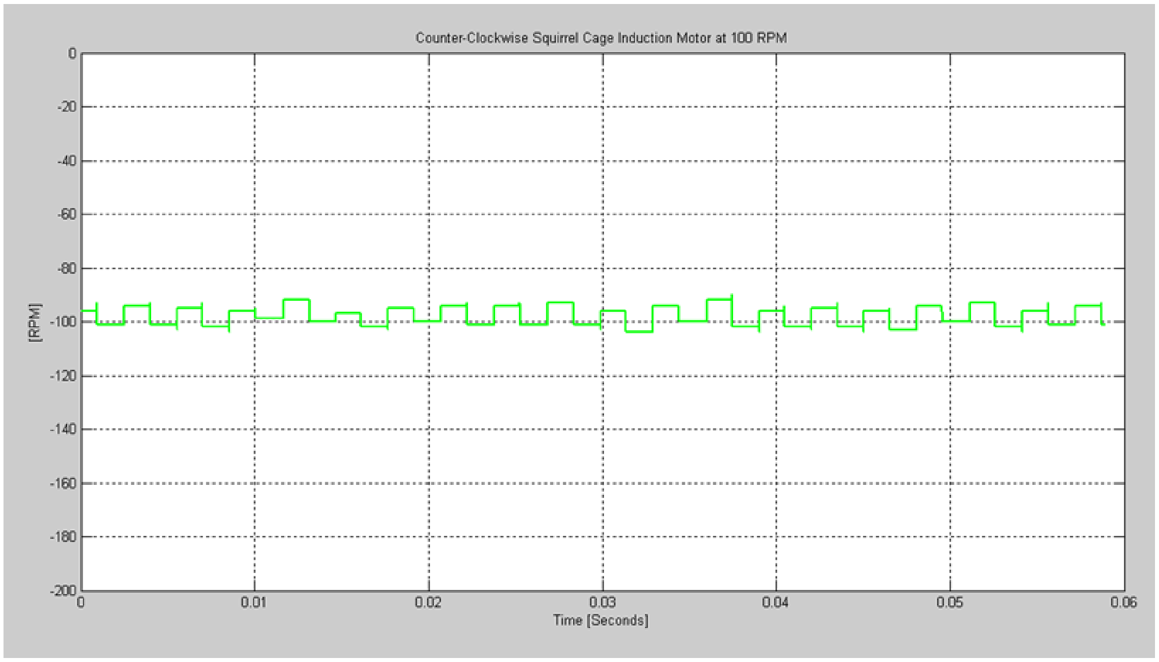


Figure 51. Squirrel Cage Induction Motor (CCW Rotation at -100RPM).

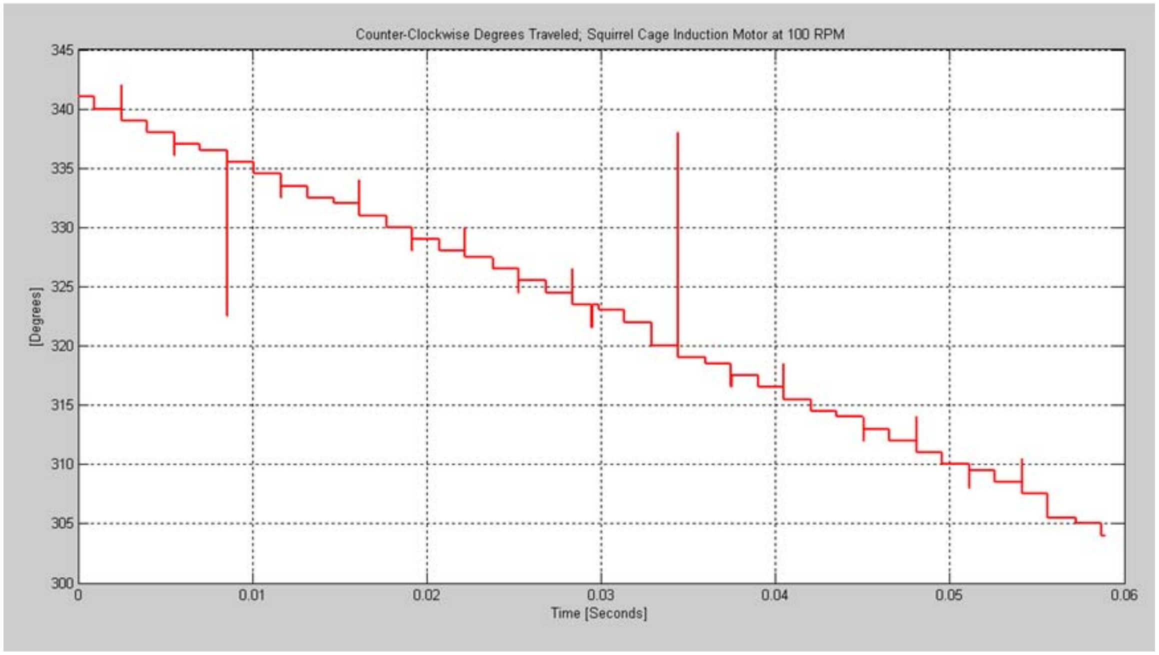


Figure 52. Degrees Traveled Squirrel Cage Induction Motor (CCW Rotation at -100RPM).

5. Algorithm Limitations and Overall Results

The threshold for accurate results in term of speed estimation appeared to be approximately 60 RPM. This value was obtained by running numerous speed tests at sequentially decreasing rate of speed. As it turns out, any speed of rotation that was below 60 RPM produced distorted and conflicting results. For example, in Figure 53, the speed of the device was set at approximately -60 RPM in the CCW direction. Again the Biddle Hand Tachometer was used to calibrate and verify the speed setting. The results however, do not reflect the actual speed of the device. Instead the values observed were recorded in a range of negative -250 to -400RPM. Similar outcomes were observed in the CW direction.

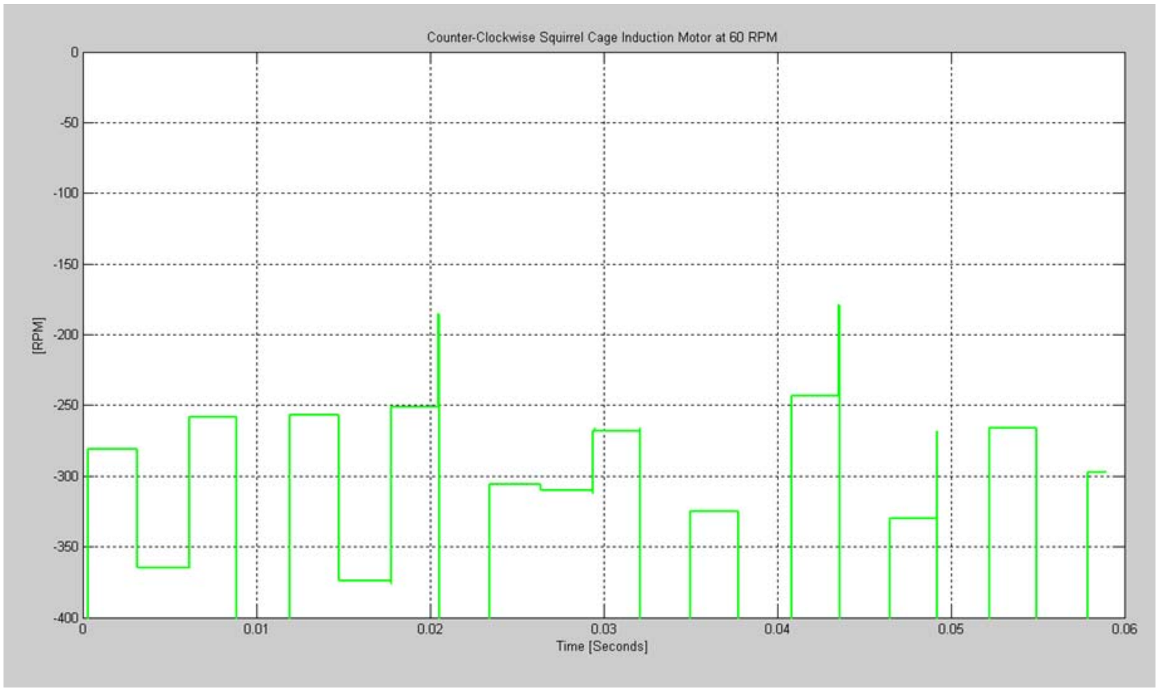


Figure 53. Squirrel Cage Induction Motor (CCW Rotation at -60RPM).

The above results place an operational limitation on the encoder's algorithm. On the other hand, the algorithm did function properly in regards to degrees traveled in the CCW and CW direction as seen in Figure 54.

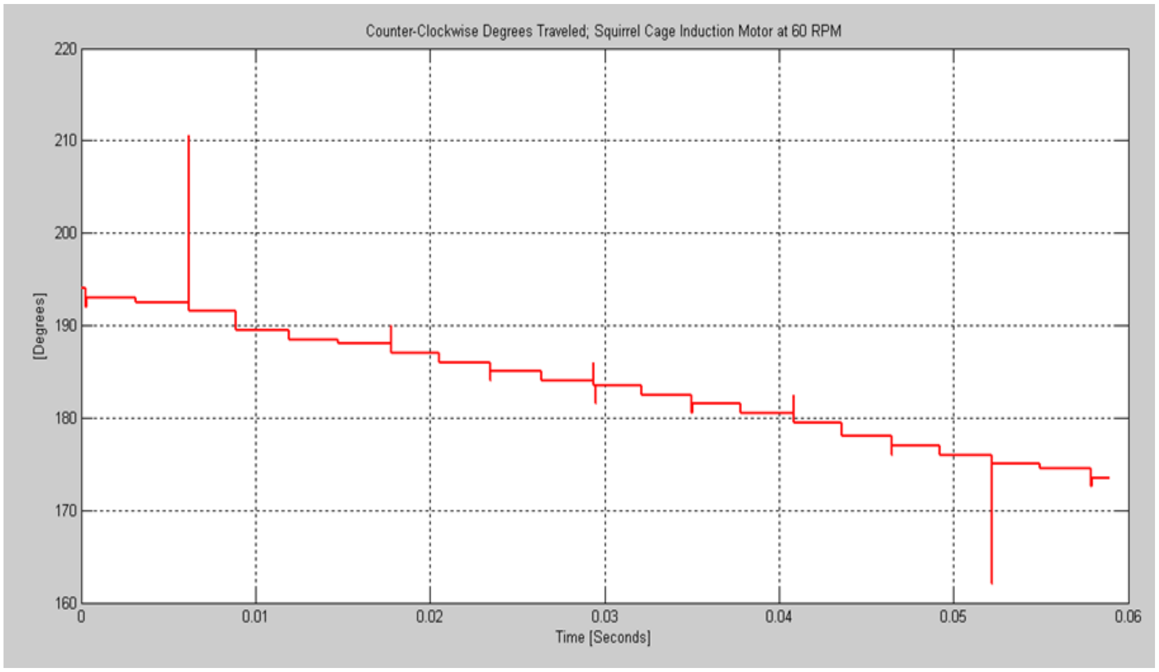


Figure 54. Degrees Traveled Squirrel Cage Induction Motor (CCW Rotation at -60RPM).

Note that the slope is decreasing; meaning the algorithm appears to be operating as designed in the CCW direction. However, to confirm this conclusion, equation (3) was utilized.

$$\left(\left(\frac{\text{1 revolution}}{\text{360 degrees}}\right)\left(\frac{\text{degrees per rotation}}{\text{sample period[seconds]}}\right)\left(\frac{\text{60 seconds}}{\text{1 minute}}\right)\right) = \text{RPM} \qquad (3)$$

If the algorithm was functioning correctly, one would expect a speed of rotation to be approximately -60RPM. Therefore, taking the total degrees traveled from time 0.01 to 0.05 seconds, which was approximately 15 degrees; then plugging this value into equation (3), it appears that the device was traveling at -62.5 RPM. As a direct result of these findings, the algorithm appears to be functioning correctly in regards to detecting and registering degrees traveled per rotation. The same equation was used in the high velocity test and similar results were recorded; which also confirmed algorithm validity at higher rotational speeds.

As an additional point to consider, the Biddle Hand Tachometer used to calibrate the speed of rotation also appeared to have great difficulty in measuring rotational speeds accurately at low velocities levels. With this in mind, as well as the results obtained from both the high and low velocity tests, it appears that the algorithm specifically designed for the ISE is functioning correctly and in accordance with the simulated computer model. This functionality includes the algorithms ability to detect and register both total degrees traveled and degrees traveled per rotation in both the CW and CCW directs. Lastly, the algorithm provides an accurate estimation of speed of rotation at speeds above 60 RPM.

B. SUMMARY

This thesis began with an overview of the SDC objective and description of the hardware and software used therein. The functionality and implementation of specific components of the SDC was highlighted in order to develop a comprehensive working knowledge of the SDC operational capabilities. Next, it provided a detailed analysis of the output signal generated by the ISE. It specifically then went on to explain how the encoder generated its three output waveforms, as well as underscored how these waveforms were utilized in rotational data collection. It then provided a detailed overview of the FPGA interface with the SDC, as well as the ISE. Furthermore, it described, in great detail, how the simulation that reproduced the ISE source signal, as well as the Rotor Speed Indicator simulated feature, was developed, implemented, and tested. Lastly, this thesis concluded by presenting the reader with operational tests results.

C. CONCLUSION

The SDC continues to be an excellent training tool in regards to digital control of power electronics design and testing. Through its use, the students gains a greater understanding of FPGA digital control of various power systems as well as gain a comprehensive working knowledge of analysis tools such as ChipScope™ Pro software in regards to signal analysis and processing.

Furthermore, with the addition of the ISE, the SDC enables the students to not only experiment with rotating magnetic field orientation problems, but it also provides the student with the hardware and software necessary to detect angular rotor position in both the CW and CCW rotation directions, total degrees traveled, as well as speed of rotation detection; which all could used as inputs for open and closed loop speed and torque control. This research also compared simulated operation of the encoder with measured results which taught the students that the physical operation of electronic equipment can be predicted via simulation prior to testing and eventually aid in the validation of the model used and developed. This greatly enhances the Department of Electrical and Computer Engineering's Power laboratory at the Naval Postgraduate School.

Moreover, as a point of interest, during development of the simulation, it became evident that the encoder's orientation relative to the shaft to which it was attached proved problematic. Therefore, the simulation was set up essentially in reverse of expected performance. Recall that CW rotation of a motor produced CCW results, and vice versa. Consequently, a small change in the state table was implemented that made the output appear to be opposite of actual rotation, hence the motor rotational data was reflected and not the encoder data. This change in the state table appeared to correct the encoder orientation problem and produced desired results.

Lastly, from the operational tests conducted, one can conclude that the algorithm was operating in accordance with original design specifications. Specifically, this algorithm functioned correctly for each and every test conducted; this included total degrees traveled and degrees traveled per rotation in both the CW and CCW direction. Speed of rotation was accurately predicted at all speeds over 60RPM. However, each test conducted also pointed out that there was at least some noise associated with each waveform produced. Although this noise did not appear to adversely affect the algorithms overall performance, the addition of a digital filter may help clear up this noise observed. All in all, a very functional algorithm was created and is now available for student use.

D. RECOMMENDATIONS FOR FURTHER STUDY

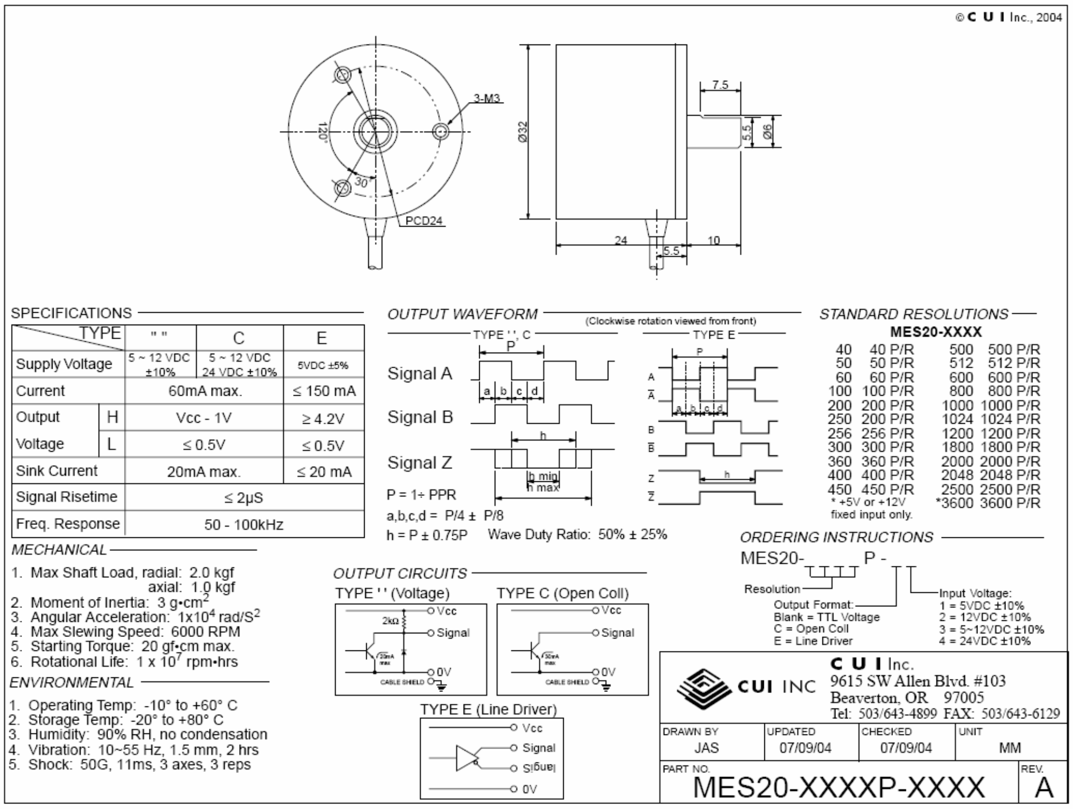
There are several opportunities for research in the area of total motor control in regards to power electronics. Below are several ideas that serve as platforms for further research:

- Development of a motor control laboratory for use in the electrical engineering curriculum tracks.
- Design and implement an algorithm to incorporate SDC current technology for closed and open loop speed control.
- Design and implement an algorithm to incorporate SDC current technology for closed and open loop torque control.

The reprogrammable nature of FPGA hardware enables great flexibility in term of design capabilities. With the addition of the MES20 (Type C) Shaft Encoder the student essentially has the tool necessary that will lead to motor control. Hence, electrical engineering design, especially at the graduate level, can benefit from the SDC use.

APPENDIX A: DATA SHEET AND MATLAB CODE

A. MES20 (TYPE C) INCREMENTAL SHAFT ENCODER DATA SHEET



3-M3
120°
30°
PCD24
Ø32
7.5
5.5
Ø6
24
10
5.5
OUTPUT WAVEFORM
(Clockwise rotation viewed from front)
TYPE ' ', C
TYPE E
P
Signal A
Signal B
Signal Z
a b c d
h
h min
h max
A
Ā
B
B̄
Z
Z̄
2kΩ
Vcc
Signal
0V
CABLE SHIELD
Signal

© C U I Inc., 2004

SPECIFICATIONS

TYPE		" "	C	E
Supply Voltage		5 ~ 12 VDC ±10%	5 ~ 12 VDC 24 VDC ±10%	5VDC ±5%
Current		60mA max.		≤ 150 mA
Output Voltage	H	Vcc - 1V		≥ 4.2V
	L	≤ 0.5V		≤ 0.5V
Sink Current		20mA max.		≤ 20 mA
Signal Risetime		≤ 2μS		
Freq. Response		50 - 100kHz		

P = 1÷ PPR
a,b,c,d = P/4 ± P/8
h = P ± 0.75P   Wave Duty Ratio: 50% ± 25%

MECHANICAL

1. Max Shaft Load, radial: 2.0 kgf
   axial: 1.0 kgf
2. Moment of Inertia: 3 g•cm^2
3. Angular Acceleration: 1x10^4 rad/S^2
4. Max Slewing Speed: 6000 RPM
5. Starting Torque: 20 gf•cm max.
6. Rotational Life: 1 x 10^7 rpm•hrs

ENVIRONMENTAL

1. Operating Temp: -10° to +60° C
2. Storage Temp: -20° to +80° C
3. Humidity: 90% RH, no condensation
4. Vibration: 10~55 Hz, 1.5 mm, 2 hrs
5. Shock: 50G, 11ms, 3 axes, 3 reps

OUTPUT CIRCUITS

TYPE ' ' (Voltage)   TYPE C (Open Coll)

TYPE E (Line Driver)

STANDARD RESOLUTIONS

MES20-XXXX

40	40 P/R	500	500 P/R
50	50 P/R	512	512 P/R
60	60 P/R	600	600 P/R
100	100 P/R	800	800 P/R
200	200 P/R	1000	1000 P/R
250	200 P/R	1024	1024 P/R
256	256 P/R	1200	1200 P/R
300	300 P/R	1800	1800 P/R
360	360 P/R	2000	2000 P/R
400	400 P/R	2048	2048 P/R
450	450 P/R	2500	2500 P/R
* +5V or +12V fixed input only.		*3600	3600 P/R

ORDERING INSTRUCTIONS

MES20-____P - _ _

Resolution

Output Format:
Blank = TTL Voltage
C = Open Coll
E = Line Driver

Input Voltage:
1 = 5VDC ±10%
2 = 12VDC ±10%
3 = 5~12VDC ±10%
4 = 24VDC ±10%

CUI INC
C U I Inc.
9615 SW Allen Blvd. #103
Beaverton, OR 97005
Tel: 503/643-4899 FAX: 503/643-6129

DRAWN BY	UPDATED	CHECKED	UNIT
JAS	07/09/04	07/09/04	MM

PART NO.	REV.
MES20-XXXXP-XXXX	A

B. MATLAB'S INITIAL CONDITIONS FILE

The Initial Conditions (IC) File defined several key conditions that are required for proper simulation operations. Specifically, the IC file defines the LUT "Reciprocal" output formula, as well as it defines the Modified 16 State Table via the output vector labeled "output_vec." Lastly it defines the vector used in the S/R Flip-Flop in regards to CW and CCW rotation detection.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File Name: Initial Conditions File for Incremental Shaft Encoder %
%   Author:    LT Andrew M. LaValley                                 %
%   Last Modified: 20 Aug 08                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t_square=2e-4;%2e-4 produces 1500RPM...4e-4 produces 750RPM
%8e-4 = 374RPM...10e-4=300RPM
%there is 400 1/2 pulses per rotation therefore:
%60sec/(400* 1/2*t_square)= correct RPM
RPM=60/(200*t_square);
Hz=RPM/60;
open_loop=0 ; %Set to one for open loop operation else set to zero for
%              closed loop voltage regulation
Kp_i=.06*2;   %current PI gain is amplified to account for the SV
%              modulation scaling
Ki_i=1*3;     %Current control loop gain
Kp_v=.2;
Ki_v=5;
f_clock=25e6;
sw_freq=15000;
sw_counter=round(f_clock/sw_freq-mod(f_clock/sw_freq,10));
%Counter for sawtooth for switching modulo 10 used so step_ct can be 10
Vdc=48;
Total_Rotations=6;
%step_ct=10;
step_ct=1;
tstep = step_ct/f_clock;
F_mat = [0 0 0 1;1 1 2 0;2 2 3 0;3 3 0 0];

Ctr=[1:2^11];
reciprocal=360/30/1./Ctr;%this defines the LUT "Reciprocal."

%this Plots the LUT Reciprocal for display%%%%%%%%%%%%%%%%%%%%%%%
figure(1);
plot(Ctr,reciprocal,'linewidth',2);
xlabel('Counter');
ylabel('1/Counter');
grid;
title('LUT Reciprocal')
MakeAxisWide
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
O_mat = F_mat;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                               %
%   This Defines the Modified 16 State Table [Pk, Nk, 0k]       %
%                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

output_vec=[0;...%Nothing is occuring with [0,0,0]
            5;...%CCW direction and ZCE [1,0,1]
            0;...%Nothing is occuring with [0,0,0]
            4;...%CCW direction with [1,0,0]
            0;...%Nothing is occurring with [0,0,0]
            3;...%CW rotation and ZCE [0,1,1]
            0;...%Nothing is occurring with [0,0,0]
            2;...%CW direction with [0,1,0]
            3;...%CW rotation and ZCE [0,1,1]
            0;...%Nothing is occurring with [0,0,0]
            2;...%CW direction with [0,1,0]
            0;...%Nothing is occurring with [0,0,0]
            5;...%CCW direction and ZCE [1,0,1]
            0;...%Nothing is occurring with [0,0,0]
            4;...%CCW direction with [0,1,0]
            0];%Nothing is occurring with [0,0,0]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                               %
%   This Defines the R/S FlipFlop [Pk, Nk, 0k]                  %
%                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        next = [0,0,1,0;1,0,1,0];
output=next;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%filter values and coefficients
A_D =1;
A_N =0.002898194633721;
B_D =-2.374094743709352;
B_N =0.008694583901164;
C_D =1.929355669091215;
C_N =0.008694583901164;
D_D =-0.532075368312092;
D_N =0.002898194633721;
```

C. M-CODE BLOCK MATLAB CODE

```
function [NewRotation, NewValue, Direction] = mcode5A(Pk, Nk, Zk,
OldRotation, OldValue)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File Name: mcode5A                                               %
%   Author:    LT Andrew M. LaValley                                 %
%   Last Modified: 14 APR 08                                         %
%                                                                    %
%   Description:    This function records CW and CCW rotations via    %
%                   matlab MCODE simulation block. It also records   %
%                   360 degrees of rotation and total degrees        %
%                   traveled.                                        %
%                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if Pk && ~Zk && ~Nk %Increments the # of pulses produced in the CW
rotation

        Direction=0;%indicates positive direction

    if OldValue==399 %this keeps the value positive if the value is
zero.

        NewValue=xfix({xlSigned,15,0},0);
        NewRotation=xfix({xlSigned,15,0},OldRotation+1);

    else
        NewValue=xfix({xlSigned,15,0},OldValue+1);
        NewRotation=xfix({xlSigned,15,0},OldRotation);

    end
elseif Nk && ~Zk && ~Pk %Decrements the # of pulses produced in the CCW
rotation if not zero

        Direction=1;%Indicates negitive direction

    if OldValue==0 %this keeps the value positive if the value is zero.

        NewValue=xfix({xlSigned,15,0},399);
        NewRotation=xfix({xlSigned,15,0},OldRotation-1);
        %Direction=1;
    else
        NewValue=xfix({xlSigned,15,0},OldValue-1);
        NewRotation=xfix({xlSigned,15,0},OldRotation);

    end


else %OldValue will equal NewValue and OldRotation will equal NewRotion
if
     %the above if/else statements do not apply.
        Direction=0;%default direction
        NewValue=xfix({xlSigned,15,0},OldValue);
        NewRotation=xfix({xlSigned,15,0},OldRotation);
```

end

D. MATLAB CODE USED IN POST-PROCESSING CHIPSCOPE DATA

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File Name: Post Processed Matlab File                              %
%   Author:    Dr. Alexander Julian                                    %
%   Last Modified: 23 Aug 08                                           %
%   Description:    This file post-processes all data coming from      %
%                   ChipScope. Essentially this file converts          %
%                   2s complement values in pos and neg output         %
%                   values.                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
datain=importdata('data.prn');
vecsize=length(datain.data);
alpha=2*pi*25000;
deltat=180/25e6;
datasize=round(vecsize-1);
adc1raw=zeros(1,datasize);
adc2raw=zeros(1,datasize);
adc3raw=zeros(1,datasize);
adc4raw=zeros(1,datasize);
for ii=1:datasize
    index=ii+vecsize-datasize-1;
    if datain.data(index,12+2)==0

adc1raw(ii)=datain.data(index,1+2)+2*datain.data(index,2+2)+2^2*datain.
data(index,3+2)+...
        2^3*datain.data(index,4+2)+2^4*datain.data(index,5+2)+...
        2^5*datain.data(index,6+2)+2^6*datain.data(index,7+2)+...
        2^7*datain.data(index,8+2)+2^8*datain.data(index,9+2)+...
        2^9*datain.data(index,10+2)+2^10*datain.data(index,11+2);
    else

adc1raw(ii)=datain.data(index,1+2)+2*datain.data(index,2+2)+2^2*datain.
data(index,3+2)+...
        2^3*datain.data(index,4+2)+2^4*datain.data(index,5+2)+...
        2^5*datain.data(index,6+2)+2^6*datain.data(index,7+2)+...
        2^7*datain.data(index,8+2)+2^8*datain.data(index,9+2)+...
        2^9*datain.data(index,10+2)+2^10*datain.data(index,11+2)-2^11;
    end

adc2raw(ii)=datain.data(index,1+14)+2*datain.data(index,2+14)+2^2*datai
n.data(index,3+14)+...
        2^3*datain.data(index,4+14)+2^4*datain.data(index,5+14)+...
        2^5*datain.data(index,6+14)+2^6*datain.data(index,7+14)+...
        2^7*datain.data(index,8+14)+2^8*datain.data(index,9+14)+...

2^9*datain.data(index,10+14)+2^10*datain.data(index,11+14)+2^11*datain.
data(index,12+14);
    if datain.data(index,12+26)==0

adc3raw(ii)=datain.data(index,1+26)+2*datain.data(index,2+26)+2^2*datai
n.data(index,3+26)+...
        2^3*datain.data(index,4+26)+2^4*datain.data(index,5+26)+...
        2^5*datain.data(index,6+26)+2^6*datain.data(index,7+26)+...
        2^7*datain.data(index,8+26)+2^8*datain.data(index,9+26)+...
        2^9*datain.data(index,10+26)+2^10*datain.data(index,11+26);
    else

adc3raw(ii)=datain.data(index,1+26)+2*datain.data(index,2+26)+2^2*datai
n.data(index,3+26)+...
        2^3*datain.data(index,4+26)+2^4*datain.data(index,5+26)+...
        2^5*datain.data(index,6+26)+2^6*datain.data(index,7+26)+...
        2^7*datain.data(index,8+26)+2^8*datain.data(index,9+26)+...
        2^9*datain.data(index,10+26)+2^10*datain.data(index,11+26);
    end
```

```
adc4raw(ii)=datain.data(index,1+38)+2*datain.data(index,2+38)+2^2*datai
n.data(index,3+38)+...
        2^3*datain.data(index,4+38)+2^4*datain.data(index,5+38)+...
        2^5*datain.data(index,6+38)+2^6*datain.data(index,7+38)+...
        2^7*datain.data(index,8+38)+2^8*datain.data(index,9+38)+...

2^9*datain.data(index,10+38)+2^10*datain.data(index,11+38)+2^11*datain.
data(index,12+38);
end



adc1=(adc1raw);
adc2=(adc2raw);
adc3=(adc3raw/2);
adc4=(adc4raw/2^3-100);

%Difference equation approximating a lowpass filter, alpha/(s+alpha)
adc2_fil=zeros(1,datasize);
adc4_fil=zeros(1,datasize);
for ii=1:datasize
if ii==1
    adc2_fil(ii)=0;
    adc4_fil(ii)=0;
else
    adc2_fil(ii)=adc2_fil(ii-1)+alpha*(adc2(ii-1)-adc2_fil(ii-
1))*deltat;
    adc4_fil(ii)=adc4_fil(ii-1)+alpha*(adc4(ii-1)-adc4_fil(ii-
1))*deltat;
end
end

time=[0:datasize-1]*deltat;

figure(1);
plot(time,adc1,'g','linewidth',2);
xlabel('Time [Seconds]')
ylabel('[RPM]')
title('Counter-Clockwise Squirrel Cage Induction Motor at 100 RPM')
% axis([0 0.06 -200 0])
MakeAxisWide
grid

figure(2);
plot(time,adc3,'r','linewidth',2);
xlabel('Time [Seconds]')
ylabel('[Degrees]')
title('Counter-Clockwise Degrees Traveled; Squirrel Cage Induction
Motor at 100 RPM')
%axis([0 0.06 0 400])
MakeAxisWide
grid
```

E. SUPPLEMENTAL MATLAB POST-PROCESSING PLOT FILES

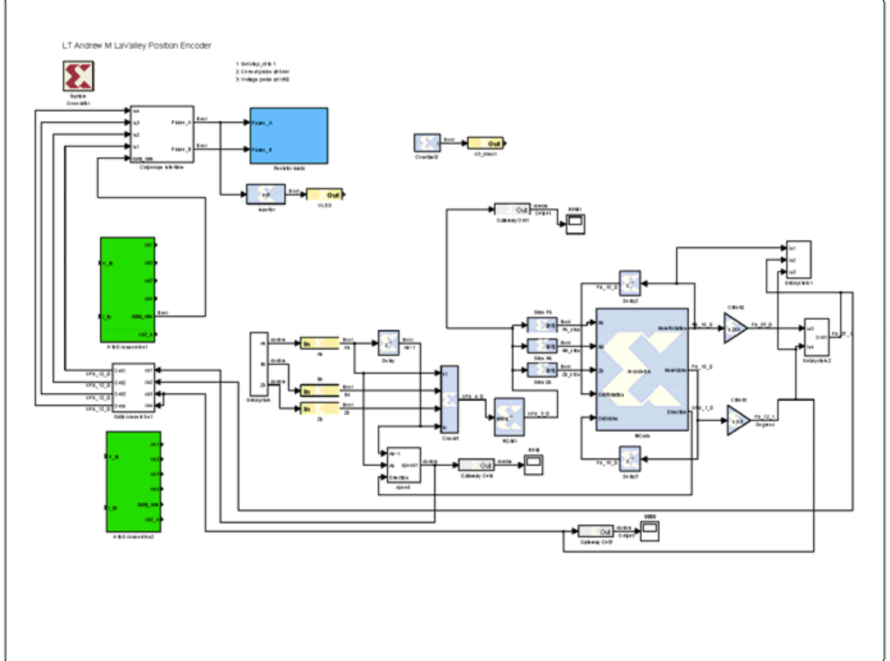
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   File Name: MakeAxisWide.m                                         %
%   Author:    LT Brian Decker                                        %
%   Last Modified: 19 APR 2006                                        %
%   Description:  takes your current axis and maximizes its width    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h_Temp = get(gca,'Position');
h_Temp(1)=0.07;
h_Temp(3)=0.9;
set(gca,'Position',h_Temp)
clear h_Temp
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: XILINX INCREMENTAL SHAFT ENCODER MODEL

Appendix *B* is a printout of the XILINK Foundation software model utilized in the algorithm and simulation development.

encoder2LFinal6

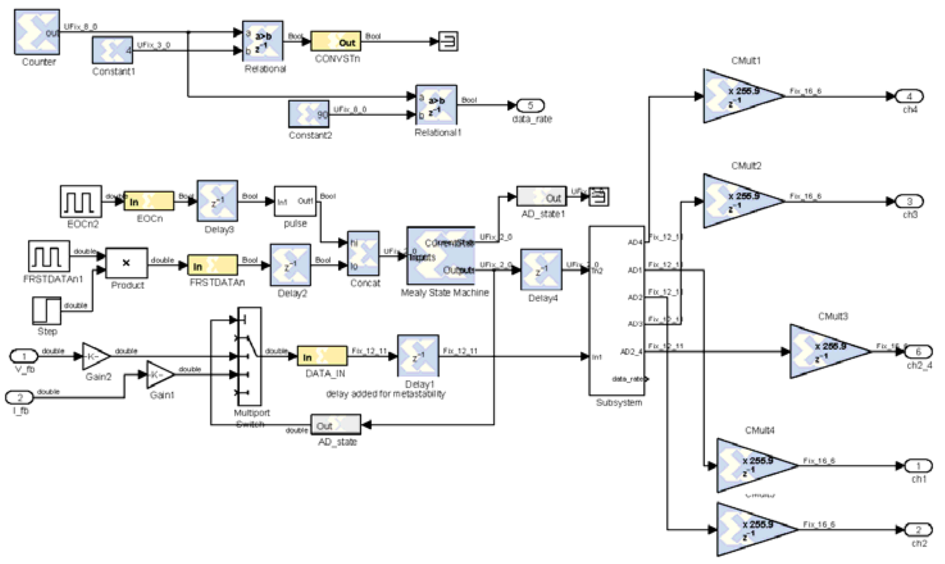


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 1/23

encoder2LFinal6/A toD conversion1

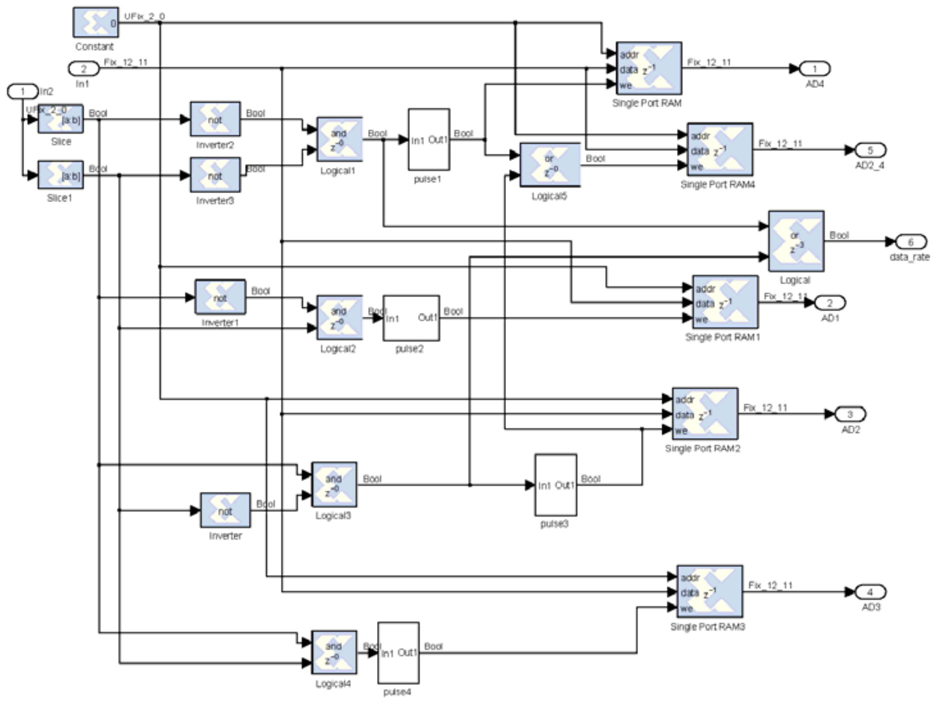


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04−Sep−2008 16:00

page 2/23

encoder2LFinal6/A toD conversion1/Subsystem

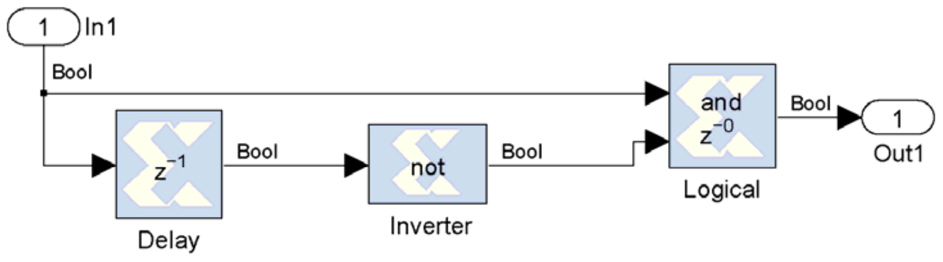


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 3/23

encoder2LFinal6/A toD conversion1/Subsystem/pulse1

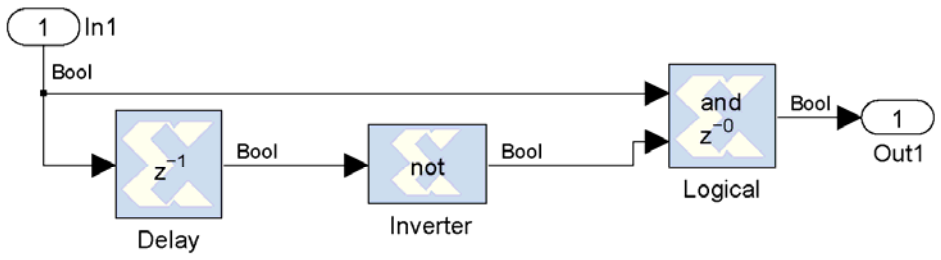


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 4/23

encoder2LFinal6/A toD conversion1/Subsystem/pulse2

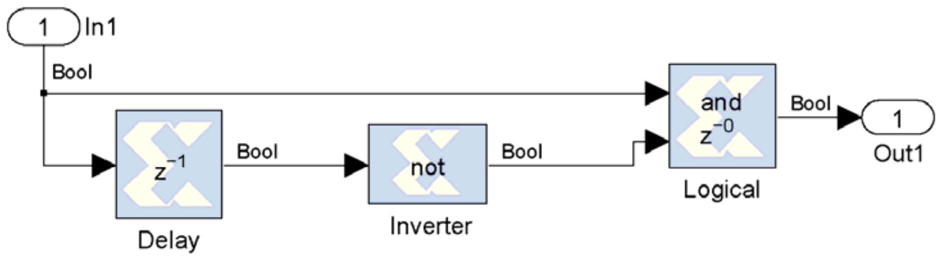


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 5/23

encoder2LFinal6/A toD conversion1/Subsystem/pulse3

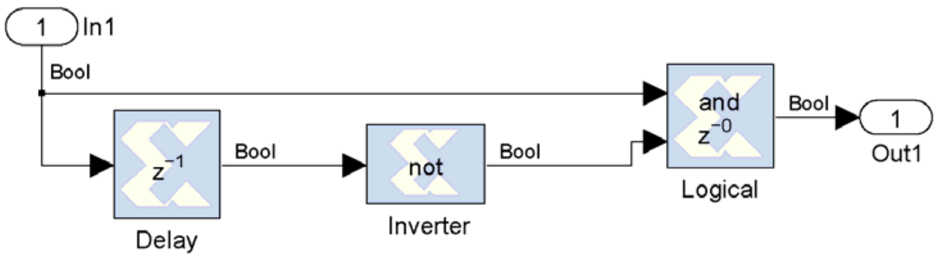


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 6/23

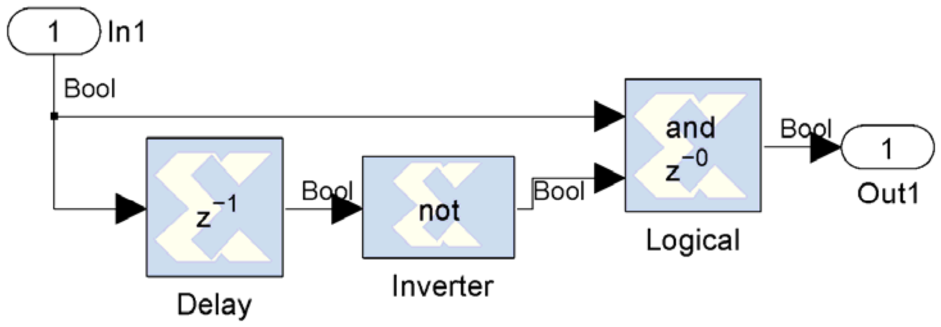
encoder2LFinal6/A toD conversion1/Subsystem/pulse4



H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00 | page 7/23

encoder2LFinal6/A toD conversion1/pulse

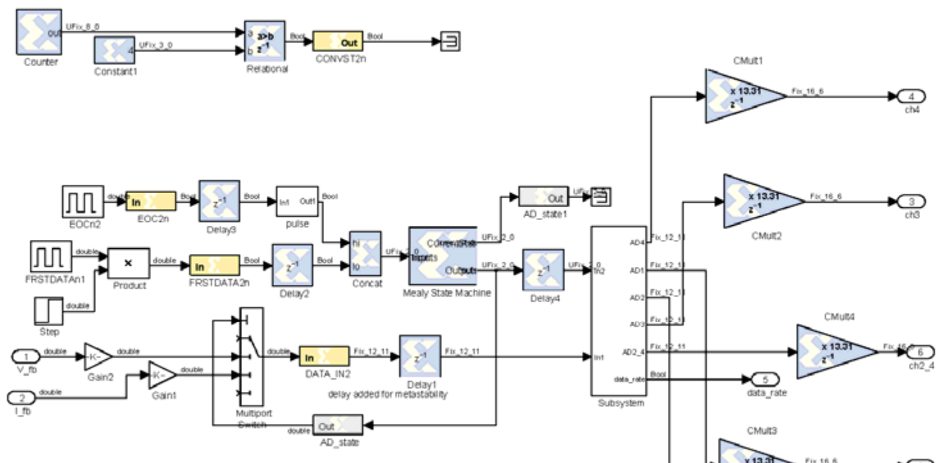


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 8/23

encoder2LFinal6/A toD conversion2

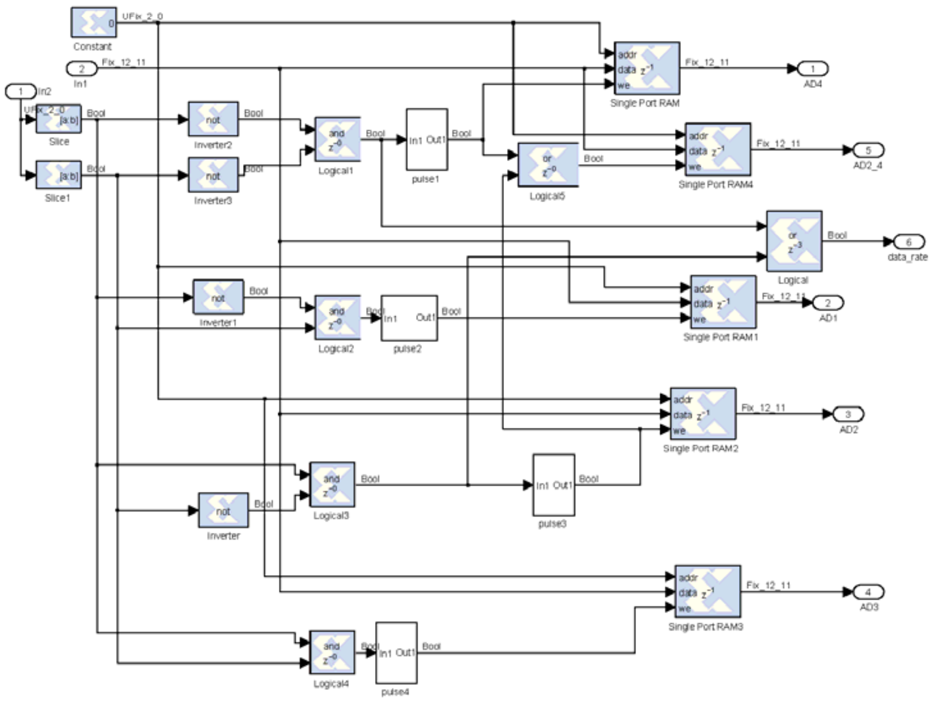


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04−Sep−2008 16:00

page 9/23

encoder2LFinal6/A toD conversion2/Subsystem

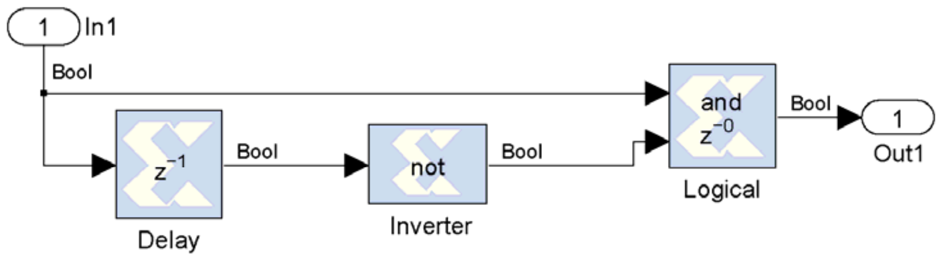


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 10/23

encoder2LFinal6/A toD conversion2/Subsystem/pulse1

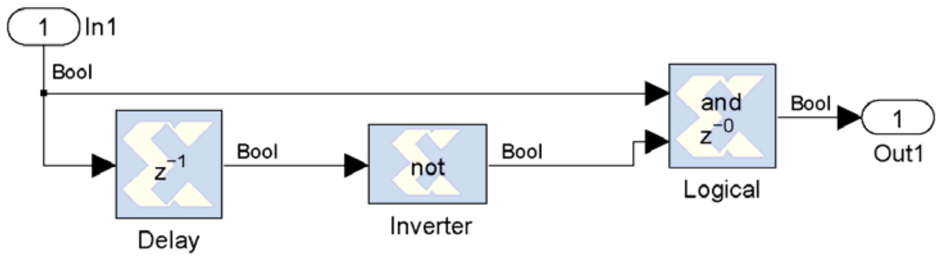


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 11/23

encoder2LFinal6/A toD conversion2/Subsystem/pulse2

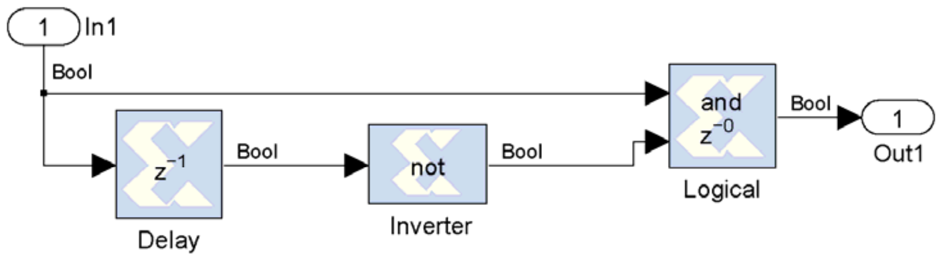


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 12/23

encoder2LFinal6/A toD conversion2/Subsystem/pulse3

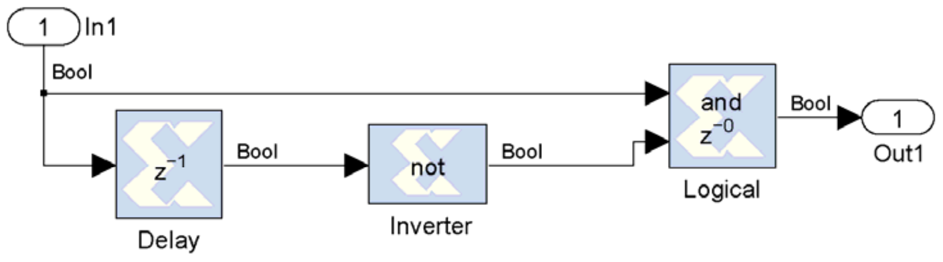


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 13/23

encoder2LFinal6/A toD conversion2/Subsystem/pulse4

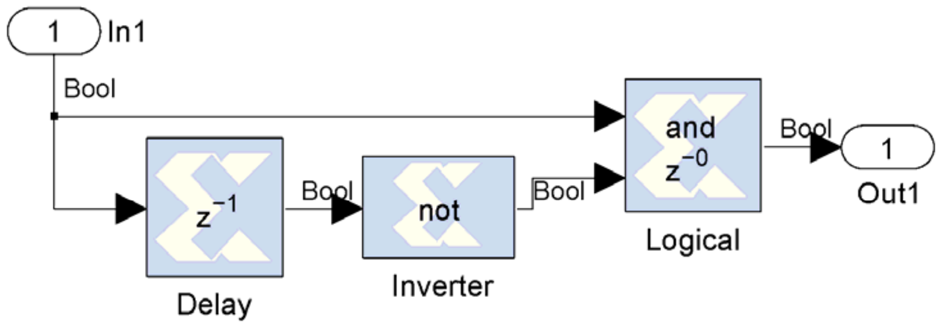


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 14/23

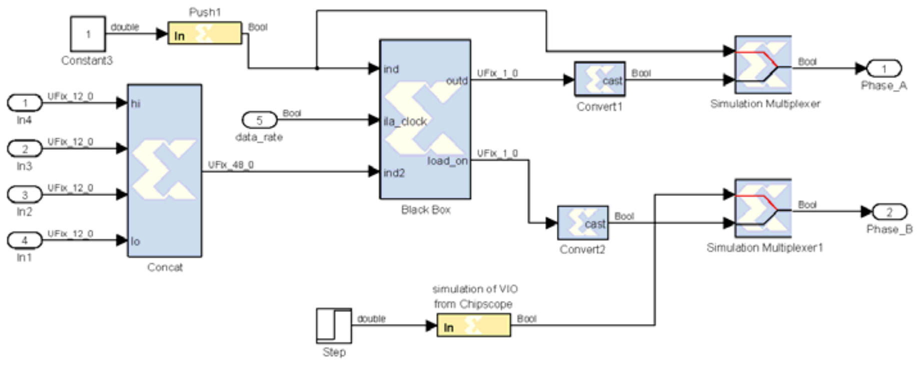
encoder2LFinal6/A toD conversion2/pulse



H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00 | page 15/23

encoder2LFinal6/Chipscope interface



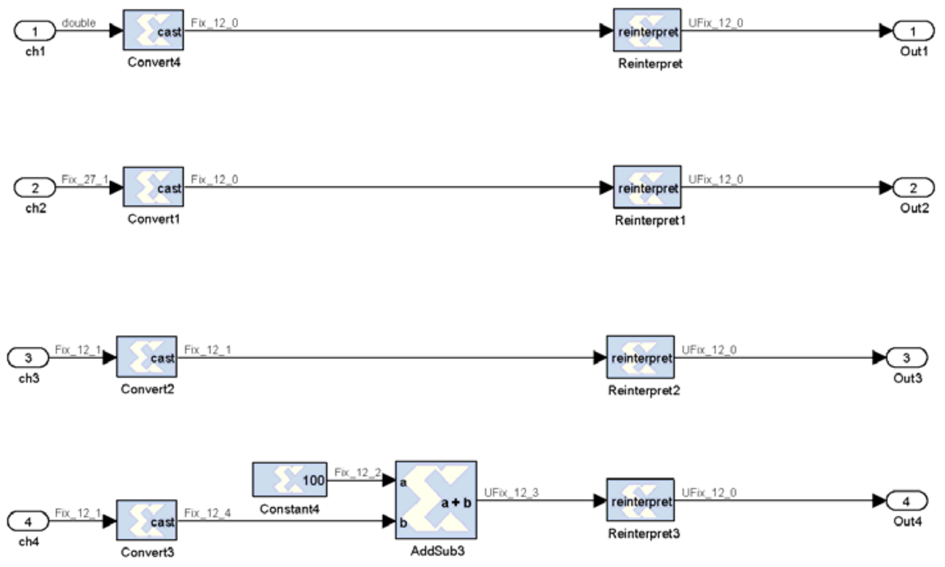
For simulation the black box must be bypassed.
Connect Out1 to COMM1
Connect load_on_off to a step function to simulate the load change

H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04–Sep–2008 16:00

page 16/23

encoder2LFinal6/Data conversion1

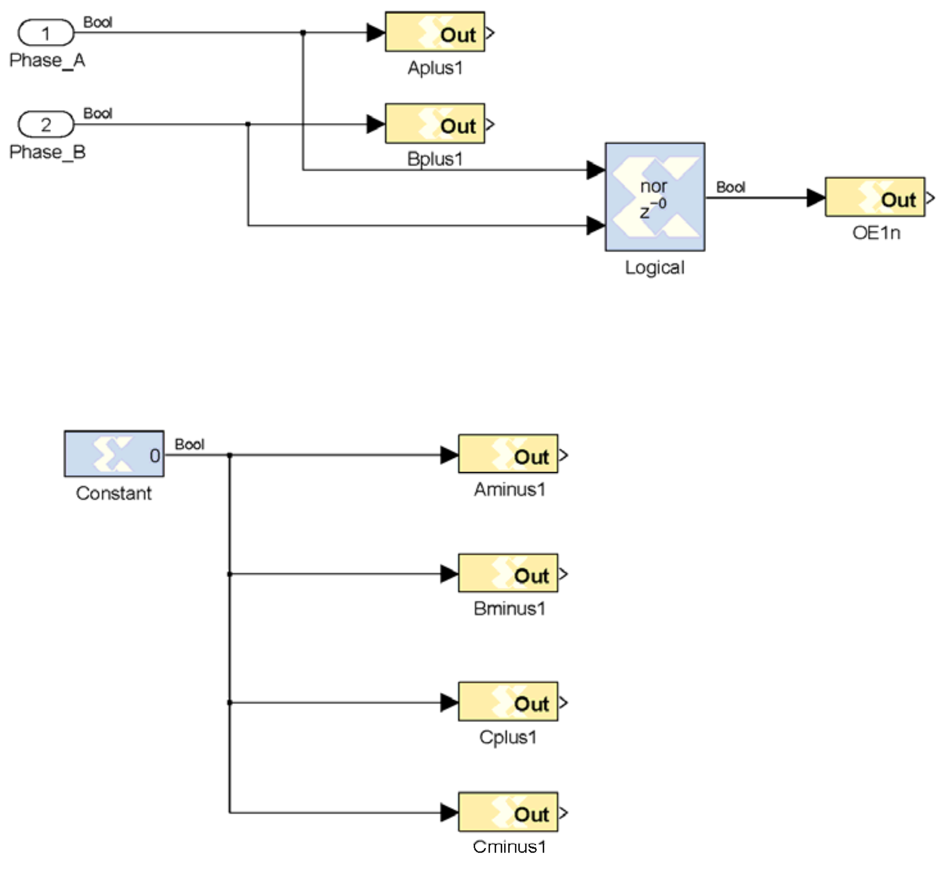


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 17/23

encoder2LFinal6/Resistor loads

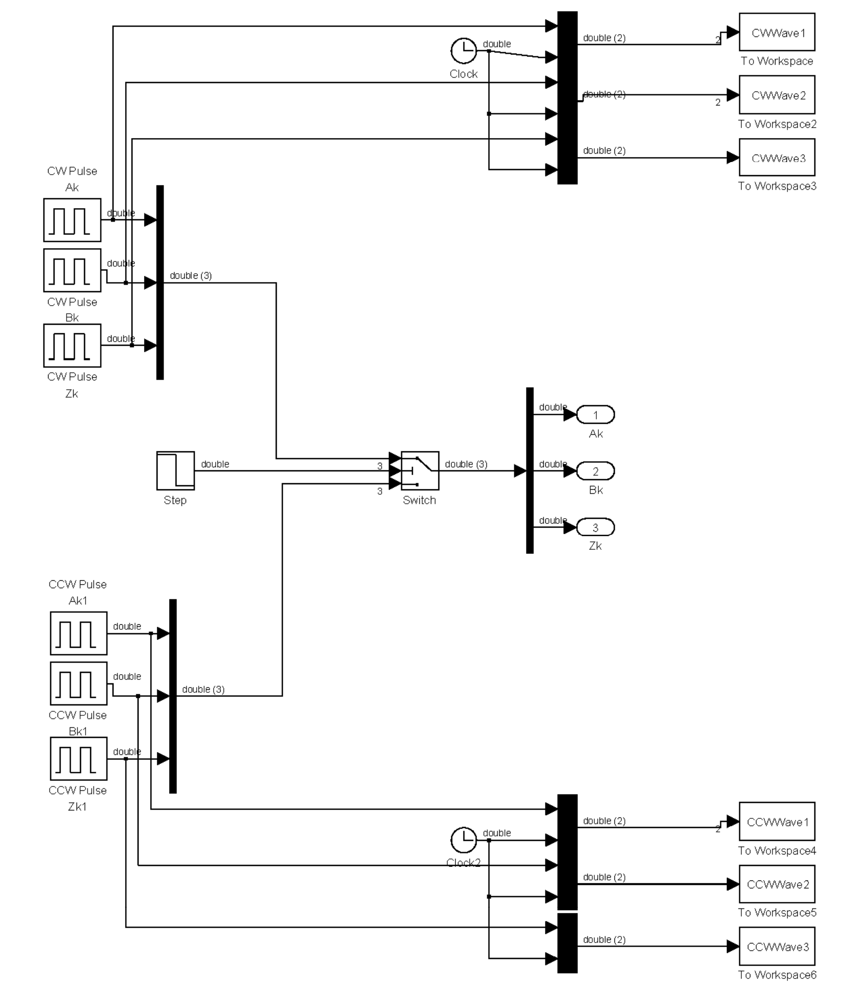


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 18/23

encoder2LFinal6/Subsystem

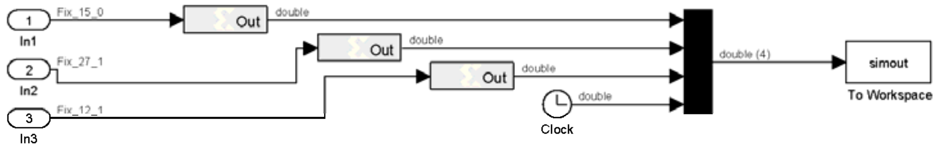


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 19/23

encoder2LFinal6/Subsystem1

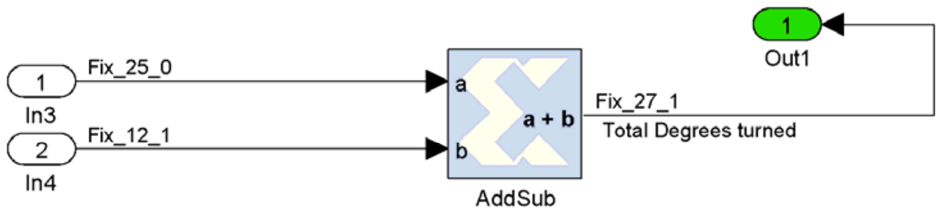


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04−Sep−2008 16:00

page 20/23

encoder2LFinal6/Subsystem2

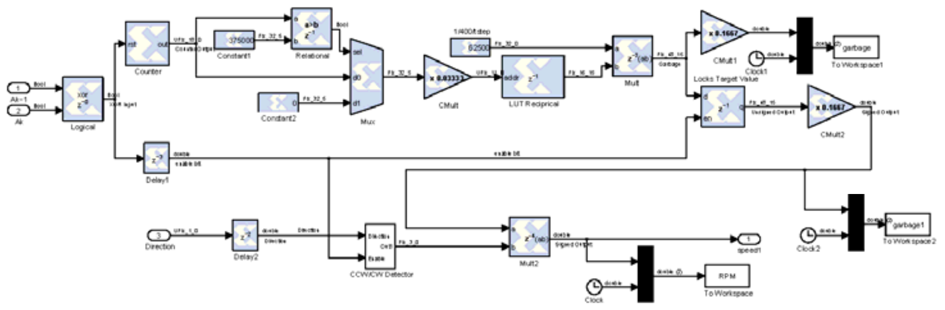


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 21/23

encoder2LFinal6/speed

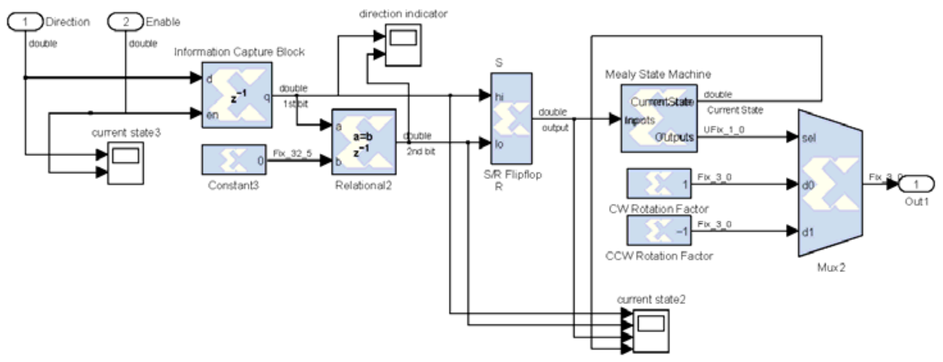


H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00

page 22/23

encoder2LFinal6/speed/CCW/CW Detector



H:\Matlab\xilinx\encoder2LFinal6.mdl

printed 04-Sep-2008 16:00 | page 23/23

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Joseph E. O’Connor, “Field programmable gate array control of power systems in graduate student laboratories, Master’s thesis, Naval Postgraduate School, Monterey, California, 2008.

[2] J. Schneider, M. Bezdek, Z. Zhang, Z. Zhang, and D. Rover, "A Platform FPGA-based hardware-software undergraduate laboratory," Presented at the IEEE International Conference on Microelectronic Systems Education, 2005.

[3] SearchCIO-Midmarket.com http://searchcioidmarket.techtarget.com/sDefinition/0,,sid183_gci530571,00.html, 03 August 2008.

[4] WIKIPEDIA®, Field-programmable gate array http://en.wikipedia.org/wiki/Field-programmable_gate_array, 03 August 2008

[5] MES20 (Type C) Incremental shaft encoder data sheet http://pdf1.alldatasheet.com/datasheet-pdf/view/238728/CUI/MES20.html, August 2008.

[6] Northwestern College Web Page http://mechatronics.mech.northwestern.edu/design_ref/sensors/encoders.html, August 2008.

[7] WIKIPEDIA®, Rotary Encoders, http://en.wikipedia.org/wiki/Shaft_encoder, 02 August 2008.

[8] WIKIPEDIA®, Tachometer http://en.wikipedia.org/wiki/Tachometer, 05 August 2008.

[9] Mathworks’ Product Marketing Information, “Simulink® simulation and model-based design,” http://www.mathworks.com/products/simulink/?BB=1, 11 August 2008.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Jeffery Knorr
   Electrical Engineering and Computer Department
   Code EC/Ko
   Naval Postgraduate School
   Monterey, California

4. Dr. Alexander Julian
   Electrical Engineering and Computer Department
   Code EC/J1
   Naval Postgraduate School
   Monterey, California

5. Dr. Roberto Cristi
   Electrical Engineering and Computer Department
   Code EC/C1
   Naval Postgraduate School
   Monterey, California