2007

Implementation of unmanned vehicle control on FPGA based platform using system generator

Shashikala Narasimha Murthy
University of South Florida

Implementation of Unmanned Vehicle Control on FPGA Based Platform Using System Generator

by

Shashikala Narasimha Murthy

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering
Department of Electrical engineering
College of Engineering
University of South Florida

Date of Approval:
October 26, 2007

Keywords: FPGA, SysGen, Matlab, PID, VHDL, Unmanned Systems

DEDICATION

I dedicate this thesis to my parents.

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my mentor and guide, Dr Wilfrido Moreno, Associate Professor in the Department of Electrical Engineering, College of Engineering, University of South Florida. His encouragement, motivation and personal guidance have provided a basis for the work represented in this thesis.

In addition, I would like thank my committee members, Dr. Kimon Valavanis and Dr. Paris Wiley for their assistance and expertise throughout the duration of this project.

My special thanks to Wendy Alvis, who did an enormous amount of work for the Autopilot project. Her experiments and suggestions for further development as well as her assistance during writing the thesis were invaluable to me.

I also would like to thank the Ibero-American Science and Technology Education Consortium, ISTEC and Xilinx for supporting my research with the required hardware and software tools.

Last, but not least, thanks to my husband, Praveen, for his love and support. His kindness and compassion have helped to keep my sanity throughout the many challenges faced over the past few years.

TABLE OF CONTENTS

LIST OF FIGURES

Implementation of Unmanned Vehicle Control on FPGA Based Platform Using System Generator

Shashikala N. Murthy

ABSTRACT

The goal of this research was to explore a new and improved software development tool for the implementation of control algorithms on Xilinx Field Programmable Gate Arrays (FPGA). The *Simulink* plug in, System Generator, complements traditional Hardware Description Language (HDL) by providing a higher level graphical language for the development of FPGA designs. The design is then translated into the lower level required by the Xilinx's ISE program. By utilizing this graphical based higher level of abstraction at the design entry level, the requirement of a detailed knowledge of HDL languages is no longer required. Because of this new environment the time required to implement the previously developed control design on the FPGA is reduced. The initial work began with a study of System Generator capabilities. One of the primary areas of interest is the difference on how the mathematical model representations are implemented between *Simulink* and the logic based hardware. From this initial work, a methodology for conversion between the developed and verified *Simulink* design and hardware implementation was obtained. As a case study, a control design was implemented for a *Simulink* model of an Unmanned Ground Vehicle (UGV) based on an RC-Truck. The control system consists of a simple mission planner to generate a vector of waypoints, a proportional-integral velocity controller and a proportional heading controller. The derived hardware design process is then utilized and validated by converting the control system into the available System Generator blocks. The final verification of the FPGA design was a hardware-in-the-loop simulation utilizing a Xilinx prototyping board. This design example demonstrated the validity of the presented approach as an efficient and reliable method for rapid system prototyping for designs developed within the *Simulink* environment.

CHAPTER ONE

INTRODUCTION

Typically efficient implementation of control system applications utilizing FPGAs requires a thorough understanding of both the hardware platform and Hardware Description Language. FPGAs have always had the advantages of parallel processing and asynchronous timing capabilities. Over the past decade both the increased number of gates and the development of new software tools have lead to a rapid increase in popularity of FPGAs.

New software tools have been developed to allow for higher level abstraction in the development of the FPGA implementations. Within the past few years, Xilinx has presented and made continued improvements to a *Simulink* add on, System Generator, that allows the design of the hardware from within the graphical, high level *Simulink* environment. System Generator replaces the traditional Hardware Description Language (HDL) design, and thus does not require a detailed knowledge of this lower level, complex language. In addition, the graphical language allows an abstraction of the design through the use of available System Generator blocks and subsystems. This reduces the time necessary between the derivation of control design and hardware implementation. This software is extremely attractive because of the popularity of *Simulink* as tool for both modeling the physical system and testing the derived control design. It is a natural progression to include the hardware simulation and hardware-in-the-loop verification from within this environment. However, the behavior of the *Simulink* mathematical simulation in comparison to the hardware implementation is not an exact match. *Simulink* allows for floating point, complicated math to be completed within a single, virtual time step by slowing the simulation to allow for precise calculations. In addition, *Simulink* will adjust the size of the time step when required by the underlying calculations. By comparison, the FPGA hardware implementation requires a pre-defined fixed time step that operates in real time. While rates can be adjusted at different points within the hardware to allow for asynchronous timing, each of these rates are run at consistent time step with a fixed word length. The conversion between these two forms must be done in a systematic way that takes these differences into consideration. Thus the derivation of the hardware implementation can become a difficult and frustrating task for those unfamiliar with the FPGA environment.

The goal of this research was twofold, to explore this new and improved software development tool and to develop a systematic approach of conversion from the verified *Simulink* design to the hardware implementation. This objective can be broken further into three smaller objectives – design, implementation and verification. The design flow allows the developers to quickly explore FPGA design options and to check if the resulting module fulfills the design constraints. The successful development of such a systematic approach allows the controls engineer to follow this procedure in order to obtain a successful hardware design without extensive knowledge in logic theory. In addition, this work compliments an FPGA based autopilot hardware design for use with unmanned systems by simplifying the implementation of the controls algorithm onto an available, off-the-shelf platform.

A case study utilizing a pre-developed *Simulink* system model of an Unmanned Ground Vehicle (UGV) based on an RC-Truck model was completed. A Simple Mission planner to generate a vector of waypoints, a velocity PI controller and proportional heading controller were designed, simulated and verified within *Simulink*. The FPGA design process was utilized in order to convert these algorithms into the available System Generator blocks. After verification from the simulated hardware, the design was then downloaded into the prototype board containing a Xilinx FPGA for Hardware-in-the-Loop verification. This design example demonstrated the validity of the presented approach as an efficient and reliable method for rapid system prototyping of control theory in the area of unmanned systems.

Chapter Two introduces the MATLAB and *Simulink* software environment, followed by an explanation of the workings of the toolboxes available within System Generator. Chapter Three gives an overview of the prototyping board utilized with this research, along with a brief description of the complimentary autopilot platform under development. An overview of the RC-Truck model, mission planner and control design is presented in Chapter Four. Chapter Five discusses the key issues with implementing control algorithms on an FPGA platform and then presents the proposed design approach. This approach is verified in Chapter Six by following the specified procedure to implement and verify the RC-Truck control design on the FPGA prototyping board. Chapter Seven completed this presented material with an overview of the knowledge obtained and recommendations for future work.

CHAPTER TWO

SOFTWARE PLATFORMS FOR SYSTEM DESIGN

An efficient rapid system prototyping environment demands a feasible and efficient development environment in which the hardware and software modules can be co-designed, co-debugged, and co-verified. The integrated software design platform containing MATLAB R2007a with *Simulink* from MathWorks, System Generator 9.2 for DSP and ISE 9.2 from Xilinx present such capabilities. Although the Xilinx ISE 9.2 foundation software is not directly utilized, it is required due to the fact that it is running in the background when the System Generator blocks are implemented. An overview of the complete design environment is presented in Figure 1.

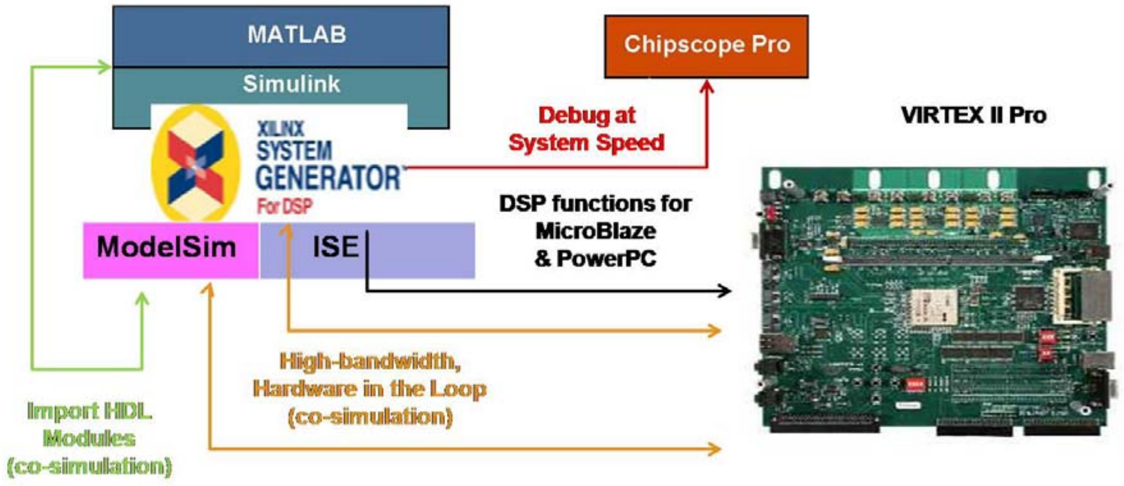

MATLAB
Simulink
XILINX SYSTEM GENERATOR For DSP
Chipscope Pro
Debug at System Speed
VIRTEX II Pro
DSP functions for MicroBlaze & PowerPC
ModelSim
ISE
High-bandwidth, Hardware in the Loop (co-simulation)
Import HDL Modules (co-simulation)

Figure 1 System Generator Block Diagram

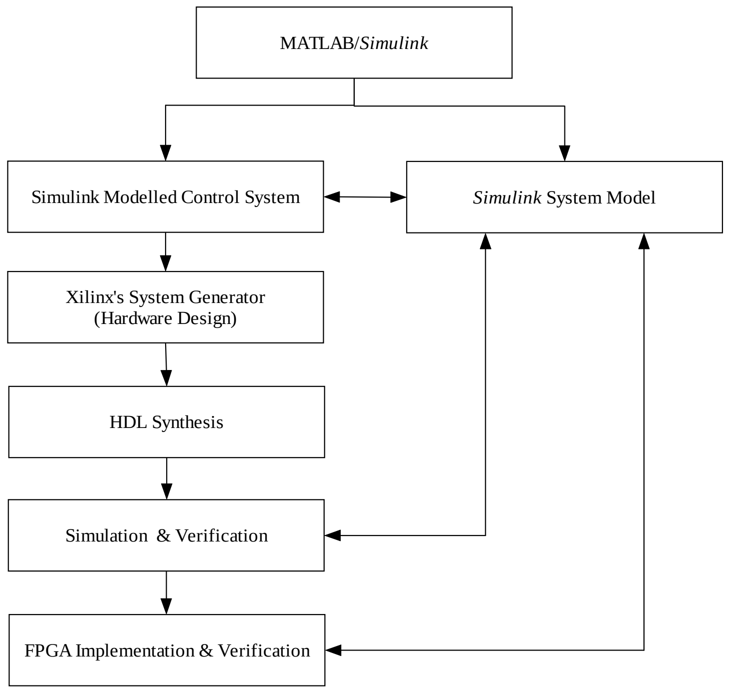
2.1 The MathWorks MATLAB® and Simulink®

MATLAB is an interactive software for doing numerical computations to simplify the implementation of linear algebra routines. Powerful operations can be performed by utilizing the provided MATLAB commands. *Simulink* is an additional MATLAB toolbox that provides for modeling, simulating and analyzing dynamic systems from within a graphical environment. This software allows for both modular and hierarchical models to be developed providing the advantage of developing a complex system design that is conceptually simplified. Due to this modular, simplified high level approach, *Simulink* has gained popularity among engineers and researchers for development, verification and modification of control algorithms [1]. Because of this wide-spread use, the ability to design and verify hardware implementation from within this same software environment becomes a great advantage for rapid prototyping of new theory and designs. In addition, the capability for hardware-in-the-loop simulation with the *Simulink* plant models provides the additional benefit of this verification to take place without risking the loss of hardware. Because the software this final verification to take place through the use of standard computer ports no additional data acquisition hardware is required. This presents a far more cost efficient solution than other methodologies. It is because of these advantages that the *Simulink*/System Generator environment was selected as the best available development platform for this project.

2.2 Xilinx System Generator for DSP

Xilinx System Generator is a MATLAB/*Simulink*-based design tool for Xilinx's line of FPGAs. Typically complicated digital circuits have been developed using multiple Hardware Description Language (HDL) modules. Because the level of abstraction is very low within the HDL environment, the difficulty increases as the design becomes more complex. These designs typically contain such considerations as feedback, word length requirements and delays. Because of the graphical nature of System Generator, the overall design is able to be viewed as a modular system with a high level of abstraction that does not require HDL code from the designer. For those designers already familiar with HDL, System Generator does provide an additional capability of allowing pre-developed HDL modules to be incorporated directly into the System Generator model. In addition, the integration with *Simulink*, provides for the hardware design and verification to be performed from within the same environment as the mathematical system model, reducing both the required design time and hardware resources [2].

Particularly relevant to this project, is the ability for the hardware-in-the-loop simulation, referred to by Xilinx as “hardware co-simulation”. The integrated Xilinx ISE software provides for an automatic generation of HDL code directly from the System Generator blocks that is then mapped to the Xilinx FPGA. This underlying code is synthesized and implemented in a Xilinx FPGA in order to perform a hardware-in-the-loop verification, as defined by Xilinx as “hardware co-simulation”. Thus, System Generator provides engineers a sophisticated platform for developing, simulating and implementing bit-true and cycle-true models [2]. Figure 2 presents an overview of the software development process from within the *Simulink* environment.



MATLAB/*Simulink*
Simulink Modelled Control System
Simulink System Model
Xilinx's System Generator (Hardware Design)
HDL Synthesis
Simulation & Verification
FPGA Implementation & Verification

Figure 2 Software Design Overview

2.3 Xilinx ISE Overview

The Xilinx Integrated Software Environment (ISE) is a powerful design environment that is working in the background when implementing System Generator blocks. This software environment consists of a set of program modules, written in HDL, that are utilized to create, capture, simulate and implement digital designs in a FPGA or CPLD target device. The synthesis of these modules creates netlist files which serve as the input to the implementation module. After generating these files, the logic design is converted into a physical file that can be downloaded on the target device. The software also provides a simulation tool where the functionality, behavior and timing can be verified for users that are familiar with the ISE software.

CHAPTER THREE

FPGA PLATFORMS

This research was started in combination with a proposed off-the-shelf autopilot hardware design. The goal of the autopilot is to both provide a flexible platform for unmanned system development and a simplification of algorithm implementation by allowing for the incorporation with the System Generator programming environment [3]. Because the hardware is still under development, an available prototyping board was selected to provide the co-simulation platform. This chapter discusses both the development board and the autopilot hardware to allow for a comparison between the research platform and the final hardware platform that will be utilized when the development has been completed.

3.1 Virtex II Pro Development Board

Diligent's Xilinx University Program Virtex-II Pro Development System, the XUP board, was selected as the hardware platform for this research. The XUP board is a powerful, multipurpose and low-cost system, which consists of a high performance Virtex-II Pro FPGA with PowerPC cores and a comprehensive collection of supporting components, such as on-board Ethernet device, serial ports and AC-97 audio codec [4]. The Virtex –II Pro FPGA consists of the following logic building blocks; 13,969 slices, 428KB distributed RAM, 136 Multiplier Blocks, 2448 KB of Block RAM and 2 PowerPC RISC Cores. The board provides 100MHz system clock which improves the performance of any complicated module. The development system also includes an embedded USB 2.0 microcontroller capable of communications with other USB hosts. This interface is used for programming or configuring the Virtex-II Pro FPGA in Boundary-Scan mode. Communication clock speeds are selectable from 750 kHz to 24 MHz.. The USB 2.0 microcontroller attaches to a desktop or laptop PC high-speed A-B USB cable.

Onboard external devices such as program memory and analog to digital converters that directly connects to the FPGA are also available. Although not necessary for the co-simulation verification, these peripheral devices can be used by defining the controls and interface logic from within the System Generator design. Because of the available program memory, the FPGA can be configured by the bit stream stored within this memory during the power up phase or directly through the volatile internal flash memory by utilizing the embedded USB2 high speed interface.

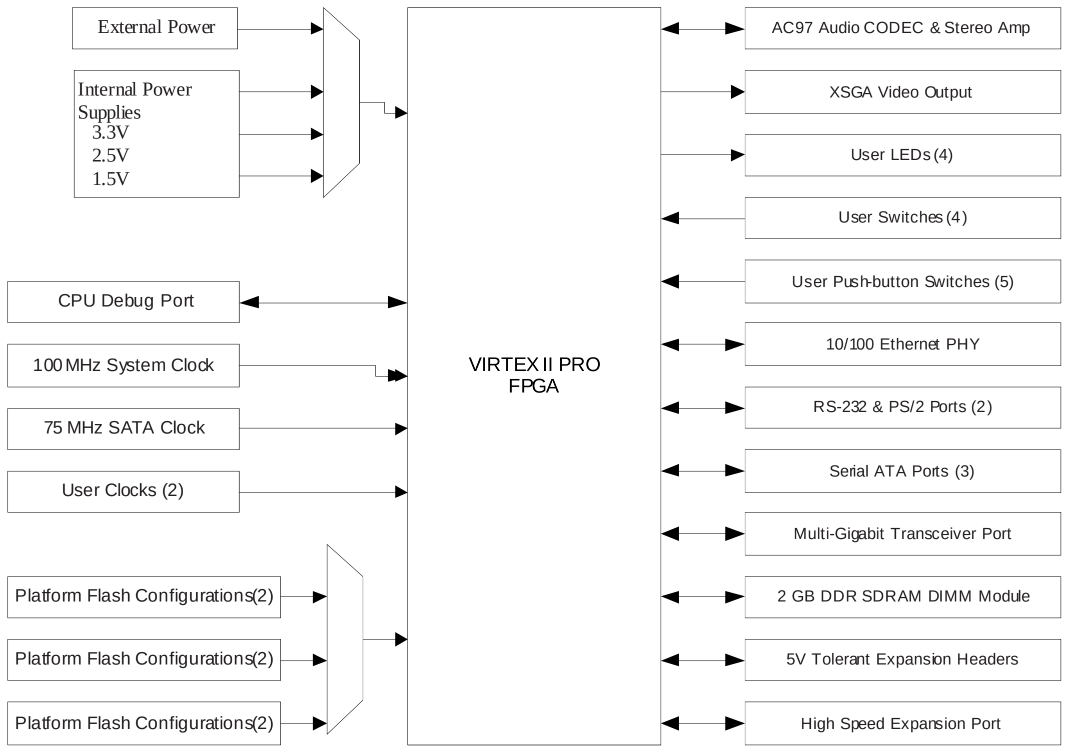

External Power
Internal Power Supplies
3.3V
2.5V
1.5V
CPU Debug Port
100 MHz System Clock
75 MHz SATA Clock
User Clocks (2)
Platform Flash Configurations(2)
Platform Flash Configurations(2)
Platform Flash Configurations(2)
VIRTEX II PRO FPGA
AC97 Audio CODEC & Stereo Amp
XSGA Video Output
User LEDs (4)
User Switches (4)
User Push-button Switches (5)
10/100 Ethernet PHY
RS-232 & PS/2 Ports (2)
Serial ATA Ports (3)
Multi-Gigabit Transceiver Port
2 GB DDR SDRAM DIMM Module
5V Tolerant Expansion Headers
High Speed Expansion Port

Figure 3 Block Diagram of the XUP Virtex-II Pro Development System

3.2 Overview of Autopilot

The autopilot hardware design that is currently under development also utilizes a Xilinx FPGA and has surrounding peripherals. However, the surrounding hardware on this design is specific for use with unmanned systems. An overview of the peripheral hardware is presented in Figure 4. This hardware includes the following;

- On board pressure sensors for a measurement of forward velocity and altitude
- A Field Programmable Analog Array to allow for flexibility in analog sensor inputs
- Digital I/O ports that can be programmed to accept voltage levels ranging from 1.8 to 5 volts

- Digital 3.3 Volt I/O ports for a custom daughter board connection
- SPI flash memory for data acquisitions
- RS232 ports that allow for communication with an external processing system
- A built in safety switch to allow take-over of the actuators by a human pilot
- A standard JTAG connector for both programming and hardware co-simulation

It was not necessary to include external program memory because the selected FPGA, the Spartan3 1400AN, has non-volatile program memory residing within the FPGA.

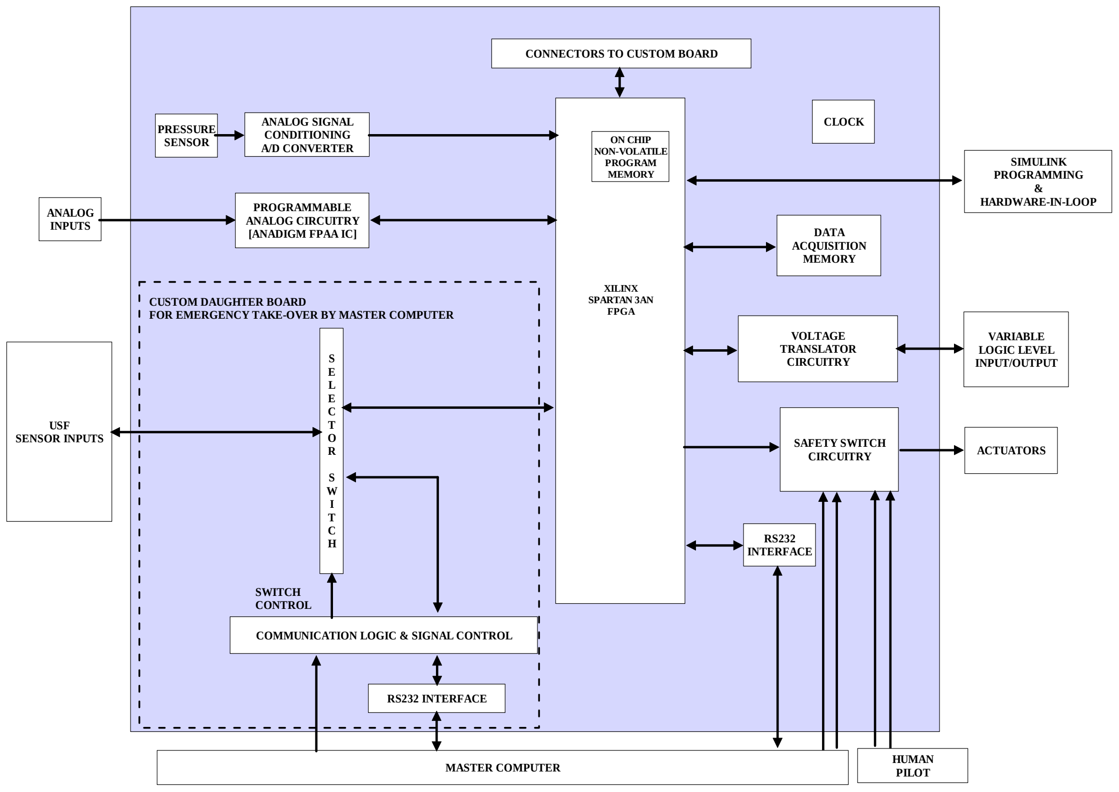

CONNECTORS TO CUSTOM BOARD
PRESSURE SENSOR
ANALOG SIGNAL CONDITIONING A/D CONVERTER
ON CHIP NON-VOLATILE PROGRAM MEMORY
CLOCK
SIMULINK PROGRAMMING & HARDWARE-IN-LOOP
ANALOG INPUTS
PROGRAMMABLE ANALOG CIRCUITRY [ANADIGM FPAA IC]
DATA ACQUISITION MEMORY
XILINX SPARTAN 3AN FPGA
CUSTOM DAUGHTER BOARD FOR EMERGENCY TAKE-OVER BY MASTER COMPUTER
SELECTOR SWITCH
VOLTAGE TRANSLATOR CIRCUITRY
VARIABLE LOGIC LEVEL INPUT/OUTPUT
USF SENSOR INPUTS
SAFETY SWITCH CIRCUITRY
ACTUATORS
RS232 INTERFACE
SWITCH CONTROL
COMMUNICATION LOGIC & SIGNAL CONTROL
RS232 INTERFACE
MASTER COMPUTER
HUMAN PILOT

Figure 4 Autopilot Hardware Overview

The selected FPGA consists of the following logic building blocks; 11,264 slices, 32 multipliers, 176K distributed RAM, 576K RAM Block and 1,400K system gates. Although the Spartan series does not include embedded PowerPCs, Xilinx's EDK program can be utilized to provide soft core DSPs.

3.3 Comparison of Hardware Platforms

Although there are major differences in the peripherals available on the XUP board and the autopilot, the primary focus of this work is the conversion of *Simulink* to System Generator, which does not utilize this hardware. Of concern for portability of this work to the autopilot design is the actual FPGA utilized for the hardware co-simulation. Because the extra functionality of the built in PowerPCs cannot be accessed from within System Generator, the portion of the XUP FPGA utilized in the co-simulation is similar to the autopilot FPGA in number of available logic gates and accessible RAM. In addition both the XUP board and the autopilot operate at a frequency of 100 MHz. Because of the similarities of the two FPGAs and the clock rate, the XUP board is a sufficient platform for developing the design process that will be utilized with the autopilot in future work. In addition the work presented is general in its nature and applicable to a wide variety of applications.

CHAPTER FOUR

OVERVIEW OF RC-TRUCK MODEL/SYSTEM

A *Simulink* model of an RC-truck robot with Ackerman steering was developed using the equations given in [5] as the physical system under study. These equations can be divided into separate portions of the overall robot model; the motor, the forward dynamics and the kinematics, Figure 5.

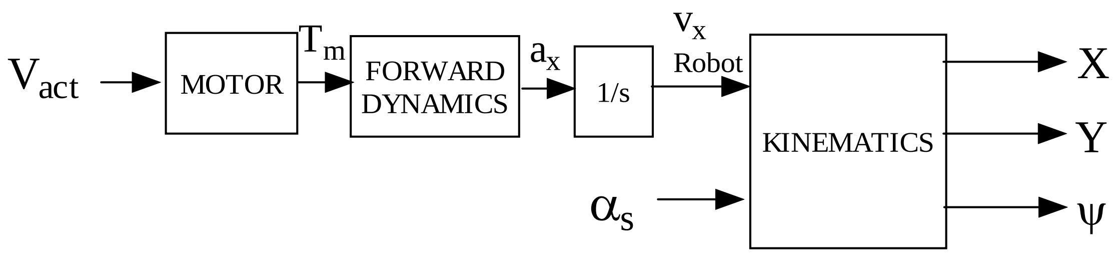


Figure 5 Open Loop RC-Truck Model

In order to complete the system, a controls design containing a mission planner, velocity control and heading control where developed and verified within the *Simulink* environment. This system was then used to confirm the effectiveness of the design procedure presented in Chapter Five.

4.1 Forward Body-Reference Dynamics

The primary force on the truck is the forward motion due to the torque produced by the motor. Other forces acting on the robot, such as ground resistance, wind or uneven ground, were not included in the model. Because this work presents a study into the design of FPGA hardware implementation of the control of a pre-developed system model, the simplifications are acceptable. The calculation of the force providing movement in the forward direction is given in Equation(4.1) where $T_e(t)$ is the torque produced by the motor, N_{mw} is the motor to wheel ratio and r is the radius of the tire. The forward body-reference velocity is obtained by integrating this force and dividing by the mass of the vehicle, Equation(4.2).

$$F_x(t) = \frac{T_e(t)}{\mathrm{N_{mw}r}} \tag{4.1}$$

$$V_x(t) = \int_0^t \frac{F_x(t)}{\mathrm{M}} dt \tag{4.2}$$

4.2 Motor Model

Equations(4.3),(4.4) and (4.5) are used to model the electric motor of the RC-truck. The input variable, or control variable, is the motor voltage. The output of the motor model is the torque that is applied to the drive train of the RC-truck. The constants relating to the motor specifications are as follows; R is the electrical resistance, L is the electrical inductance, K_t is the motor torque constant, K_v is the motor voltage constant, and J is the motor inertia. The motor variables are the current, *i(t),* the angular velocity, $\omega(t)$, the input voltage $V_{in}(t)$, and the output torque, $T_e(t)$.

$$\frac{di(t)}{dt} = -\frac{\mathrm{R}}{\mathrm{L}} i(t) - \frac{1}{\mathrm{K_v L}} \omega(t) + \frac{V_{in}(t)}{\mathrm{L}} \tag{4.3}$$

$$\frac{d\omega(t)}{dt} = \frac{\mathrm{K_t}}{\mathrm{J}} i(t) - \frac{\mathrm{K_v}}{\mathrm{J}} \omega(t) \tag{4.4}$$

$$T_e(t) = \mathrm{K_t} i(t) \tag{4.5}$$

4.3 Kinematic Calculations

Kinematic equations for a bicycle model have been used to convert the motion along the x-body axis to robot's position in the world reference frame. These equations are given in Equations (4.6), (4.7), and (4.8) where $v_s(t)$ is equal to the velocity in the body reference frame, L is the distance between the center of the front and back wheels, $\alpha_s(t)$ is the steering angle of the front tires, and $\psi(t)$ is the heading in the world reference frame. In addition, the steering angle of the truck has been limited to *+/-30* degrees due to the physical limitations of Ackerman steering.

$$X(t) = \int_0^t v_s(t)\cos(\alpha_s(t))\cos(\psi(t))dt \tag{4.6}$$

$$Y(t) = \int_0^t v_s(t)\cos(\alpha_s(t))\sin(\psi(t))dt \tag{4.7}$$

$$\psi(t) = \int_0^t \frac{v_s(t)}{\mathrm{L}}\sin(\alpha_s(t))dt \tag{4.8}$$

4.4 Control System

The control system comprises of a mission planner, a heading controller and a velocity controller, Figure 6. The velocity controller is proportional-integral (PI) control, Equation(4.9) with the proportional gain, K_P equal to 0.15 and the integral gain, K_I, equal to 0.002. The error between a measured process variable and a desired set point is corrected by choosing gain parameters appropriately. The proportional gain determines the reaction to the current error and the integral gain determines the reaction based on the sum of recent errors. The heading control is a proportional controller, Equation(4.10), with K_P equal to 1. This forces the wheels to turn in the direction of the error. With a large error the turning angle is limited to the maximum 30 degrees, which is representative of the true behavior of the vehicle.

$$K_P e(t) + K_I \int e(t)dt \tag{4.9}$$

$$K_P e(t) \tag{4.10}$$

The mission planner is contained in a single m-file, given in Figure 7. The individual way points are contained two vectors, X_{traj} and Y_{traj}, with the index of the arrays as the way point number. The distance is calculated so that when the robot is close to the current way point, the next way point is sent out as the mission planner as the X and Y position set point.

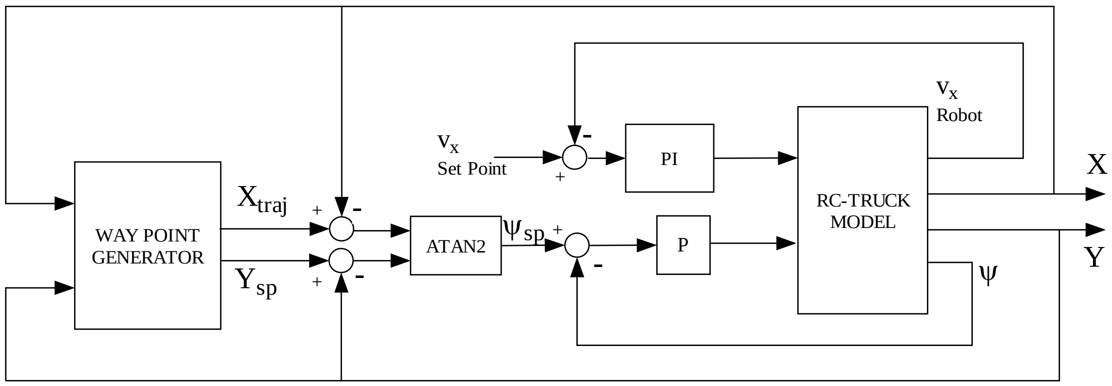
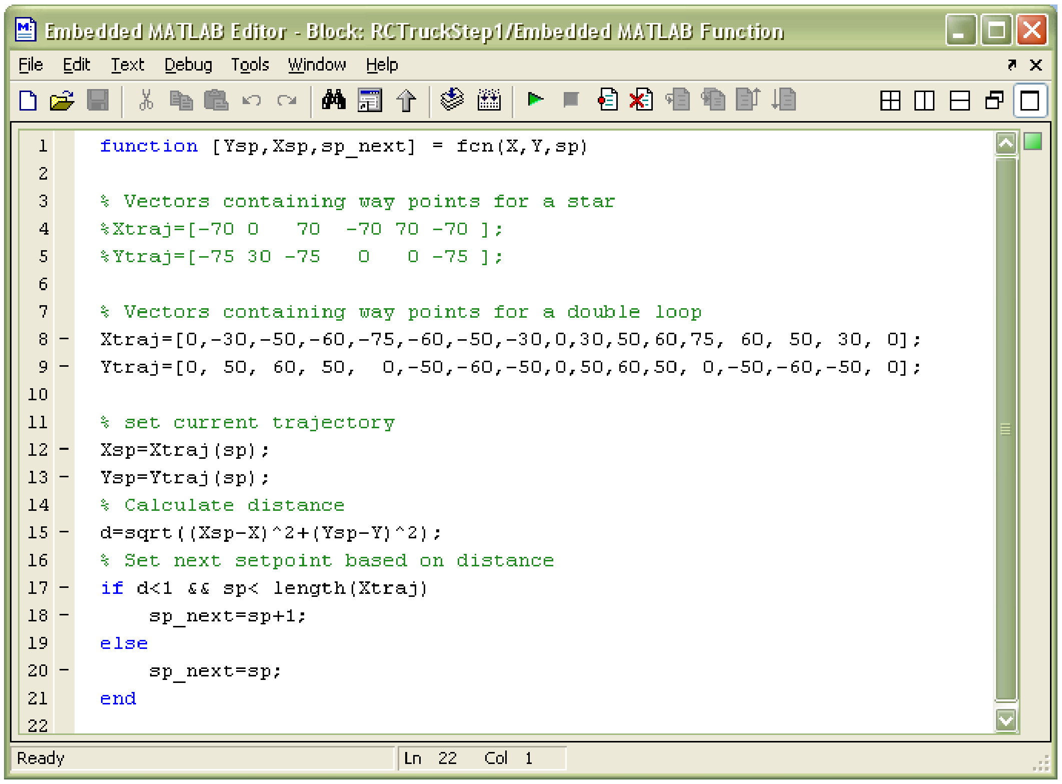


Figure 6 RC-Truck System


Embedded MATLAB Editor - Block: RCTruckStep1/Embedded MATLAB Function

```
function [Ysp,Xsp,sp_next] = fcn(X,Y,sp)

% Vectors containing way points for a star
%Xtraj=[-70 0   70  -70 70 -70 ];
%Ytraj=[-75 30 -75   0   0 -75 ];

% Vectors containing way points for a double loop
Xtraj=[0,-30,-50,-60,-75,-60,-50,-30,0,30,50,60,75, 60, 50, 30, 0];
Ytraj=[0, 50, 60, 50,  0,-50,-60,-50,0,50,60,50, 0,-50,-60,-50, 0];

% set current trajectory
Xsp=Xtraj(sp);
Ysp=Ytraj(sp);
% Calculate distance
d=sqrt((Xsp-X)^2+(Ysp-Y)^2);
% Set next setpoint based on distance
if d<1 && sp< length(Xtraj)
    sp_next=sp+1;
else
    sp_next=sp;
end

```

Ready   Ln 22   Col 1

Figure 7 Mission Planner M-File

CHAPTER FIVE

DESIGN PROCESS

Several issues create difficulties in the process of converting the *Simulink* design to the FPGA implementation due to the underlying hardware characteristics of the available System Generator blocks. The designer must give careful consideration to certain issues such as timing synchronization, delays associated with complicated mathematical calculations, and conversion to fixed point. After consideration of these issues and the tools provided by System Generator, a design procedure was derived for systematically converting the *Simulink* design into System Generator blocks. This chapter discusses each of these issues, the System Generator tools available and presents a detailed overview of the method for development of the hardware implementation.

5.1 Timing Issues with Algebraic Loops

In the design of the mathematical algorithms, there are times when the result of one calculation must be returned to use in a comparison or cumulative type calculation. This is done in the context of an algebraic feedback loop. Logic gates that occur within this loop may have an associated delay that will affect the stability of the system that was not present in the *Simulink* mathematical model. For example, within the truck control system, the set point number is held in a register then returned to the m-file controlling the way point generation, creating a delay of one hardware clock cycle.

5.2 Math Issues

System Generator provides a math block set that includes standard functions that carry only small delay because of both the efficient algorithms that underlie the blocks and the simplicity of the math itself. This block set includes calculations such as add, subtract, shift, multiply, divide by 2^n, and the cosine and sine function. These functions can be included within the system with only a slight delay that can be easily compensated for with careful timing synchronization.

A second mathematical block set is provided that contains the Coordinate Rotation Digital Computer (CORDIC) algorithms for a few calculations that are not easily translated to the gate level. The provided blocks are for the division, log, sine, cosine, square root and inverse tangent functions. These CORDIC algorithms utilize coordinate rotations as the basis for an iterative method for the calculation of more complex math functions [6]. It is particularly suited to hardware implementations because it does not require any multiplies. CORDIC revolves around the idea of "rotating" the phase of a complex number, by multiplying it by a succession of constant values. However, the "multiplies" can all be powers of 2, so in binary arithmetic they can be done using just shifts and adds; no actual "multiplier" is needed. These provided blocks must be used with care due to the longer potential delay that may occur.

Within the blocks there are settings that allow the user to compromise between accuracy and hardware usage. The user may select the number of number of processing elements, the input data word length, and the latency for each processing element. Although there is no clear cut process for selecting this, running a few simulations with varying selections will allow a good estimation.

Although time and resource consuming math cannot always be avoided, certain considerations can be taken to minimize this cost. Whenever a divide by can be replaced by a divide by 2^n, this should be considered. For example, if a series of sensor readings are to be averaged, selecting a 2^n value should prove sufficient. In addition, selecting the minimum number of processing elements and shortest delay for the required precision within the CORDIC blocks will also reduce the cost.

5.3 Floating to Fixed Point Conversion, Quantization and Overflow Issues

The FPGA requires fixed point arithmetic that must be defined during the hardware design phase. However, a great deal of flexibility is provided by allowing the definition of signed or unsigned, word length and binary point position at any point within the logic design flow. In order to determine these settings, the designer must weigh the necessary precision against increased logic and potential delays associated with long word length.

FPGAs handle the signed numbers the same manner as a microprocessor. The signed number is represented as twos complement using a binary sequence of 1's and 0's. It is the designer's responsibility to select whether or not to utilize the signed extension bit in any selected design environment, in this case System Generator. A simple solution is to sign extend as a general rule. In many cases, since this only requires one additional bit, this provides the best solution.

Because MATLAB/*Simulink* uses floating point representation any N-bit number can have any value from -2^{N-1} to $+2^{N}$. The standard format assigns an N value of 32 which is able to provide a fractional resolution as small as $1/2^N$, equal to $2.3283(10^{-10})$, when all the bits are assigned to the binary point. The only way to assign this type of resolution throughout the fixed point design is to allocate 64-bits with 32 for the binary point. This cannot be implemented in the fixed point hardware and still maintain hardware and timing efficiency. For this reason, all the values in logic must be represented in a smaller pre-defined word length. Figure 8 demonstrates the conversion of floating point number into fixed. The decimal in the top number will adjusts as the size of the number changes. When converted to the lower representation, the decimal maintains the same position, creating potential issues when the number is either too large or too small to be accurately represented.

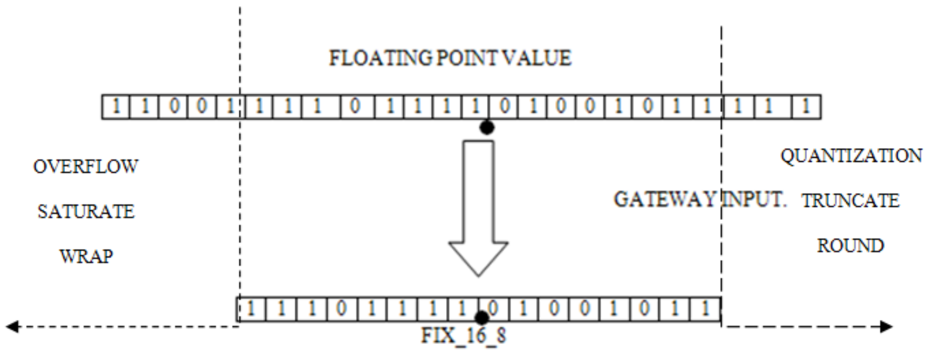
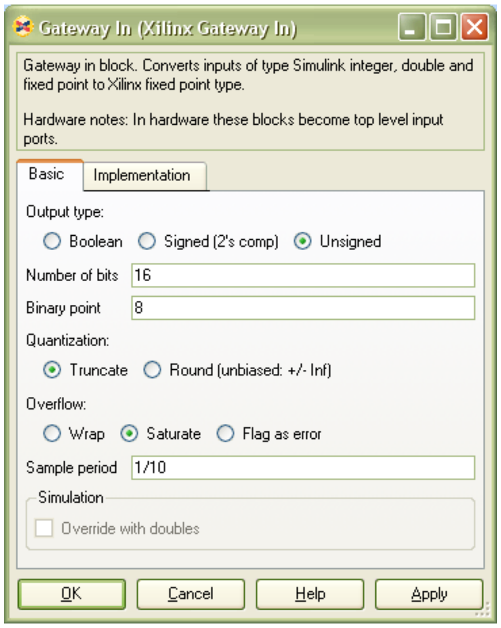


Figure 8 Floating Point to Fixed Point Conversion

The conversion from floating point to fixed-point is not a lossless transition. Two phenomena can occur, called overflow and quantization. Quantization can be handled in two ways, either truncation which discards the bits to the right of the most significant bit after the number of decimal value or rounding which estimates to the nearest representable value. Overflow occurs when the resulting output from mathematical calculation lies outside the range of the fixed-point representation set. The System

Generator blocks allow for the output values be either saturated, where the MSB's are neglected, or wrapped to the nearest value.

Within the System Generator blocks, fields are available to select the necessary settings to control the hardware with respect to word length. Using these settings the designer can choose the required type of fixed point number (signed/unsigned or Boolean), width and the position of the least significant count of decimal point. For example in the 'gateway in' block settings, Figure 9, 16 is written in the 'no of bits field' and 8 to the 'binary point' field. This directs System Generator to create a 16-bit fixed point number with eight bits reserved for the fractional portion.


Gateway In (Xilinx Gateway In)

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

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

Basic | Implementation

Output type:

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

Number of bits: 16

Binary point: 8

Quantization:

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

Overflow:

Wrap [✓ Saturate] Flag as error

Sample period: 1/10

Simulation

Override with doubles

Figure 9 Gateway In Block Settings

In terms of hardware usage, the saturate and truncate selections are preferable because they use less hardware resources as compared to round and wrap. In addition, if the word length is selected only to reflect the necessary precision, rather than all possible values, wrap should be avoided due to roll-over to the zero value on overflow.

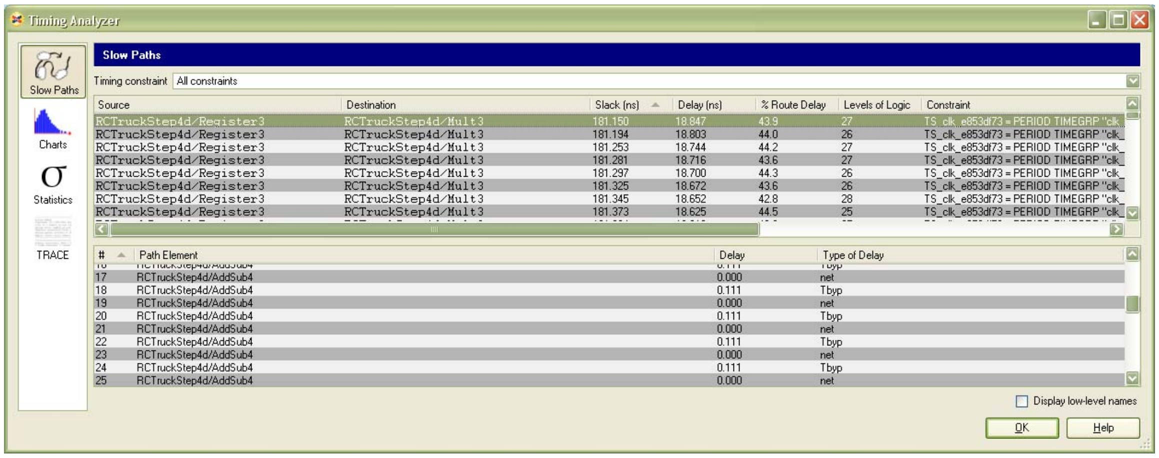
Several iterations may be required to fine tune each block in the system until both acceptable amounts of quantization error results and overflow is eliminated. This iterative analysis of quantization and overflow, along with verification within the *Simulink* environment, results in a high level software design tool.

5.4 Timing Analysis

While the system generator blocks produce a bit and cyclic true simulation, they do not take into account the timing issues that may occur when converted and download to the hardware implementation. This is an advantage because it allows testing the System Generator design before optimizing for speed and hardware usage. However, before finalization, the design must be checked with respect to the processor and clock speed to be sure that all timing constraints are met.

Timing violations occurs when the signal form one synchronous output stage does not reach the input of the next stage within the required time allocated by the system design. The System Generator timing analysis tool is provided to assist with this aspect of the hardware design. It provides a report on any slow paths within the design flow and clearly displays the specific paths that will fail in hardware.

When the timing analysis is invoked from the System Generator block, the design is compiled, netlisted into HDL source and a timing analysis run. The results appear in the System Generator timing analyzer tool, Figure 10. Selecting the histogram displays a detailed chart providing the path timing information. In addition, this display will highlight each path that does not meet the specifications. The trace icon provides the details about each specific path analyzed.



Timing constraint: All constraints

Source	Destination	Slack (ns)	Delay (ns)	% Route Delay	Levels of Logic	Constraint
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.150	18.847	43.9	27	TS_clk_e853df73 = PERIOD TIMEGRP "clk
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.194	18.803	44.0	26	TS_clk_e853df73 = PERIOD TIMEGRP "clk_
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.253	18.744	44.2	27	TS_clk_e853df73 = PERIOD TIMEGRP "clk_
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.281	18.716	43.6	27	TS_clk_e853df73 = PERIOD TIMEGRP "clk_
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.297	18.700	44.3	26	TS_clk_e853df73 = PERIOD TIMEGRP "clk_
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.325	18.672	43.6	26	TS_clk_e853df73 = PERIOD TIMEGRP "clk_
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.345	18.652	42.8	28	TS_clk_e853df73 = PERIOD TIMEGRP "clk_
RCTruckStep4d/Register3	RCTruckStep4d/Mult3	181.373	18.625	44.5	25	TS_clk_e853df73 = PERIOD TIMEGRP "clk_

#	Path Element	Delay	Type of Delay
17	RCTruckStep4d/AddSub4	0.000	net
18	RCTruckStep4d/AddSub4	0.111	Tbyp
19	RCTruckStep4d/AddSub4	0.000	net
20	RCTruckStep4d/AddSub4	0.111	Tbyp
21	RCTruckStep4d/AddSub4	0.000	net
22	RCTruckStep4d/AddSub4	0.111	Tbyp
23	RCTruckStep4d/AddSub4	0.000	net
24	RCTruckStep4d/AddSub4	0.111	Tbyp
25	RCTruckStep4d/AddSub4	0.000	net

☐ Display low-level names

Figure 10 Timing Analyzer

Once an issue is discovered both replication of registers and increased control over sampling time can be utilized for correction. Replication is often performed automatically by the tools in order to reduce the capacitance of the neat, which in turn reduces the net delay. While adding these pipelining registers does increases latency and the number of logic gates; it should be seen that this provides a balance to other portions of the design and also reduces the fan out on the replicated objects. Up-sample and down-sample blocks can be included to be sure that those portions of the design that can be operated at a slower rate are calculated this reduced rate. If these blocks are not used, then the timing analyzer will generate the over constraint error.

5.5 Hardware Co-Simulation

After the System Generator model is verified through both simulation and a timing analysis, hardware co-simulation should be performed in order to validate the design operating on the FPGA platform.

The co-simulation process uses Xilinx ISE and core generator to synthesize and generate an FPGA programming bit file from the System Generator blocks in the design.

A new system block is generated called ‘JTAG co-sim block’. This block replaces the previously used System Generator design. The hardware implementation is then executed by connecting the board to the PC, thereby, closing the loop. When the model is run, a pearl script links the Xilinx ISE and the core generator software. The Xilinx ISE program then generates the bit file and loads it into FPGA through a standard JTAG connection.

There are two selections for the System Generator Hardware co-simulation, the single-step mode and free running mode. In free-running mode, the FPGA is under the control internal clock signal on the hardware platform. In single-step mode the hardware receives the clock signal through the JTAG connection which is synchronized to the simulation environment. This allows the co-simulation operating in this mode to be bit-true and cycle true [7] to the original design while allowing the correct timing to occur between the simulated plant and the logic gates within the FPGA. For this reason, the single step mode must be selected when working with a *Simulink* system model.

5.6 Proposed Design Procedure

The design process is an iterative one that requires revisions and re-testing, see Figure 11. The hardware design process begins once the *Simulink* model containing mathematical algorithm used to control the system has been verified.

Before the hardware model is build consideration must be given to any potential simplification of complicated math to help prevent potential timing issues. This modification should be made and re-verified within the *Simulink* blocks before beginning the hardware phase. Once any mathematical modifications have checked, the design can be optimized for hardware and build with the provided System Generator blocks. The most efficient approach is to look for any effective modularity to the original design that can be brought into the hardware blocks and tested a portion at a time. This optimization of hardware involves breaking any long m-file code into smaller blocks and looking for any mathematical functions that can run in parallel. Once the System Generator model is developed, an initial check for potential timing synchronization issues should be completed, in particular with algebraic loops. In addition sampling rates at different stages of the hardware flow should be considered and ‘sample-up’ and ‘sample-down’ blocks inserted where necessary. At this stage of the design development, the word length does not necessarily need to be considered. Because the timing issues are dependent on the clock rate and specific hardware, the hardware simulation process will allow for more flexible test of the hardware design. Once the general design is verified, then the word length can be minimized in order reduce the amount of logic gates required and reduce the chances of timing issues. Once the word length modifications are checked, then a timing analysis can be run and, after any necessary corrections, a hardware co-simulation test can be performed.

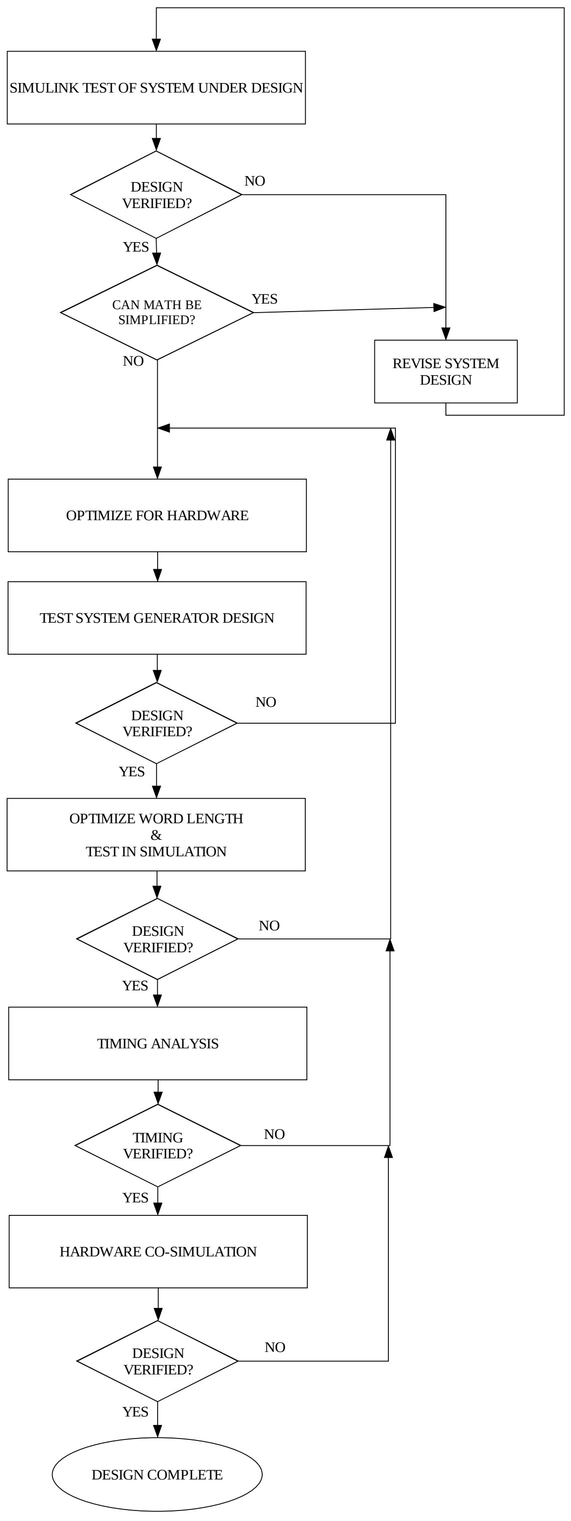


Figure 11 Design Flow

CHAPTER SIX

TESTING AND VERIFICATION

The design procedures were applied to the RC-Truck system presented in Chapter Four. This resulted in a systematic conversion of the tested control design into System Generator blocks, simulation and modification of these blocks and a final verification of the design utilizing co-simulation. This chapter presents the work done in each of the design flow steps along with the results presented by the final hardware co-simulation.

6.1 Step One: Testing Control Algorithms in *Simulink*

Figure 12 presents the *Simulink* model containing the RC-Truck model, the mission planner, the heading control and velocity control. The system was tested in the variable time step setting of *Simulink* and proved to work properly as demonstrated in Figure 14, Figure 15, Figure 16, and Figure 17.

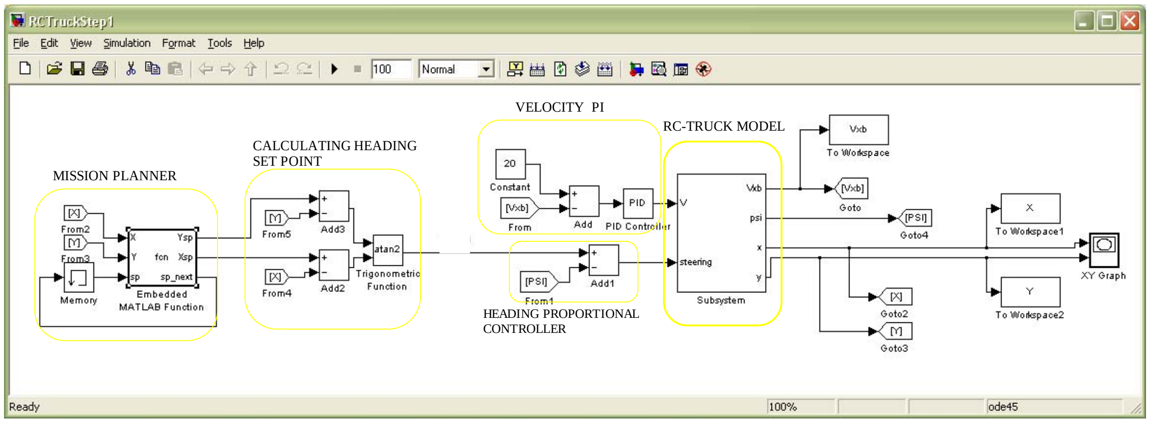
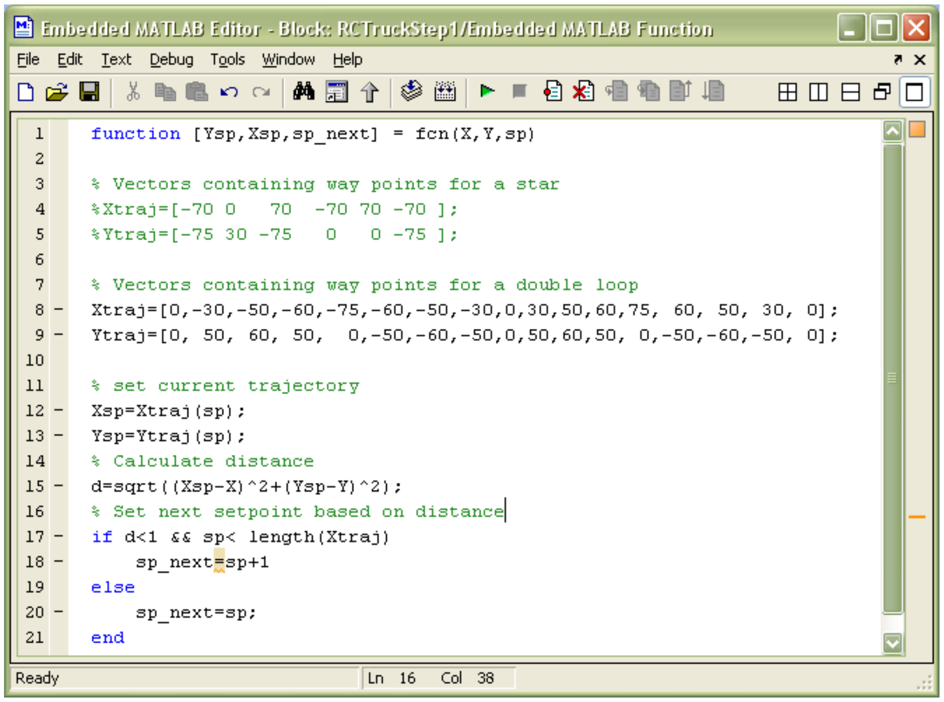


Figure 12 *Simulink* Implementation of RC-Truck System


Embedded MATLAB Editor - Block: RCTruckStep1/Embedded MATLAB Function
File Edit Text Debug Tools Window Help
Ready
Ln 16 Col 38

```
function [Ysp,Xsp,sp_next] = fcn(X,Y,sp)

% Vectors containing way points for a star
%Xtraj=[-70 0   70  -70 70 -70 ];
%Ytraj=[-75 30 -75   0   0 -75 ];

% Vectors containing way points for a double loop
Xtraj=[0,-30,-50,-60,-75,-60,-50,-30,0,30,50,60,75, 60, 50, 30, 0];
Ytraj=[0, 50, 60, 50,  0,-50,-60,-50,0,50,60,50, 0,-50,-60,-50, 0];

% set current trajectory
Xsp=Xtraj(sp);
Ysp=Ytraj(sp);
% Calculate distance
d=sqrt((Xsp-X)^2+(Ysp-Y)^2);
% Set next setpoint based on distance
if d<1 && sp< length(Xtraj)
    sp_next=sp+1
else
    sp_next=sp;
end
```

Figure 13 Mission Planner M-File

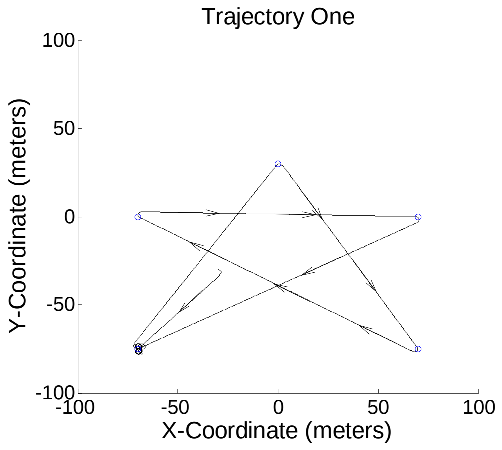


Figure 14 Star Trajectory

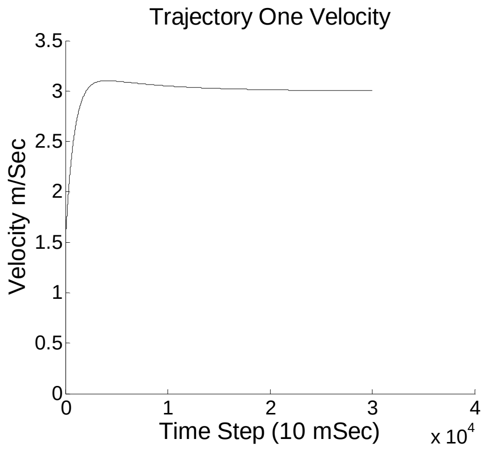


Figure 15 Star Trajectory Velocity

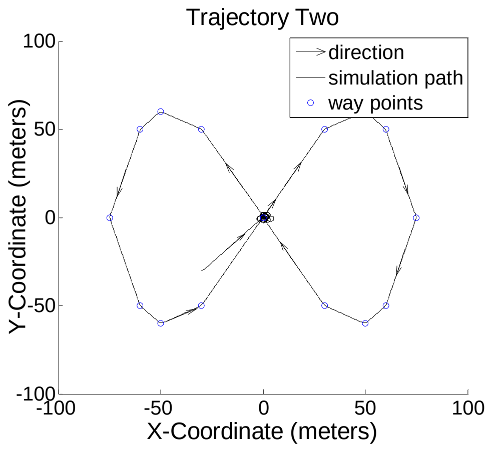


Figure 16 Figure Eight Trajectory

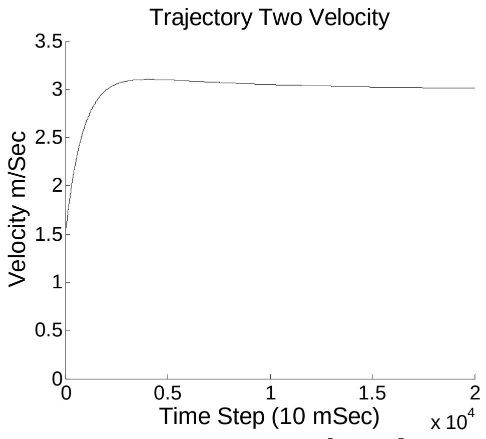


Figure 17 Figure Eight Velocity

6.2 Step Two: Simplify Any Complex Math

A potential simplification is the removal of the square root function utilized when calculating the distance. This function is a potential issue because it requires utilizing the 'cordic sqrt' block which, not only utilizes quite a bit of logic gates, but has a long delay associated with it which would occur within an algebraic loop. Because the distance is only used as a measure for determining when to increment to the next way point, an acceptable approximation can be obtained utilizing Equation(6.1). The simulation was re-run to confirm this and presented good results.

6.3 Step Three: Optimize for Hardware Characteristics

The m-file utilized for generating the way points contains the calculations for the approximation of the distance to the way point. The squared terms shown in Equation (6.1) can be calculated in parallel. For this reason, this equation was removed from the embedded m-file and calculated with System Generator blocks.

$$d = (X_{sp} - X)^2 + (Y_{sp} - Y)^2 \qquad (6.1)$$

The mission planner must calculate the next way point from the current position. This creates a feedback loop that may create some timing issues. This must accounted for when converting to System Generator blocks. The memory block was replaced by a register, which essentially works the same, except the register adds a delay of one time step. The sampling time of the input port, 0.1 sec, sets the rates of following the blocks, including the register. In order for the position set points to 'line up' with respect to the correct samples of the robot position, X and Y, a delay of one time step must also be added to both of these paths. This was accomplished by utilizing two register blocks.

When the PI controller was implemented with *Simulink* blocks, the provided PID block was used with the derivative gain set to zero. This block utilizes the correct digital algorithms and time step in order to approximate the continuous time calculations. However, implementing this controller in the hardware FPGA requires a digital version because a PID block is not provided by System Generator. The integral portion of the controller requires a digital integration algorithm, which is more complicated than the proportional which only requires a simple multiplication. The standard trapezoidal rule is implemented for the integral calculation. The controller algorithm is given in Equation(6.2), where K_P is the proportional gain and K_I is the integral gain.

$$[\mathrm{K_P} + \mathrm{K_I}\frac{z}{z-1}]E(z) \qquad (6.2)$$

6.4 Step Four: Build and Test in System Generator

The System Generator was developed in two steps, first the heading control, Figure 18, and then the velocity control logic, Figure 19. This system was tested using word lengths of 64 bits, with 32 allocated to the decimal portion and the most significant bit as the sign bit. While this would result in an excessive use of resources and more than likely cause issues with timing in the hardware, it allows a verification of the logic design before reducing the word length. An additional a change was made from the *Simulink* model within the register holding the set point number. This value was started at -1 because the delay in the loop causes the set point to be incremented immediately. This is due to the multiplier's initial output of 0. Setting the initial set point number to be equal to -1, allows a value of 0 as the starting point, causing the model to start at the first way point setting. It is also important to note that System Generator starts the vector numbering at 0, rather than the value of 1 used by *Simulink*.

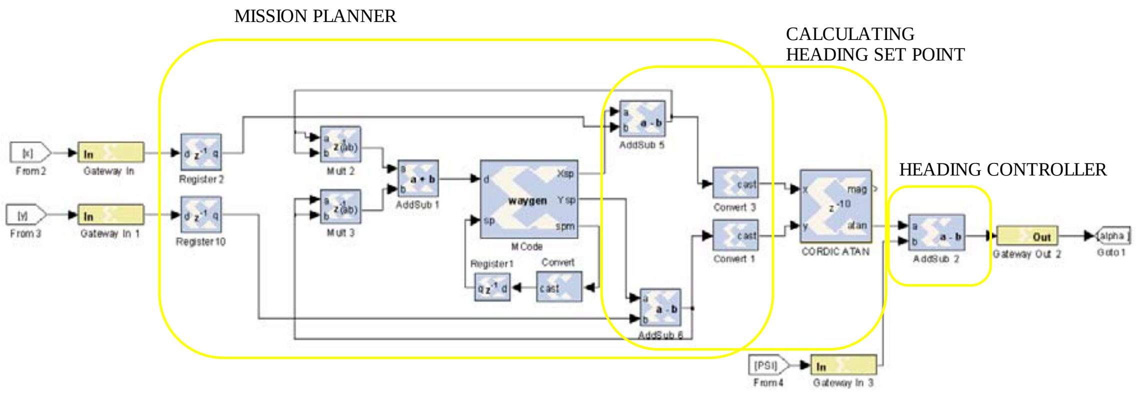


Figure 18 System Generator Mission Planner and Heading Control

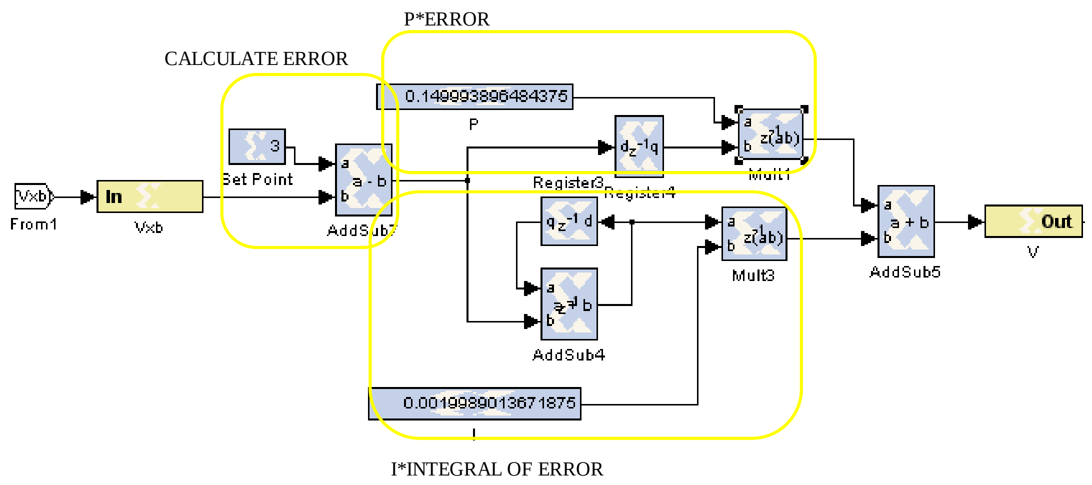


Figure 19 System Generator Velocity Control

The developed System Generator was simulated to test the hardware design before modifying the word size, Figure 20-Figure 23. It was noticed that there was a difference in the behavior of the velocity controller. It was determined that this was due to the sampling time characteristics of the digital implementation. The digital version had more of an overshoot and slightly quicker response, but still performed well.

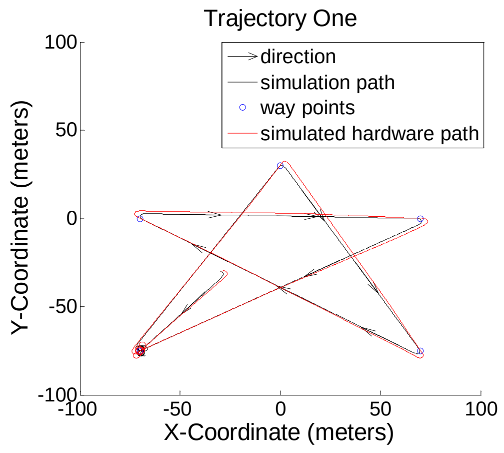


Figure 20 Star Trajectory Hardware Simulation

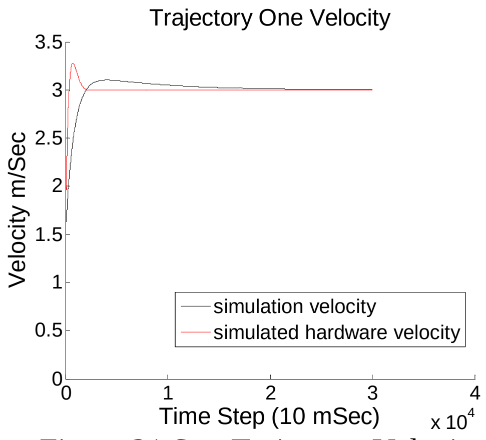


Figure 21 Star Trajectory Velocity Hardware Simulation

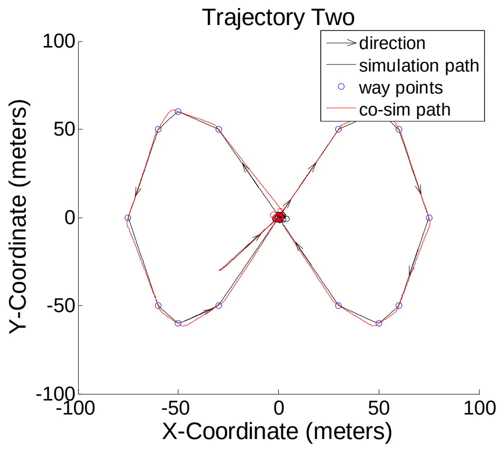


Figure 22 Figure Eight Trajectory Hardware Simulation

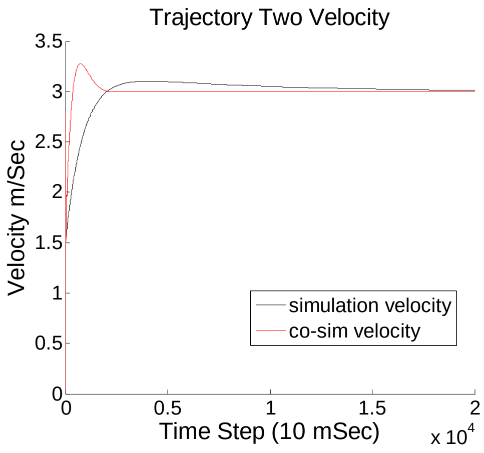
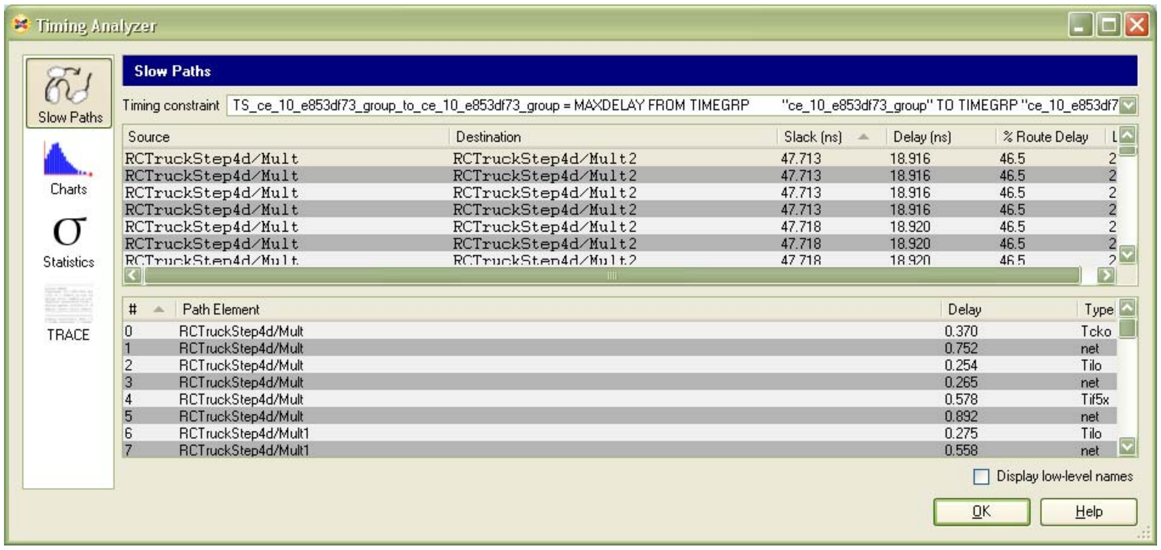


Figure 23 Figure Eight Velocity Hardware Simulation

6.5 Step Five: Calculate Word Length

The goal in determining the word length is to use the minimum length while still achieving the necessary accuracy. The input ports through which X and Y are sampled allow were first considered. It is sufficient for the RC-Truck to travel within a meter's accuracy, for this reason the decimal accuracy was limited to $1/(2^4)$. Because X and Y are limited to approximately +/-100 by the predefined trajectory area, a word length of 12 is sufficient. This allows for the 4 binary bits, 7 bits to allow for a maximum magnitude of 128 and a sign bit. The velocity input port along with the controller blocks were set to 32 bits, with 16 allocated to the decimal and the highest bit for the sign bit. The longer word length was selected due to the potential variation in values within the velocity controller calculations. A timing analysis was run, and indicated that the design met all timing constraints, Figure 24. Had the timing failed for the paths within the velocity controller, the word length could have been iteratively reduced until satisfactory results obtained.



Slow Paths

Timing constraint TS_ce_10_e853df73_group_to_ce_10_e853df73_group = MAXDELAY FROM TIMEGRP "ce_10_e853df73_group" TO TIMEGRP "ce_10_e853df7

Source	Destination	Slack (ns)	Delay (ns)	% Route Delay	L
RCTruckStep4d/Mult	RCTruckStep4d/Mult2	47.713	18.916	46.5	2
RCTruckStep4d/Mult	RCTruckStep4d/Mult2	47.713	18.916	46.5	2
RCTruckStep4d/Mult	RCTruckStep4d/Mult2	47.713	18.916	46.5	2
RCTruckStep4d/Mult	RCTruckStep4d/Mult2	47.713	18.916	46.5	2
RCTruckStep4d/Mult	RCTruckStep4d/Mult2	47.718	18.920	46.5	2
RCTruckStep4d/Mult	RCTruckStep4d/Mult2	47.718	18.920	46.5	2
RCTruckStep4d/Mult	RCTruckStep4d/Mult2	47.718	18.920	46.5	2

#	Path Element	Delay	Type
0	RCTruckStep4d/Mult	0.370	Tcko
1	RCTruckStep4d/Mult	0.752	net
2	RCTruckStep4d/Mult	0.254	Tilo
3	RCTruckStep4d/Mult	0.265	net
4	RCTruckStep4d/Mult	0.578	Tif5x
5	RCTruckStep4d/Mult	0.892	net
6	RCTruckStep4d/Mult1	0.275	Tilo
7	RCTruckStep4d/Mult1	0.558	net

☐ Display low-level names

Figure 24 Timing Analysis of RC-Truck System

6.6 Step Seven: Hardware Co-Simulation & Final Results

The final verification was completed by implementing the hardware co-simulation of the system, Figure 25. By selecting the XUP platform and implementing the 'generate' a new hardware co-simulation block is automatically. A *Simulink* library is created where the hardware co-simulation block present, Figure 26. This block is copied into the *Simulink* project file replacing all the Xilinx System Generator blocks.

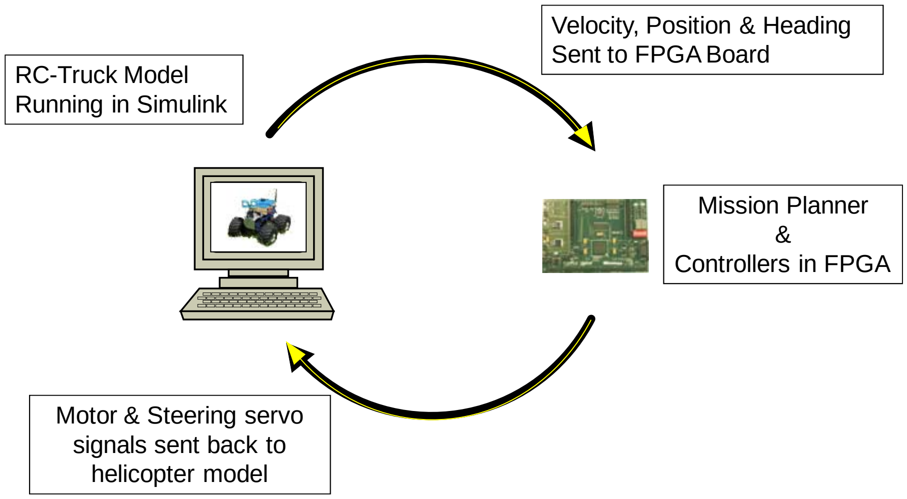


Figure 25 Hardware Co-Simulation Block Diagram

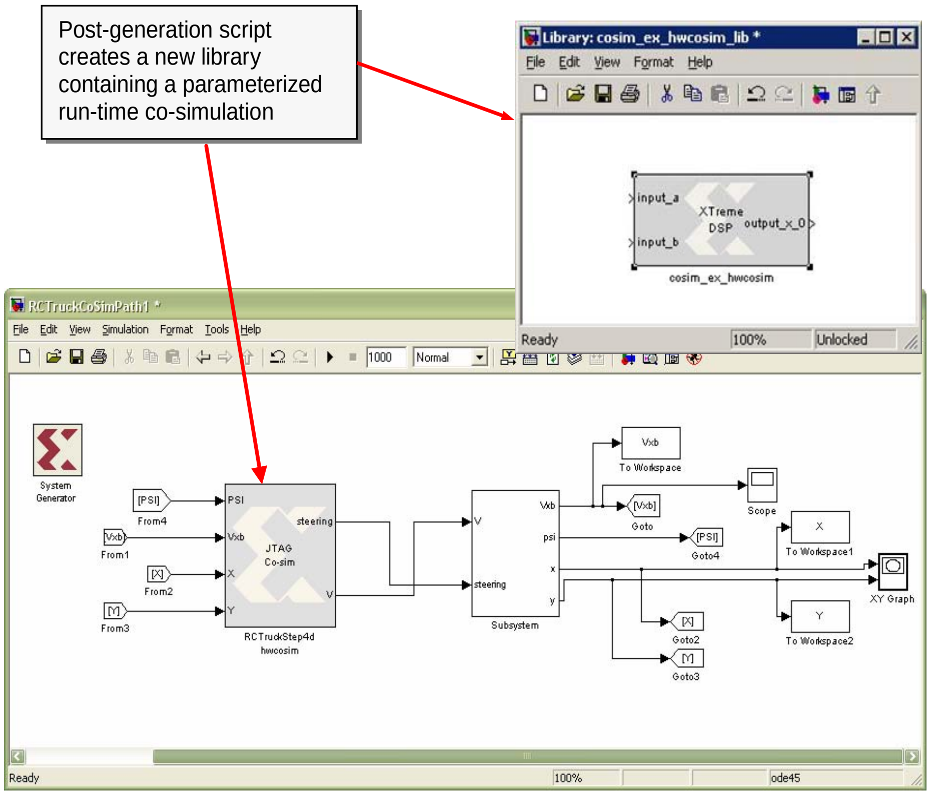


Figure 26 Hardware Co-Simulation Module

The port names on the hardware co-simulation block are matched to the port names on the original subsystem. The port types and rates also match the original design. When a value is written to one of the block's input ports, the block sends the corresponding data to the appropriate location in hardware, the controller output from the hardware is read back into the *Simulink* module using the USB interface, the output port converts the fixed data type into the *Simulink* format and fed into the model. The output plots generated is similar to the simulation path. The controller in co-simulation is tested for two different paths and the results in both the cases are shown in Figure 27, Figure 28, Figure 29 and Figure 30.

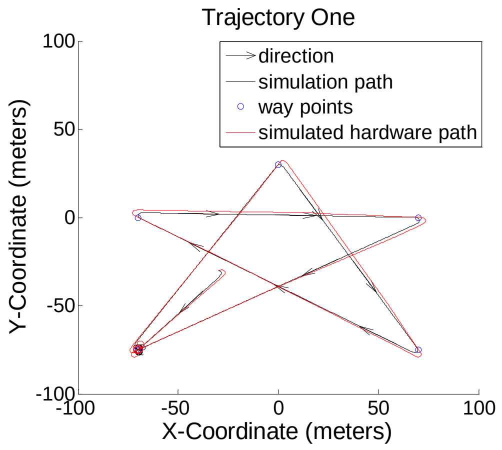


Figure 27 Star Trajectory Co-Simulation

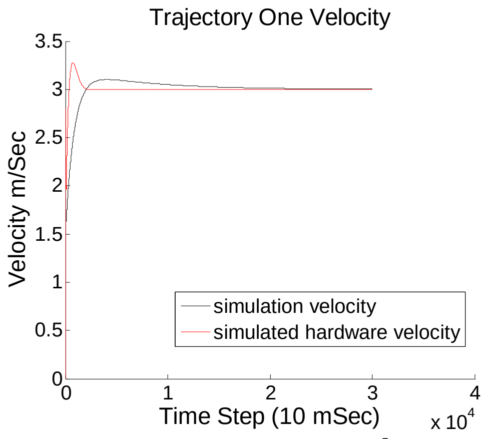


Figure 28 Star Trajectory Velocity Co-Simulation

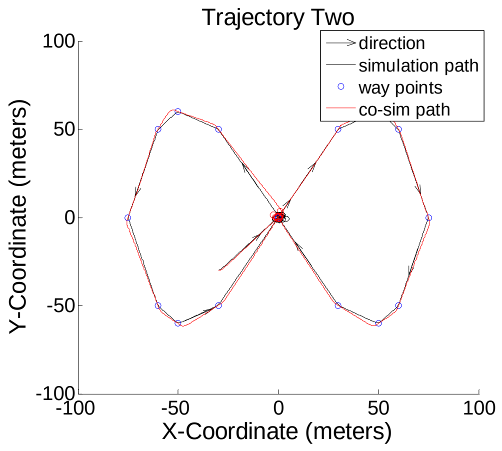


Figure 29 Figure Eight Trajectory Co-Simulation

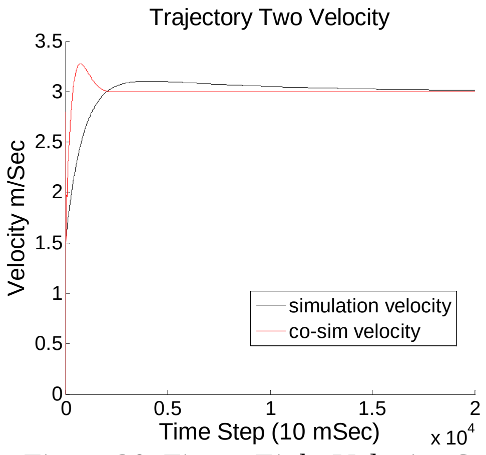


Figure 30 Figure Eight Velocity Co-Simulation

In addition the differences in error between the *Simulink* and FPGA implementation are also considered. The *Simulink* system outperformed the hardware implementation. This finding is not unexpected and can be typical of the hardware implementations due all the issues previously discussed. The goal of a designer is not to design a perfect system, but meet design specifications within a compromise between resources and precision. Had the hardware results not been acceptable, then the process would include iterations until the best compromise found. Figure 31 through Figure 38 demonstrate that for both the *Simulink* and hardware implementation, the error is less than two meters just before a way point update. This indicates that the robot comes within 2 meters for each way point along the trajectory.

Because the error jumps to a high value each time a new way point is implement as the X and Y set points, it is difficult to get a feel for the differences in error between the two implementations. In order to more accurately compare the errors, the vectors containing the magnitude of the errors was sorted in descending order and plotted in Figure 39 and Figure 40.

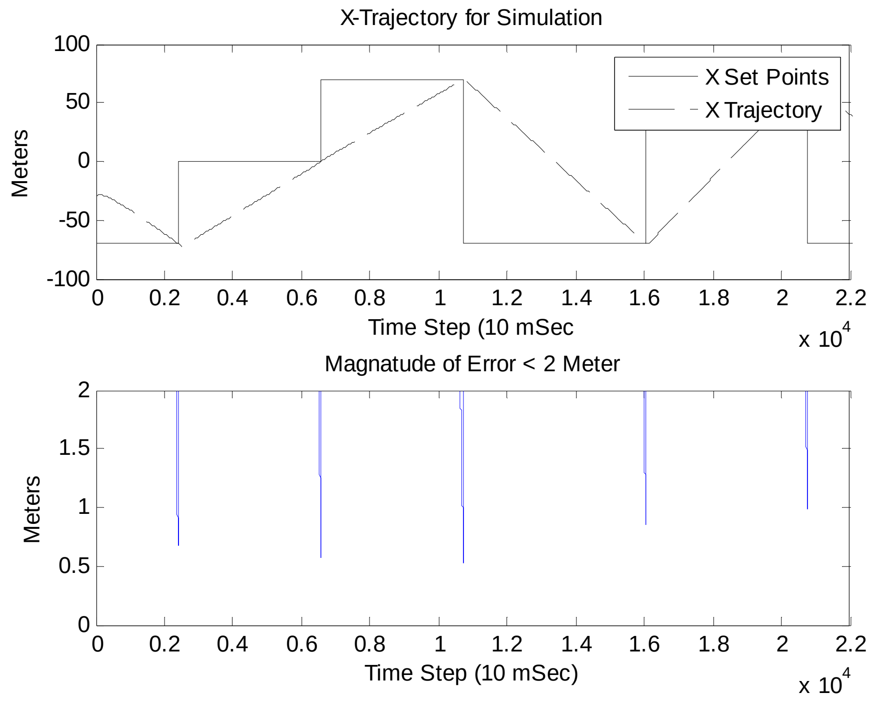


Figure 31 *Simulink* Star Trajectory X Vs. Error

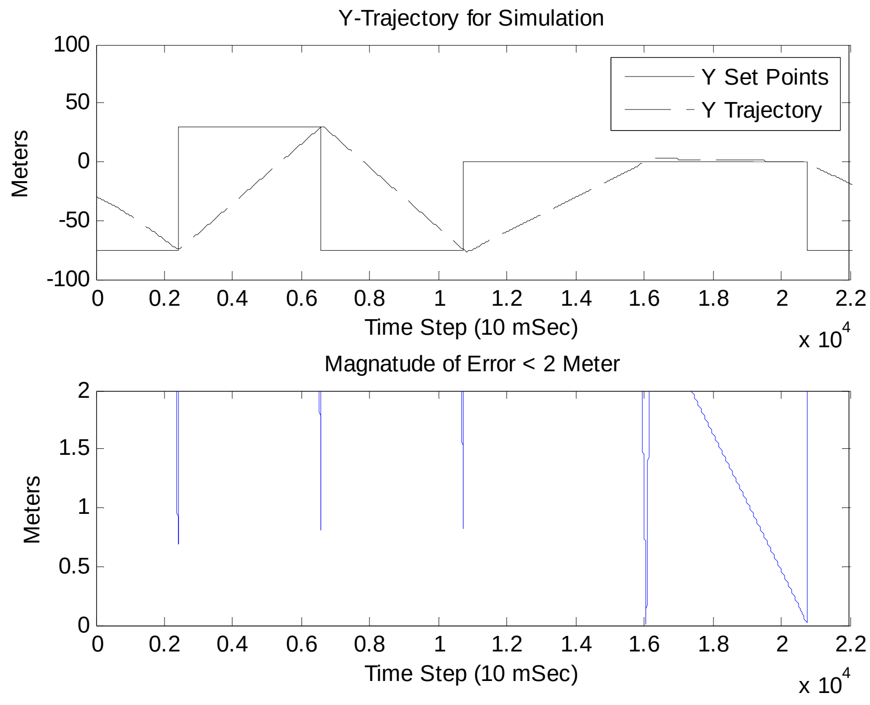


Figure 32 *Simulink* Star Trajectory Y Vs. Error

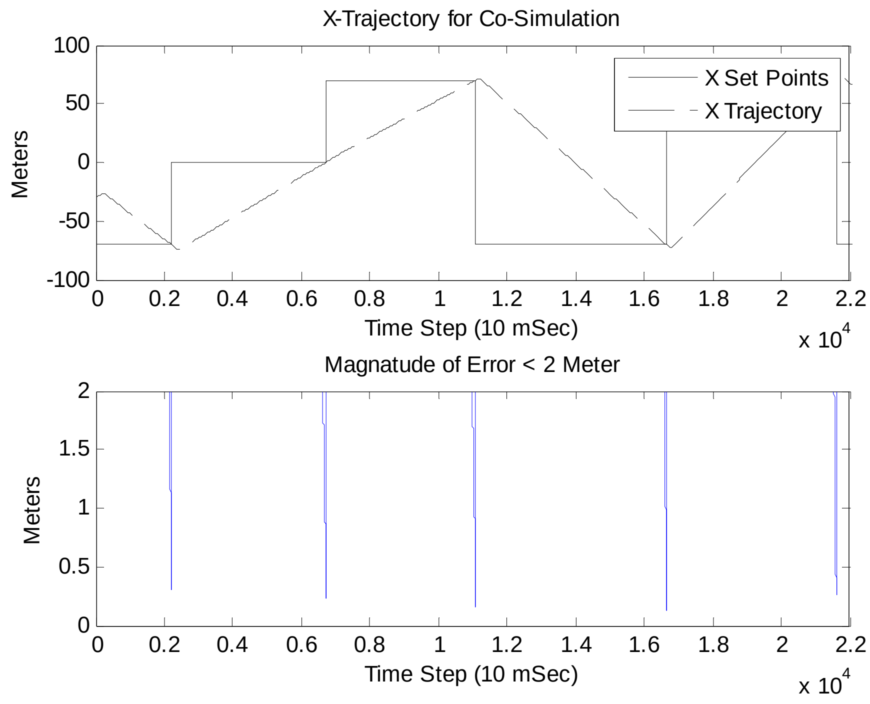


Figure 33 Co-Simulation Star Trajectory X Vs. Error

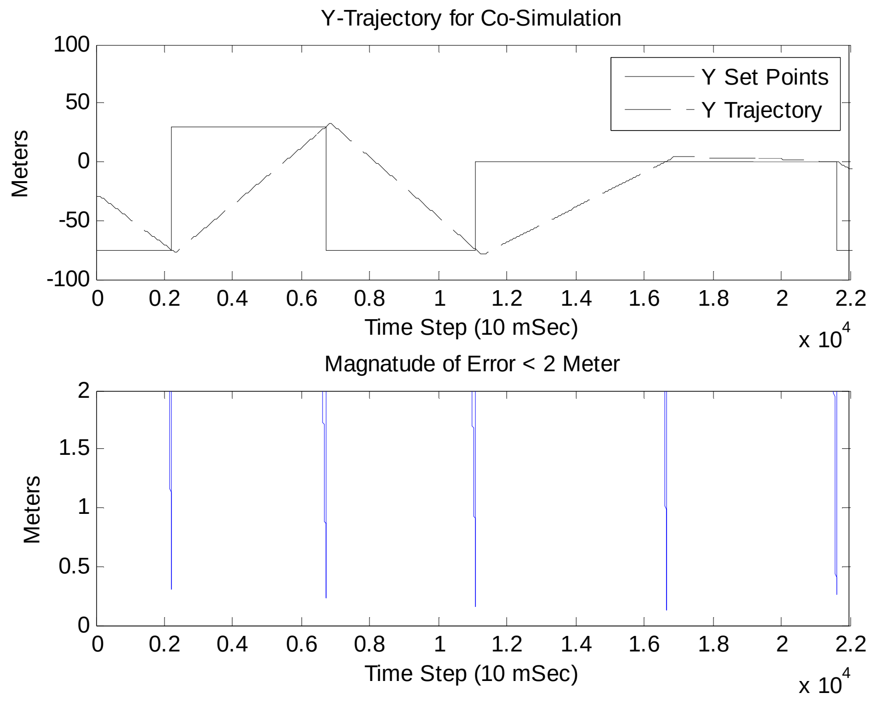


Figure 34 Co-Simulation Star Trajectory Y Vs. Error

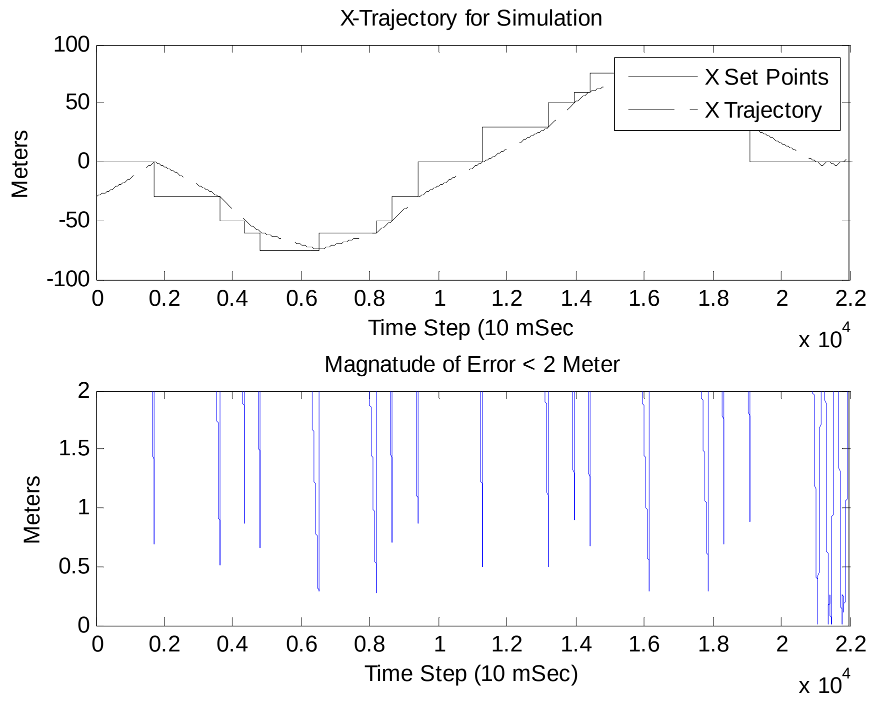


Figure 35 *Simulink* Figure Eight Trajectory X Vs. Error

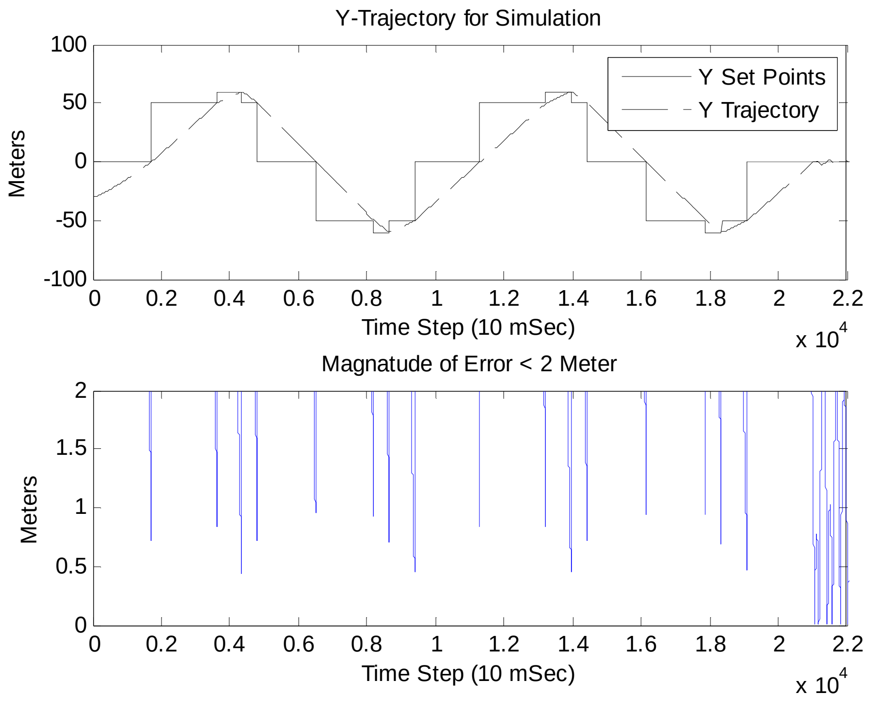


Figure 36 *Simulink* Figure Eight Trajectory Y Vs. Error

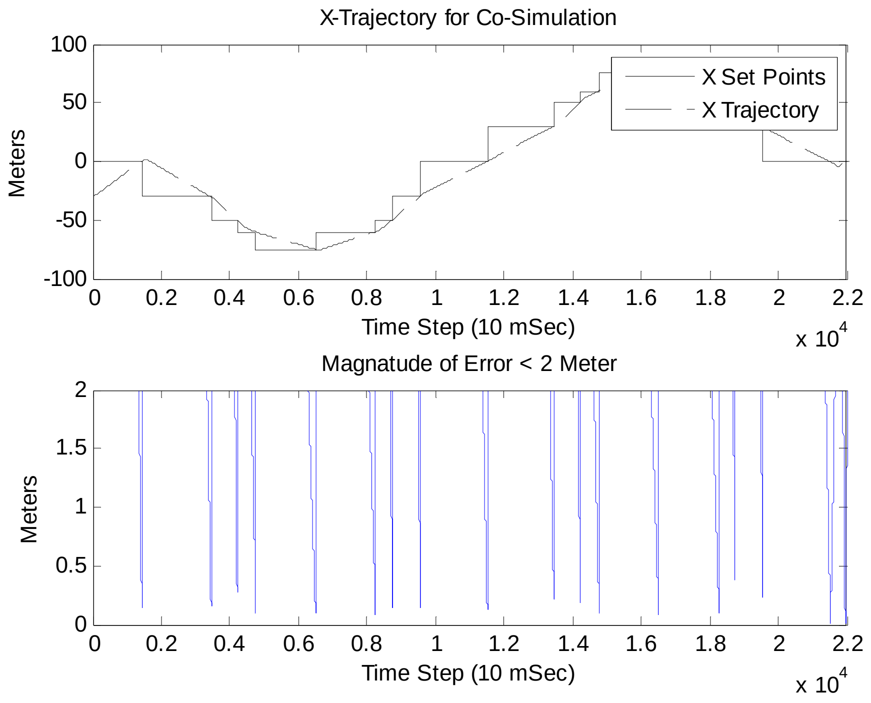


Figure 37 Co-Simulation Figure Eight Trajectory X Vs. Error

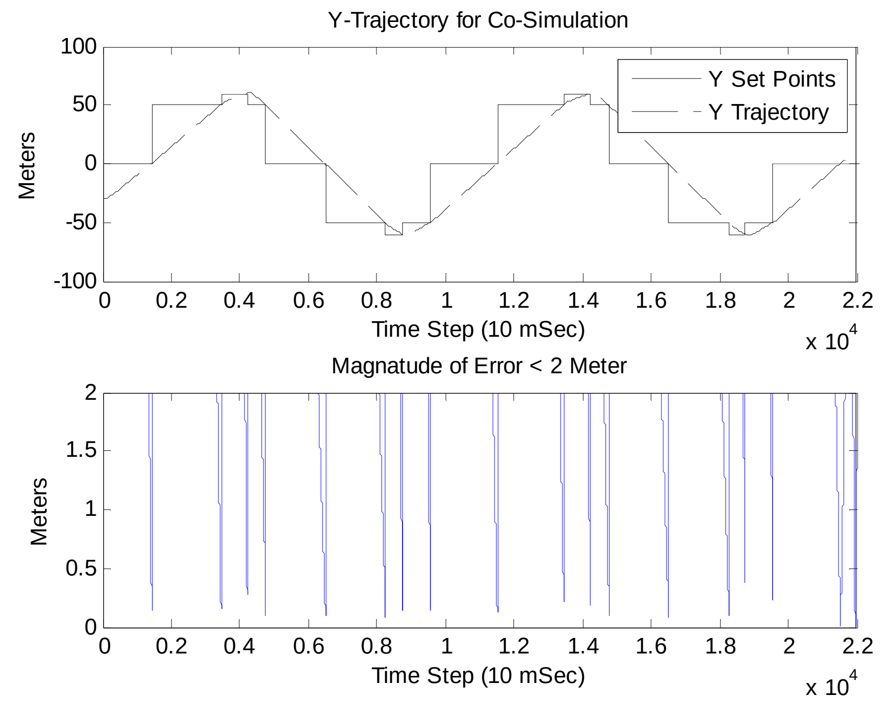


Figure 38 Co-Simulation Figure Eight Trajectory Y Vs. Error

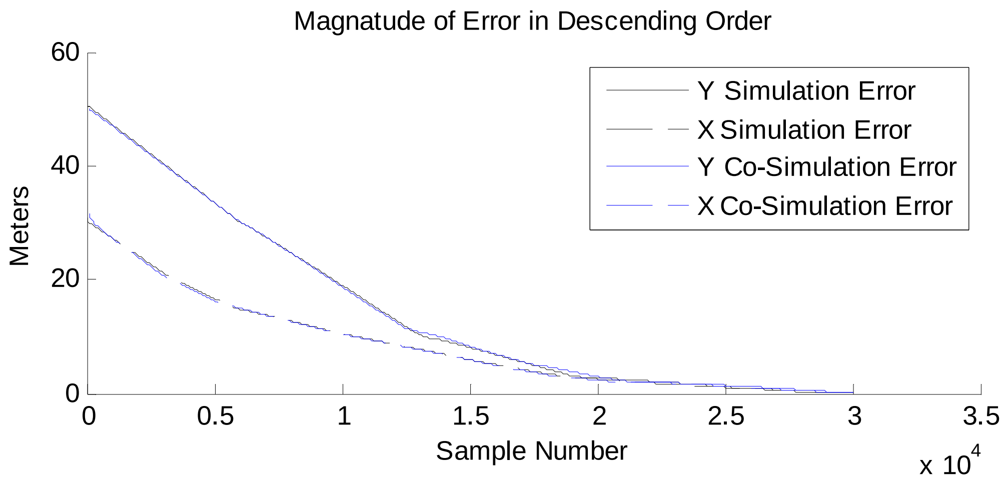


Figure 39 Errors of Star Trajectory

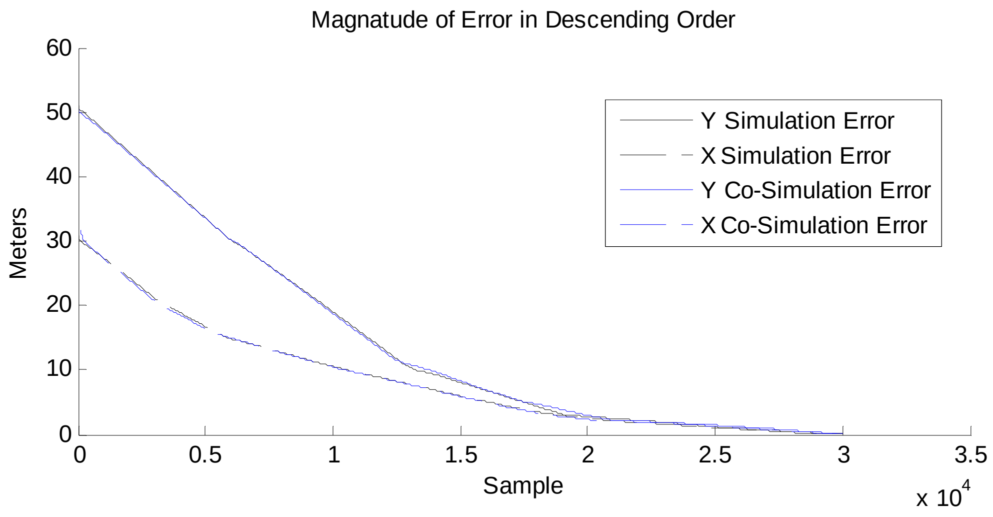


Figure 40 Errors of Figure Eight Trajectory

CHAPTER SEVEN

CONCLUSIONS AND FUTURE WORK

This rapid prototyping design flow was initiated from the idea proposed by the team of researchers from unmanned systems lab for developing a FPGA based autopilot system for research and development across multiple platforms. The work presented is intended to compliment this hardware platform by providing a systematic approach for converting designs that have been built and tested in *Simulink* to FPGA hardware implementation.

Utilizing the System Generator Environment allows the software developers to explore the design options in terms of size and speed to fulfill the design constraints. This is due to the fact that system generator allows the algorithms designed to be implemented from within the *Simulink* environment. This allows the designer the flexibility to analyze the issues that causes the error when the design is transferred from the MATLAB simulation to the FPGA. While the initial concept seems a simple one, the distinct differences between the FPGA hardware behavior and the *Simulink* simulation environment can make this a difficult task for those unfamiliar with logic/FPGA design. This research has, not only analyzed and discussed these differences, but in addition, has developed a systematic approach to the design process. Utilizing the presented methodology, along with the graphical System generator environment, is extremely simple as compared to the manual conversion when done using a hardware description language. In addition the design process was verified by utilizing it to convert an RC-Truck control system into system generator blocks and then successfully running a hardware-in-the-loop simulation.

While this work has demonstrated the effectiveness and efficiency of System Generator for the rapid systems prototyping of control systems on FPGAs, the process is still not a simple one. As the systems become more complex, for example, the inclusion of a Kalman Filtering for sensor integration and more complex controllers such as model predictive or sliding mode control, the conversion from m-file to simplistic graphical math blocks and fixed point also becomes much more difficult. Within the past few years Xilinx has also included AccelDSP to its line of software packages. This software provides assistance to the designer by converting a floating point MATLAB m-file to a fixed point *Simulink* block by running numerous iterations and comparisons to provided m-file input and output data. Future work should include research into this software to learn of the capabilities and potentially incorporate into the design flow for further simplification of this rapid system prototyping approach.

REFERENCES

[1] Mathworks, "http://www.mathworks.com."

[2] Xilinx, "User's Guide for Xilinx System Generator for DSP," 2006.

[3] W. Alvis, S. Murthy, K. Valavanis, W. Moreno, S. Katkoori, and M. Fields, "FPGA Based Flexible Autopilot Platform for Unmanned Systems," presented at 15th Mediterranean Conference on Control and Automation, Athens, Greece, 2006.

[4] Xilinx, "Xilinx University Program Virtex II-Pro Development System, Hardware Reference Manual."

[5] L. Barnes, W. Alvis, M. Fields, K. Valavanis, and W. Moreno, "Heterogeneous Swarm Formation Control Using Bivariate Normal Functions," *IEEE Workshop on Distributed Systems: Collective Intellegence and Its Applications*, pp. 85-94, 2006.

[6] R. Andraka, "A Survey of CORDIC Algorithms for FPGAs," presented at Proceedings ACM/SIGDA Conference Sixth International Symposium on Field Programmable Gate Arrays 1998.

[7] G. D. Michell and R. K. Gupta, "Hardware/software co-design," *Proc. IEEE*, vol. 85, pp. 349-365, 1997.

[8] D. N. Borys and R. Colgren, "Advances in Intellegent Autopilot Systems for Unmanned Aerial Vehicles," *2004 IEEE International Conference on Industrial Technology*, pp. 1394-1397, 2002.

[9] G. Brandmayr, G. Humer, and M. Rupp, "AUTOMATIC CO-VERIFICATION OF FPGA DESIGNS IN SIMULINK."

[10] C.A.Wisknesky, "Analysis of Xilinx FPGA Architecture and FPGA Test: A Basis for FPGA Enhanced DSP Algorithmic Acceleration and Development in Matlab/Simulink via Xilinx System Generator," *Binghamton University, State University of New York, Master Thesis*, 2004.

[11] B. Dhillon, "Optimization of DSSS Receivers Using Hardware-in-the-Loop Simulations," *The University of Tennessee*, 2005.

[12] G. Gateau, A. M. Lienhardt, and T. Meynard, "Digital sliding mode observer implementation using FPGA," *IEEE Trans. Ind. Electron.*, vol. 54, pp. 867-880, 2003.

[13] T. H. S. Li, C. Shih-Jie, and C. Yi-Xiang, "Implementation of humanlike driving skills by autonomous fuzzy behavior control on an FPGAbased car-like mobile robot," *IEEE Trans. Ind. Electron.*, vol. 50.

[14] E. Monmasson and M. N. Cirstea, "IEEE FPGA Design Methodology for Industrial Control Systems—A Review," *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, vol. 54, 2007.

[15] Xilinx, "Presentation Material & Lab Files."

[16] Xilinx, "Processing with FPGAs Workshop," *www.xilinx.com*.

[17] Xilinx, "Xilinx University Program Virtex II-Pro Development System, Hardware Reference Manual."

[18] Xilinx, "User's Guide for Xilinx System Generator for DSP," 2006.

[19] Y.-J. Yang, J.-P. Chen, J.-S. Cheng, C. Zhang, and Y.-L. Xiao, "Autonomous Micro-Helicopter Control Based on Reinforcement Learning with Replacing Eligibility Traces," *Proceedings of the First International Conference on Machine Learning and Cybernetics*, pp. 860-864, 2002.