Micro-controller based hardware-in-the-loop controller for electric drives

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

Tamil Kadir Rajavel

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

NED MOHAN

July, 2012

Acknowledgements

I would like to express my deepest gratitude to Prof. Ned Mohan for providing me the opportunity to work with him on this project and, offering support and encouragement throughout the course of the project. Working in his lab has been a wonderful experience.

I sincerely thank Prof. Tom Posbergh for his guidance and support.

Special thanks to Shanker Narayan for his advice, mentorship and the love and support he and his wife Preeti Subramanian have given me over the years.

I would also like to thank the members of Prof. Mohan's research group for their support.

Above all, I am grateful to my parents Rajavel Ganesan and Indira Rajavel and my grandparents for their infinite love, support and encouragement.

Dedication

To Nikola Tesla, the genius inventor of polyphase electric machines, which still powers our world and the foremost contributor to the technology of modern radio.

And to the millions of contributors to open source software and hardware, who make our world a better place.

Abstract

The growing focus toward renewable energy has made universities nation-wide to update their energy systems curriculum. A hardware lab for electric drives forms an essential part of such programs. To enable students to design control strategies, implement them in-class and test them on real machines, expensive DSP based hardware are currently being used. This thesis provides an alternative approach, exploiting the processing power of today's powerful computers. Using MATLAB/Simulink®'s Real-Time Windows Target, control systems can be run in realtime on regular windows machines. The controller developed as a part of this thesis provides the necessary hardware for utilizing this feature. The hardware-in-the-loop controller forms a communication link between the computer and the power converters driving the motor. The controller also reads analog sensor values and generates gate driving signals. The controller is microcontroller based and is relatively inexpensive. The controller has been designed and implemented in hardware, with a minimal firmware demonstrating the essential features of the set up. Satisfactory test results were obtained assuring the feasibility of this approach. Further work would help this technology mature into a replacement for industrial grade real-time controllers.

Contents

Acknowledgements i

Dedication ii

Abstract iii

List of Figures vi

1 Introduction 1

2 Hardware-in-the-loop simulation 3

3 Overview 5

4 Hardware design 7

4.1 Component Selection 8

4.2 Micro-controller PIC24E 9

4.2.1 Features of PIC24EP512GU810 9

4.2.2 Basic hardware configuration for the micro-controller 9

4.2.3 Analog-to-digital converter 11

4.2.4 Pulse-width-modulation signal generator 12

4.2.5 Serial Communication 13

4.2.6 General purpose input-output 13

4.3 Signal conditioning for analog-to-digital converter 14

4.4 Level-translation for pulse-width-modulated signals 15

4.5 USB communication 16

4.6 Power supplies . . . 18
4.6.1 5 volt supply . . . 18
4.6.2 3.3 volt supply . . . 19
4.6.3 ±12 volt supply . . . 19
4.6.4 Isolation of digital and analog power supplies . . . 20
4.6.5 Power supply selector and indicators . . . 21
4.7 General outline of board layout . . . 23

5 Software design **25**
5.1 Programming the PIC24E micro-controller . . . 25
5.1.1 MPLAB® IDE and C30 Complier . . . 25
5.1.2 Basic configurations . . . 26
5.1.3 Peripheral Pin Select . . . 29
5.1.4 ADC module . . . 30
5.1.5 PWM/Output Compare module . . . 31
5.1.6 Serial communication . . . 33
5.2 Programming the FT4232H USB controller . . . 35
5.3 Programming the computer . . . 37
5.3.1 Using FT4232H in UART mode . . . 37
5.3.2 Using FT4232H in MPSSE mode . . . 37
5.3.3 MATLAB/Simulink® programming . . . 38

6 Results **42**

7 Conclusion and Future Work **45**

References **47**

Appendix A. PIC24E Firmware **48**
A.1 Configuration File . . . 48
A.2 PWM Generation . . . 49
A.3 Reading ADC values using UART . . . 51
A.4 SPI Communication . . . 54

List of Figures

3.1 Overview of the system 5
4.1 Hardware-in-the-loop controller board outline 7
4.2 Basic hardware configuration for PIC24EP512GU810 10
4.3 Analog input pins on PIC24EP512GU810 12
4.4 Serial Communication on PIC24EP512GU810 13
4.5 General Purpose Input/Output on PIC24EP512GU810 14
4.6 Signal conditioning for analog-to-digital converter 15
4.7 Level-translation for pulse-width-modulated signals 16
4.8 FT4232H configuration for USB communication 17
4.9 5V supply from USB port 18
4.10 5V supply from external 9V adaptor 19
4.11 3.3V supply 19
4.12 ±12V supply design 20
4.13 Isolation of digital and analog power supplies 21
4.14 Power supply selector 22
4.15 Power supply indicators 22
4.16 PCB outline 23
5.1 MPLAB® IDE 26
5.2 PIC24E configuration bits 27
5.3 FTDI FTProg utility 36
5.4 MATLAB/Simulink® Real-Time Windows Target block set 38
5.5 Stream input block parameters 39
5.6 Board setup 40
5.7 Board verification 40

5.8 Sample control system 41
6.1 Assembled board 42
6.2 Sample ADC output in HyperTerminal 43
6.3 Sample PWM output 43

Chapter 1

Introduction

The first practical AC motor was patented by Nikola Tesla in the year 1888. Although there has not been any big change in the way electric machines are built since then, there have been lot of advancements in the way the machines are controlled. The (almost) exponential growth of the computing power available at our disposal, as predicted by Moore's law, has made the implementation of various complex machine control algorithms possible.

With the advent of smart grids, higher penetration of renewable energy sources and thus the increasing scope of research and jobs in the energy sector, universities across the country are beginning to focus more on their programs on energy systems. A hardware lab for electric drives is an integral part of such programs. With the emphasis still being on new control strategies, it calls for a system which can be used for rapid prototyping and testing of such control strategies.

The current setup being used at the electric drives lab at University of Minnesota is built around a DSP based hardware *dSPACE*[1]. It is a standalone real time control system, with an array of digital signal processors and an input/output module. The cost of such a system might be one of the barriers for many universities.

With general purpose PCs becoming more powerful with each passing day, their processing power could be exploited to design a real time control system, eliminating

the need for dedicated processors. This thesis explores this idea, using Simulink®'s Real-Time Windows Target to run Simulink® models in real time on any Microsoft Windows® based PC. A hardware, for taking care of the input/output functions and essential processing for motor control applications, has also been developed. The input/output module communicates with the PC via a USB port. This setup enables hardware-in-the-loop electrical machine simulations, which give the users deeper understanding of real machines.

Chapter 2

Hardware-in-the-loop simulation

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

In a hardware-in-the-loop simulation, hardware components are included into the system's control loop and the simulation is executed in real time. Such a system is closer to the real product, and thus provides a better understanding of the system.

Hardware-in-the-loop simulations have numerous benefits. The development cycle is reduced and the cost to innovate is lowered, as the traditional "simulate and prototype" loop is avoided. Instead, any modifications to the design can be quickly tested on hardware. This also improves the reliability and increases the efficiency of the testing system. It also helps in identifying design issues early in the development cycle.

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

level tests.

This thesis illustrates such a setup for hardware-in-the-loop simulations for power electronics and electric drives applications.

Chapter 3

Overview

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

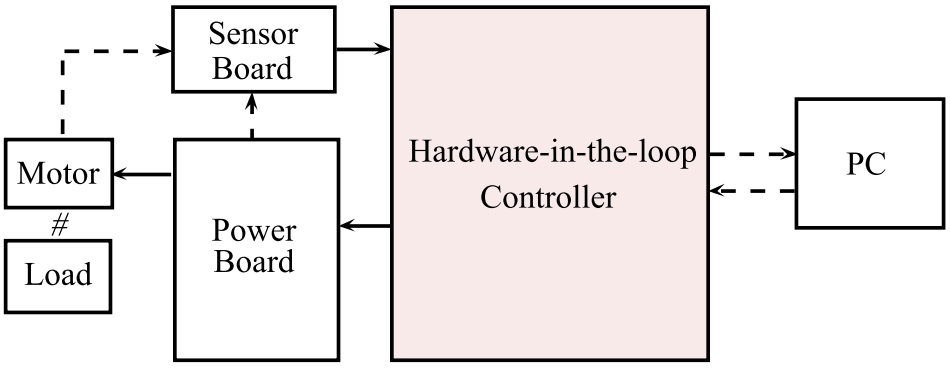

Figure 3.1: Overview of the system

The signals from the motor's encoder and the current and voltage sensors on the power board are sent to the hardware-in-the-loop controller, which has analog-to-digital converters to read the signal. The voltages read are converted into plain text and sent

to the computer via USB. The data is processed in real-time by the Simulink® model running on a Real-Time Windows Target. The model produces outputs, which are duty ratios for switching the gates on the power board. The output data is sent over USB as plain text to the hardware-in-the-loop controller, which then converts the data into numbers and produces appropriate PWM signals, that are fed to the gate drivers.

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

Chapter 4

Hardware design

The hardware-in-the-loop controller mentioned in the previous chapter was designed and developed as a part of this thesis. Figure 4.1 illustrates the outline of the controller board.

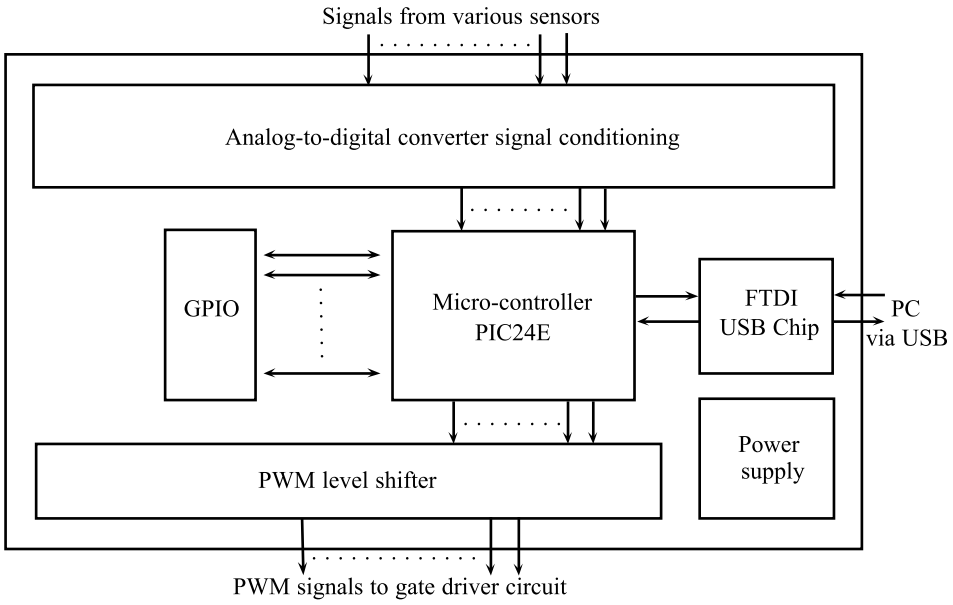

Figure 4.1: Hardware-in-the-loop controller board outline

The board was built around a Microchip's PIC24 series micro-controller. The peripherals used for this design were the analog-to-digital converter module and the pulse-width-modulation generation module. Communication between the micro-controller and the computer via USB is taken care of by a hi-speed USB to serial converter chip from FTDI. There is also additional circuity for conditioning the signals for ADC input to match input range of the micro-controller's peripheral. The PWM signals are level-shifted to match the voltage levels of the driver circuit on the power board. The board can be either supplied power from the USB port or can be powered from an external 9 volt supply. The power supply circuits are designed to supply power at various voltage levels required by the onboard components.

4.1 Component Selection

The main functions of the controller board are reading analog signals, generating PWM signals and communicating with the computer. Hence, the components were selected based those criteria. The analog to digital conversion capabilities required at least 10 bit conversion with the slowest conversion speed being 500 kilo samples per second. A minimum of eight channels were required. A minimum of 9 PWM outputs were required. The communication speed required was around 15 Mbps, which is higher than the USB 2.0 full-speed specification. Hence, a controller capable of handling USB 2.0 hi-speed communication was needed. Micro-controllers from various vendors including Microchip, Texas Instruments and Atmel were considered. Micro-controllers supporting USB 2.0 hi-speed communication were very limited, like Atmel's SAM3U. They did not have the required ADC or PWM peripherals. Hence, it was decided to use an external controller for USB communication. FTDI's FT4232H, a Hi-Speed Quad USB UART IC, was selected for this purpose. The chip has the entire USB stack implemented in hardware, and thus eliminates the need for additional programming on the micro-controller. Microchip's dsPIC line of controllers had specifications that exceeded the ADC and PWM requirements. PIC24F micro-controller, which is built on dsPIC controllers, was selected for this purpose, as it met all other requirements and had better features than a dsPIC. Also, ready availability of development resources for Microchip's micro-controllers influenced the selection.

4.2 Micro-controller PIC24E

Various salient features of the micro-controller chosen for this design, the PIC24EP512GU810, and the peripherals that are used are explained in detail in the following sections.

4.2.1 Features of PIC24EP512GU810

PIC24EP512GU810 has a 16 bit modified harvard architecture and can be configured to run at a maximum speed of 70 Million Instructions Per Second (MIPS). It has 32 analog channels, which can be configured to run at a maximum speed of 1.1 Mega samples per second (Msps) or a maximum resolution of 12-bits. The controller is equipped with 16 output compare/PWM modules and 16 bit timers. It also has standard digital communication peripherals like UART (Universal asynchronous receiver/transmitter), SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit).

One of the very special features of the PIC24E controller is the Direct Memory Access (DMA), which allows direct data transfer between the CPU and its peripherals such as UART, SPI, ADC, Output Compare, Timers, etc. without CPU assistance. PIC24EP512GU810 has 15 DMA channels. DMA is very useful in systems where reducing latency is a top priority.

PIC24E micro-controllers have a flexible feature called Peripheral Pin Select (PPS), which enables the placement of peripheral pins on a wide range of I/O pins. Any digital peripheral can be mapped over a fixed set of digital I/O pins. The mapping can be done in the micro-controller code. This feature gives more flexibility while laying out the board and also gives room for architecture modifications in the future.

4.2.2 Basic hardware configuration for the micro-controller

The schematic shown in Figure 4.2 represents the basic hardware configuration for the PIC24E micro-controller. Each essential component is discussed in detail below:

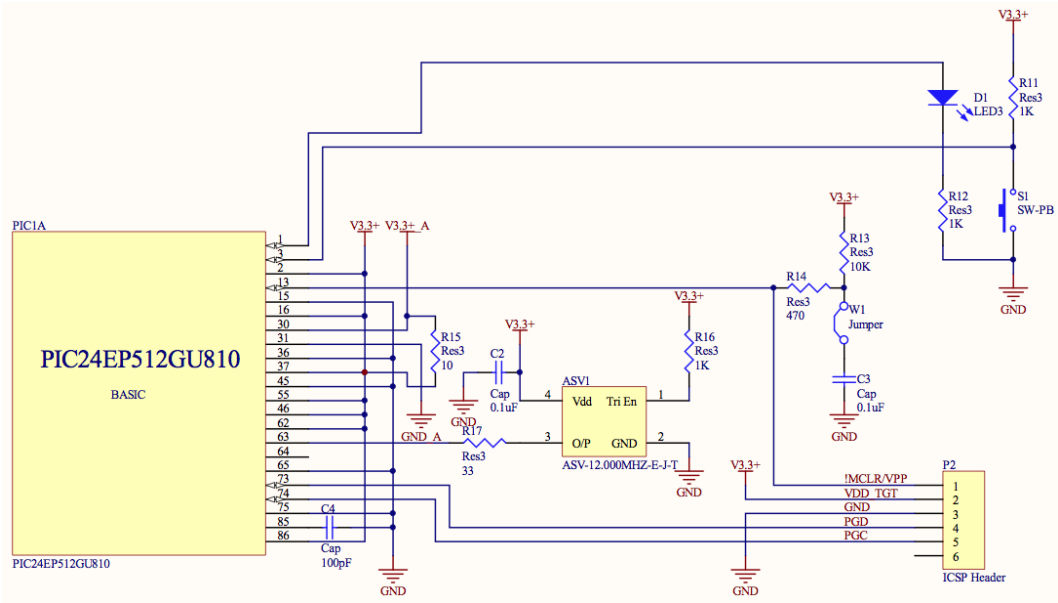

Figure 4.2: Basic hardware configuration for PIC24EP512GU810

Power supplies, decoupling capacitors and filtering capacitors

The operating voltage of the PIC24EP512GU810 is between 3.0V and 3.6V. The on-board power supply produces 3.3V, which is used by the micro-controller. Decoupling capacitors are required for every power supply pin pair on the micro-controller. As recommended by Microchip, a 100nF, 25V ceramic capacitor is used between V_{DD}, V_{SS}, V_{USB3V3}, AV_{DD} and AV_{SS} pairs. A filtering capacitor is connected to the V_{CAP} pin, to stabilize the voltage regulator output voltage.

The decoupling capacitors must be placed as close to the micro-controller pins as possible, not exceeding a distance of 6mm. They must also be as close as possible to the power supply. In this particular board, the capacitors are generally placed in the bottom layer and connected to the micro-controller pins through a via. The power is supplied to the capacitors directly from the internal power planes. Care has been taken to ensure that the capacitors appear in the power chain before the micro-controller pins.

Master clear ($\overline{\text{MCLR}}$) pin

The master clear ($\overline{\text{MCLR}}$) pin is used to reset the device and also during device programming and debugging. During device programming and debugging, the ($\overline{\text{MCLR}}$) pin is driven by the programmer or debugger. A pull-up resistor of $10k\Omega$ is connected to meet the V_{IH} and V_{IL} specifications. A current limiting resistor of 470Ω is connected in series, to limit the current flowing in to and out of the capacitor in the event of pin breakdown. A capacitor value of $0.1\mu F$ is chosen so that the fast signal transitions during are not affected.

External oscillator pins

The system clock can be sourced either from Primary Oscillator, Secondary Oscillator, Internal Fast RC Oscillator or Internal Low-Power RC Oscillator. In this design, external clock has been used as the primary oscillator. A $3.3V_{dc}$ HCMOS/SMD crystal clock oscillator from Abracon Corporation is used to generate 12.000Mhz $\pm$20ppm as a primary clock source. A on-chip Phase-Locked Loop (PLL) can be used to boost the operating frequency further. PLL configuration is discussed in the software design chapter.

ICSP Pins

PGEC2 and PGED2 pins are used for In-circuit serial programming (ICSP) and debugging. The tool used for this purpose is Microchip® PICkit 3^{TM}. Appropriate pins are brought out at the ICSP header, to match the PICkit 3 pin-outs. Pull-up / pull-down resistors, series diodes and capacitors are avoided on the programming pins, since they could affect the communication between the micro-controller and the programmer/debugger.

4.2.3 Analog-to-digital converter

The analog-to-digital converter in the PIC24EP512GU810 is a successive approximation(SAR) converter. The micro-controller has 32 analog input pins. The converter can be programmed to use external voltage references. The converter can operate either in 10-bit mode or 12-bit mode. In the 10-bit mode, conversion speeds up to 1.1Msps are

achievable, with simultaneous sampling of up to four inputs. In the 12-bit mode, conversion speeds only up to 500ksps are supported and only one sample and hold amplifier is available. Further details on configuring the ADC module are discussed in the next chapter.

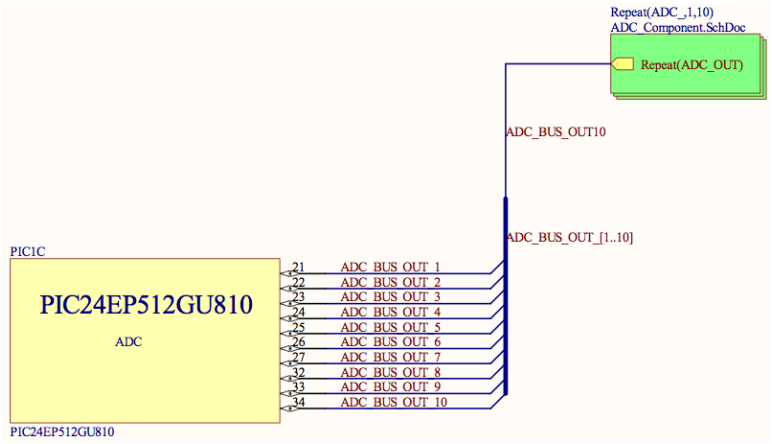

Figure 4.3: Analog input pins on PIC24EP512GU810

The schematic shown above illustrates the analog input pins connections. The signals from the sensors are conditioned in signal conditioning circuits, whose output drives the analog input pins of the micro-controller.

4.2.4 Pulse-width-modulation signal generator

The PWM signals can be generated by using the output compare module. The output compare module compares the value of its timer with the value stored in the compare register. When the values match, the state of the output pin is changed as programmed. The timer can be driven from any of the eight available clock sources. The module is capable of operating in edge-aligned or center-aligned PWM modes. The module also has fault control features with three fault input pins, which are currently not implemented, but can be extended as required using the GPIO pins. The generated PWM signals are processed by a level shifter and buffer before being supplied to the gate drivers on the power board.

4.2.5 Serial Communication

PIC24EP512GU810 offers various digital serial communication peripherals such as UART, SPI and I2C. The micro-controller communicates with the computer through the FTDI USB chip, which is connected to the micro-controller via a serial interface. The FTDI chip also supports various communication standards. The board is designed to communicate with the FTDI chip using two SPI channels, being connected to two ports on the FTDI chip.

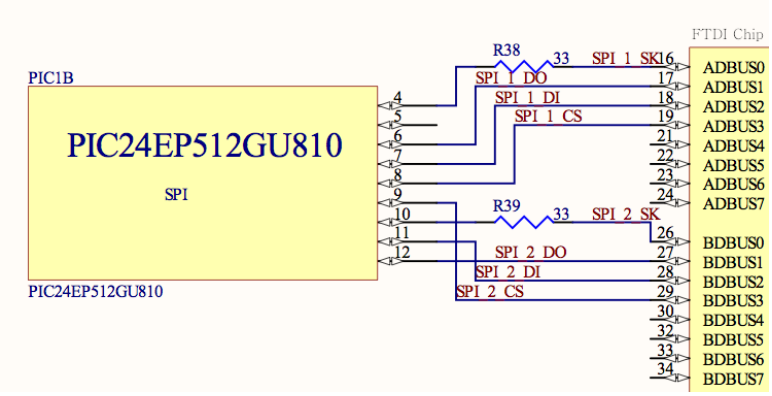

Figure 4.4: Serial Communication on PIC24EP512GU810

By taking advantage of the peripheral pin select feature in PIC24E, the same hardware connections can be used for other serial communication standards as well, if needed.

4.2.6 General purpose input-output

The PIC24EP512GU810 has seven parallel bi-directional 16 bit ports - Port A till Port G. They generally share the pin with various onboard peripherals. Nine of such general purpose input/output pins have been brought out to a screw-less terminal block, for debugging purposes or for future use.

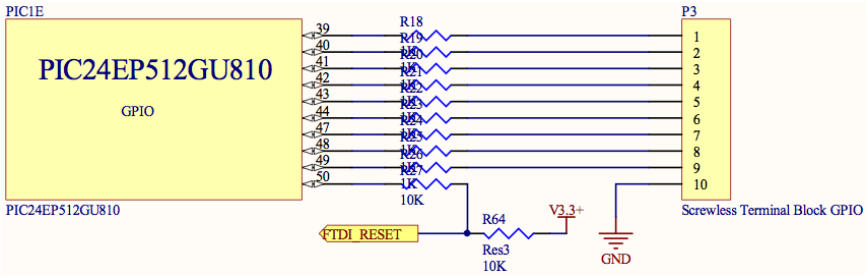

Figure 4.5: General Purpose Input/Output on PIC24EP512GU810

As seen in the schematic above, one I/O pin is connected to the device reset pin of the FTDI chip. This will be discussed in detail in a later section.

4.3 Signal conditioning for analog-to-digital converter

The voltage levels from the sensors on the the power board are bipolar in nature (±10V), but the ADC module in the micro-controller can accept only unipolar voltages, within the voltage reference levels of 0V and 3.3V. Hence, the signals from the sensors have to be conditioned in a way that is acceptable by the analog-to-digital converter module on the PIC24EP512GU810.

As shown in the schematic in figure 4.6, a modular approach was used for designing the conditioner circuit[6]. A 50Ω resistor, R5 was added for impedance matching of the BNC probes, for maximum power transfer. The circuit essentially has two stages, the first one being for attenuation and the second one being for level shifting. The first stage is handled by Texas Instrument's OPA277. This op-amp was chosen for its low drift and low bipolar swing. The value of resistor R8 dictates the input range. The voltages in the voltage divider consisting of R8 and R3 can be given as:

$$\frac{R8}{R3} = \frac{V_{OUT}}{V_{IN} - V_{OUT}}$$

In our case, V_{IN} =20V (±10V) and V_{OUT}=3.3V. Choosing R3=10K, we get R8=1.97K. The nearest standard value of 1.98K is chosen.

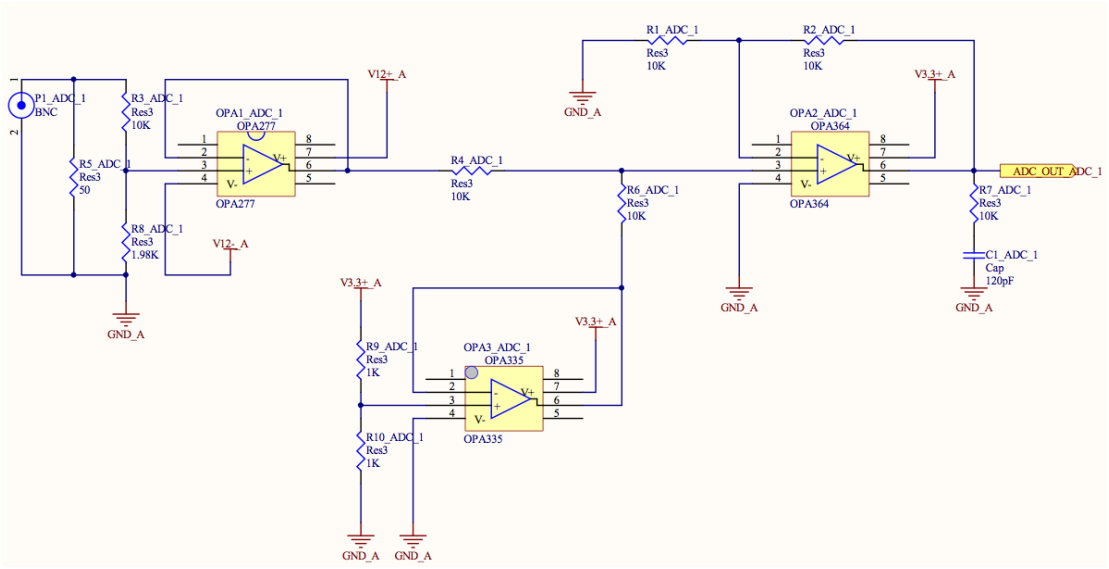

Figure 4.6: Signal conditioning for analog-to-digital converter

The second stage is formed around OPA364. This op-amp has ideal characteristics for this stage - it has large input common mode voltage range and also has zero cross over distortion for linear, monotonic, large-signal output. The function of this stage is to level shift the output of the previous stage to the desired range. The reference voltage of 3.3V/2 = 1.65V is generated by OPA335. Thus, the output of the signal conditioning circuit is 0-3.3V for a input range of ±10V. This output is then fed to the analog inputs of the micro-controller.

4.4 Level-translation for pulse-width-modulated signals

The pulse-width-modulation signals generated by the PIC24EP512GU810 are in the 3.3V logic level, and have a current driving capability of 8mA. The drivers on the power board operate at 5V logic. TXB0104, a 4-bit voltage level translator from Texas Instruments is used for this purpose. The chip is bi-directional and can sense the direction automatically. Its input side is driven by 3.3V logic signals from the micro-controller. The level translated 5V logic signals from it output are then fed to the gate drivers on the power board.

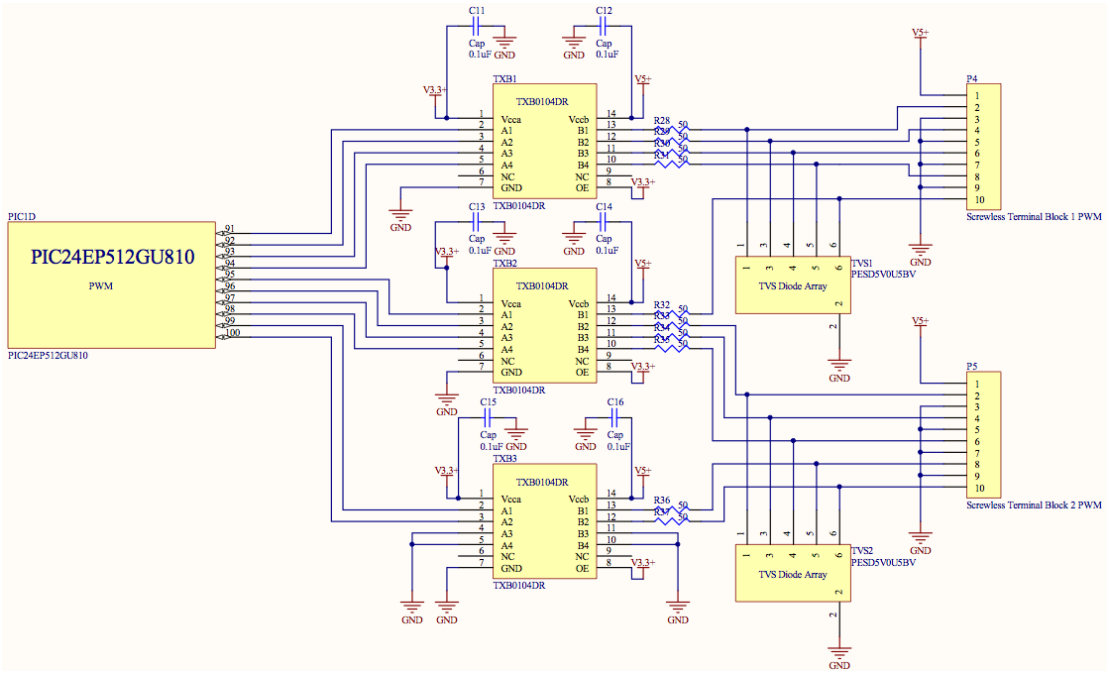

Figure 4.7: Level-translation for pulse-width-modulated signals

The TXB0104 can supply a continuous current of up to ±50mA. The input transition rate is 40ns/V and the output transition rate is 30ns/V, which are well within the range rise/fall times of the PWM signals that are needed for the gate drivers. The chip has onboard ESD protection of for voltages up to ±15kV. Also, for extra protection, transient voltage suppressor diodes are added to the output stage.

4.5 USB communication

The hardware-in-the-loop controller communicates with the PC through Universal Serial Bus (USB). The USB communication is handled by a dedicated IC from FTDI - the FT4232H. The device supports hi-speed USB, which is a maximum of 480 Mbits/s (60MB/s). The entire USB protocol is handled by the chip in hardware, which reduces the amount of code that needs to be written for establishing the communication channel.

FT4232H has four channels, which can be used as UART or Bit-Bang interfaces. Two of those channels can be used in Multi-Protocol Synchronous Serial Engine (MPSSE)

mode, which supports serial communication protocols like SPI, I2C, JTAG, etc.. As seen in figure 4.8, Port A and Port B of FT4232H are connected to the re-programmable peripheral pins of the micro-controller, for using FT4232H in the MPSSE mode or UART mode as required.

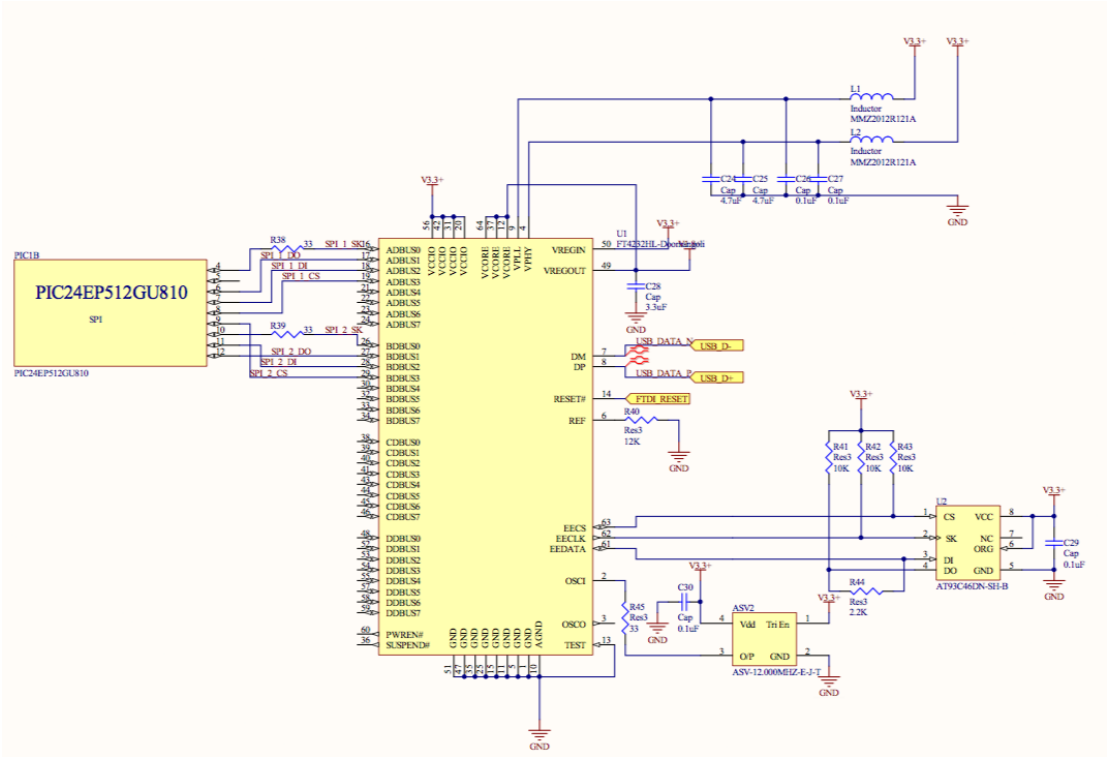

Figure 4.8: FT4232H configuration for USB communication

The FT4232H's clock is driven by an external 12Mhz clock, similar to the one used for the micro-controller as described earlier. The core of the chip, which requires 1.8V, is supplied by the on-chip voltage regulator. Inductive and capacitive filters are used to eliminate disturbances in the VPLL and VPHY supplies. An external EEPROM, AT93C46, is connected to the chip for storing device configuration data and other optional manufacturer data.

4.6 Power supplies

The hardware-in-the-loop-controller board has been designed using components that operate at 3.3V logic level. However, the PWM outputs need to be at 5V logic and the first stage of the analog signal conditioning circuit require ±12V supply. The input power is obtained either from the USB port or from external 9V supply. This makes it necessary to have voltage converters for the different voltage levels required by the components of the board.

4.6.1 5 volt supply

The 5V supply can either be sourced from the USB port or from the external 9V supply adapter.

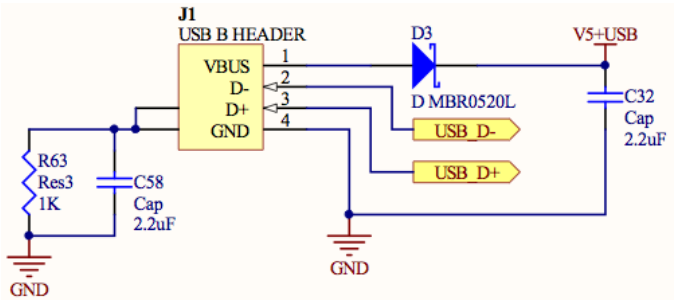

Figure 4.9: 5V supply from USB port

Figure 4.9 shows the connections for the USB port. Apart from the data lines D+ and D-, there are two power pins - V_{BUS} and GND, which are used for this supply. The USB bus voltage is 5V. Typically, USB hosts allow a maximum current of 500mA to be drained. However, special devices like battery chargers may allow currents up-to 1.5A to be drained from their USB ports. A schottky rectifier is added in the power line for protection.

The primary power source for the controller board is from an external 9V supply. A linear regulator, LM1117 is used to step-down the voltage from 9V to 5V.

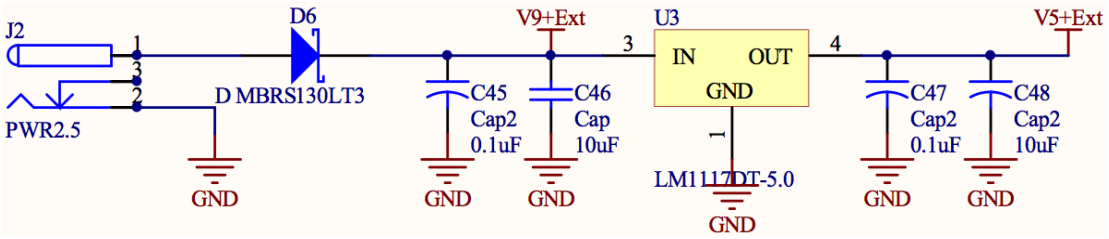

Figure 4.10: 5V supply from external 9V adaptor

As seen in figure 4.10, filtering capacitors accompany the regulator IC and also a schottky rectifier is used for protection. The power jack is a regular 2.5mm jack.

The board can be powered by USB only when the ADC modules and PWM modules are not in use, e.g., while programming, debugging the micro-controller and other digital components etc.. When the ADC modules and PWM modules are used, the current consumption is higher and the board must be powered only by the external 9V supply.

4.6.2 3.3 volt supply

The PIC24E micro-controller, the FTDI USB controller and the second stage of the analog signal conditioning circuit operates at 3.3V voltage level.

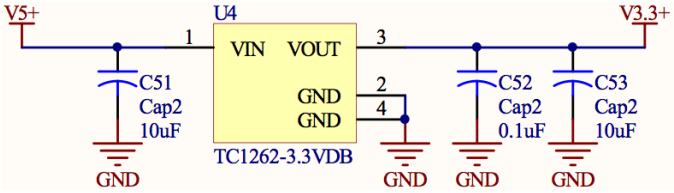

Figure 4.11: 3.3V supply

The schematic in figure 4.11 shows the configuration of TC1262, a fixed output CMOS low-dropout regulator from Microchip. Filter capacitors are used as needed.

4.6.3 ±12 volt supply

The operational amplifier OPA277, used in the first stage of analog signal conditioning circuit, requires a bipolar 12V supply.

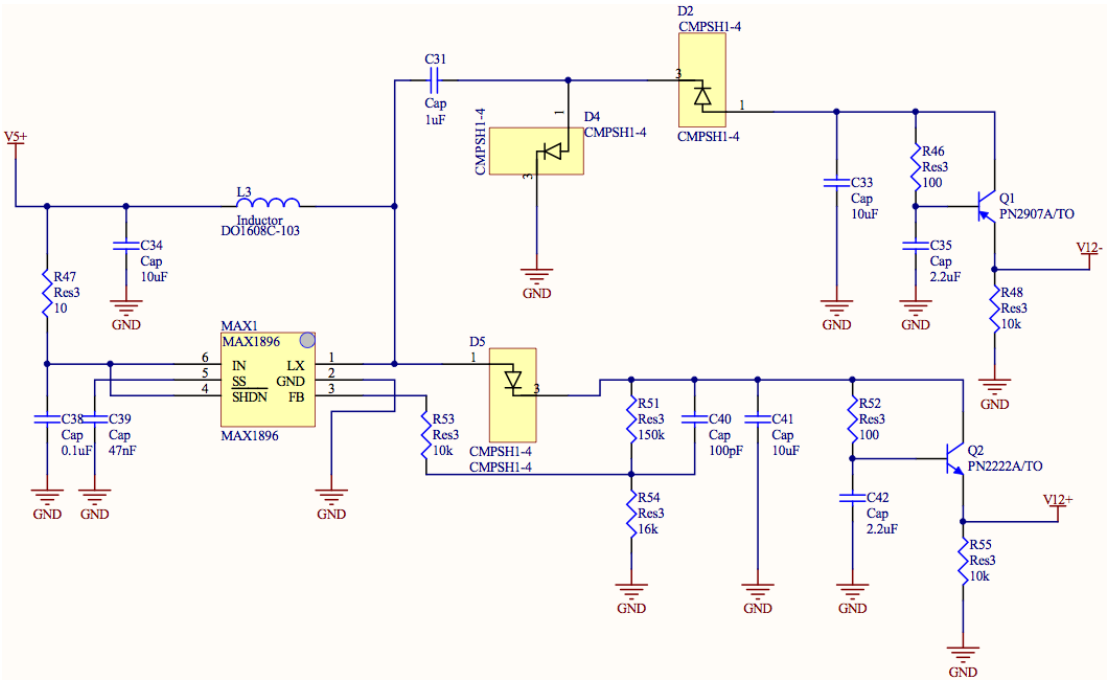

Figure 4.12: ±12V supply design

The bipolar 12V supply is built around Maxim's MAX1896. It is a constant frequency, current mode step-up converter. MAX1896 steps up the onboard 5V to +12V. The -12V supply is obtained from an external charge pump, formed by D2, D4, C31 and C33[7]. Since the converter operates at constant high frequency, it is easy to filter out the noise. It is done by transistor buffered RC filters as shown in Figure 4.12. MAX1896 also has a current limiting feature, so that the PC's USB port is not accidentally damaged.

4.6.4 Isolation of digital and analog power supplies

Digital components of the board include the micro-controller, the FTDI chip and the PWM buffer circuits. They operate at high speeds and that might induce noises in the analog parts of the board, which are the analog signal conditioning circuits.

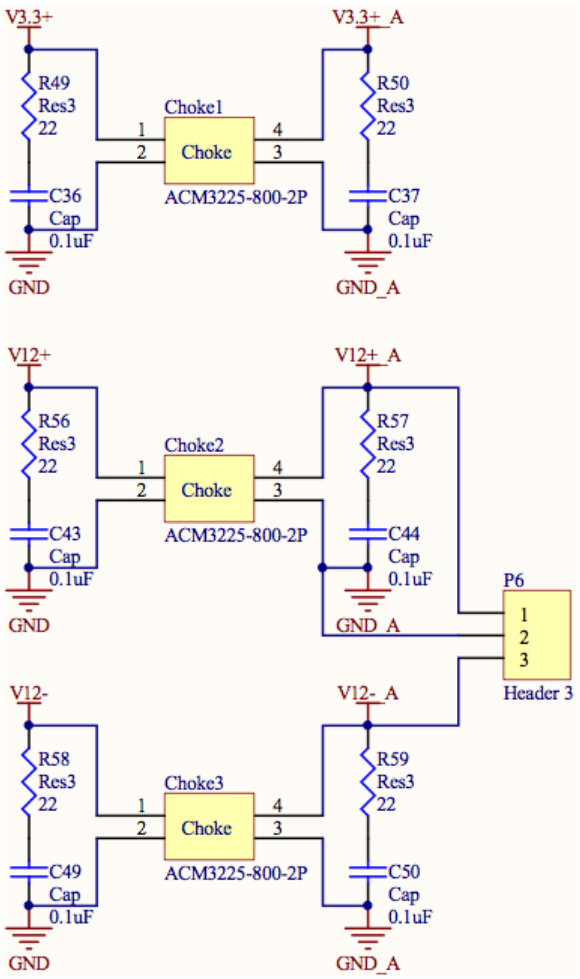

Figure 4.13: Isolation of digital and analog power supplies

To avoid digital noise affecting the analog signals, the digital and analog power supplies are isolated using a common mode filter choke. The chokes also filter the noise from the external power supply adapter.

4.6.5 Power supply selector and indicators

As mentioned earlier, the board can either be powered directly from the USB port or from an external 9V supply. As shown in figure 4.14, there is a selector switch for selecting the power source for the board. The switch can be kept in the middle position for powering the board down.

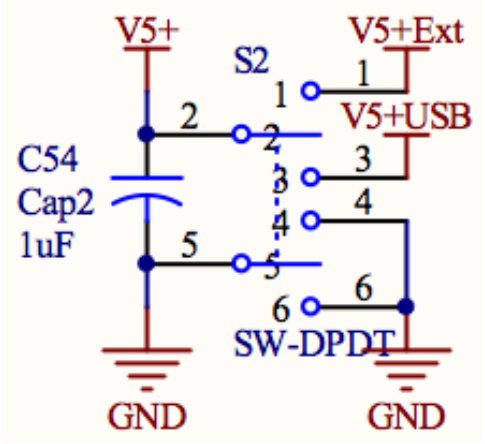

Figure 4.14: Power supply selector

There are indicator LEDs for indicating which supply is powered up. There is an indicator for the 3.3V supply, the 5V USB supply and the 5V derived from external 9V supply.

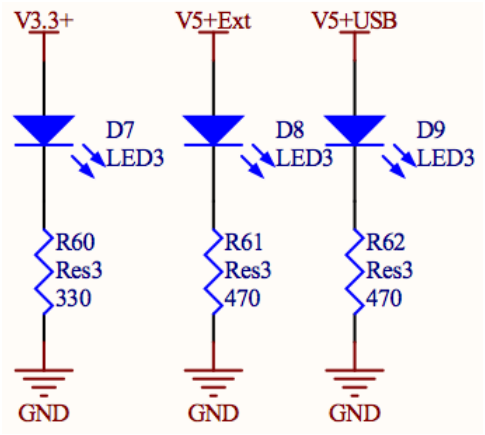

Figure 4.15: Power supply indicators

There are also test points for checking the status of the ±12V supply.

4.7 General outline of board layout

The dimensions of the board are 7.695 × 4.6038 inches. It is a four layer board, with top and bottom copper layers and two internal power planes.

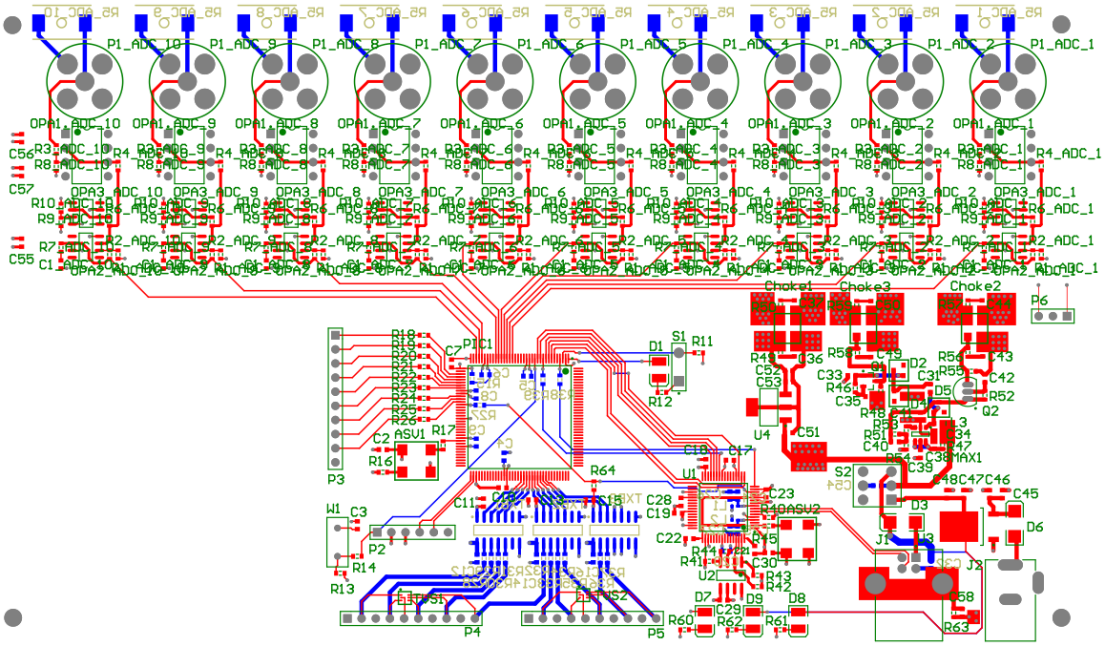

Figure 4.16: PCB outline

As seen in figure 4.16, the BNC sockets for analog inputs and the accompanying signal conditioning circuits occupy the top part of the board. The 50Ω impedance matching resistors are placed as close to the BNC sockets as possible.

Power supplies, power socket, power supply indicators and USB socket are placed in the bottom right of the board. Traces wider than 40mils are used for the power chain. The board has two internal power layers - one for ground and one for power. The ground and power lines are connected to the internal layers through a group of stitching vias. There are no dedicated power traces running to the components, since the power is taken directly from the power planes using appropriate vias.

The PIC24E micro-controller and the FTDI chip are placed in the bottom middle of the board. De-coupling capacitors are connected to the power pins in both the chips. The capacitors are generally placed on the bottom layer, as close as possible to the power pins and appear in the power chain before the power pins of the chips. The external clocks for both the chips are placed as close as possible to the chips and it has been made sure than no signal traces pass under them in any layer.

Since the outputs of the PWM signal buffer would need to handle higher power, wider traces are used for them. The outputs are then drawn out from the headers using a screw-less terminal block.

The USB data signals between the USB port and the FTDI chip and the serial data signals between the FTDI chip and the PIC24E micro-controller are treated as differential pairs, hence are kept close to each other and the traces are almost of the same length.

In general, the board has been designed based on placement and layout recommendations found in the data-sheets and application notes from the manufacturers. The goal has been to minimize noise issues, with an emphasis on signal integrity and protection.

Chapter 5

Software design

This chapter deals with the design of the software which is required to make use of the hardware discussed in the previous chapter. Programming needs to be done on the PIC24E micro-controller for taking care of the analog to digital conversion, PWM signal generation and communication with the FTDI chip. The FT4232H USB controller must be set up in such a way to form a USB based data bridge between the micro-controller and computer. Also, necessary programming has to be done on the computer for communicating with the FT4232H USB controller and running the Simulink® models in real-time.

5.1 Programming the PIC24E micro-controller

Microchip offers a full set of tools for programming the PIC24E micro-controller. They provide a powerful development environment and a full-featured ANSI C compliant complier free of charge. Microchip also sells the PICkit 3 In-Circuit Debugger, which can be used for programming and debugging the micro-controller.

5.1.1 MPLAB® IDE and C30 Complier

MPLAB® X is an integrated development environment for programming and debugging PIC micro-controllers. It is a set of tools that include an editor for editing the programs, a compiler for compiling higher language programs to machine code, and a support for various programming and debugging tools such as PICKit 3. MPLAB also has a

simulator, which can be used to simulate the code before flashing the micro-controller.

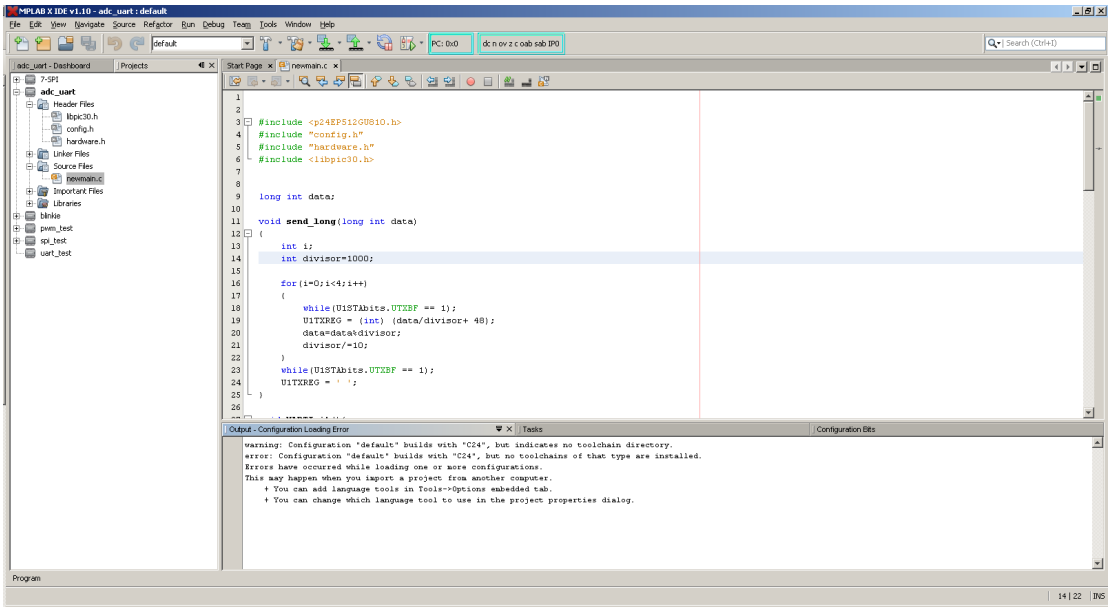

Figure 5.1: MPLAB® IDE

As seen in the figure above, the IDE, built on NetBeans, has standard features such as live code check and autocompletion. It also has a familiar interface.

Microchip offers a free C complier for it's line of 16 bit devices, which include PIC24, dsPIC33F, etc.. The C30 compiler is a full-featured ANSI C complaint compiler, with standard, fixed & floating point math, memory and data conversion libraries. The compiler also supports in-line assembly usage. To use the C30 compiler, the appropriate options must be selected in the tool chain section of MPLAB®'s configuration.

5.1.2 Basic configurations

The basic configurations for operating a PIC24 micro-controller include setting up the appropriate configuration bits, selecting the proper clock source and setting up the I/O ports for proper operation. They are discussed in the following sub-sections.

Configuration Bits

Configuration bits are non-volatile bits that need to be set to the appropriate values for proper functioning of the micro-controller. The bits control features like clock selection, code protection, watchdog timer configuration, etc., which are all mentioned in detail in the device's data-sheet[3].

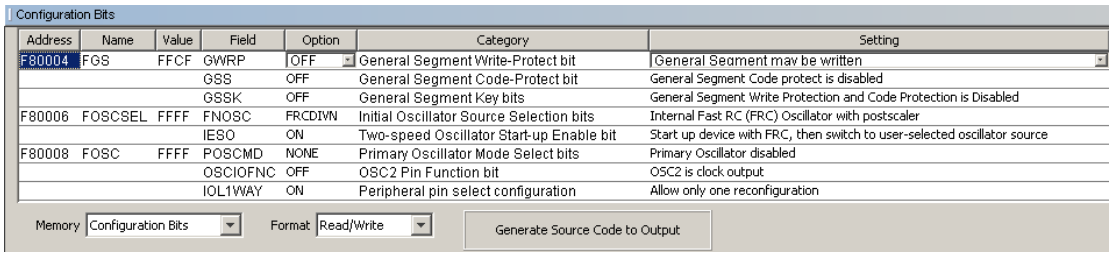

Figure 5.2: PIC24E configuration bits

MPLAB® offers a graphical utility for selecting configuration bits. It can be accessed from the "Window" menu. After selecting the appropriate configuration bits, a code containing the information can be generated from the utility. The code must then be included in the source code. It is a good practice to have the configuration data in a separate file, which can then be included in the main source code file.

Oscillator Configuration

PIC24E offers four external and internal oscillator options, which can be selected using appropriate values of the configuration bits. The available oscillator options are internal fast RC oscillator with or without PLL, primary oscillator with or without PLL, secondary oscillator, low power RC oscillator and fast RC oscillator with post-scaler. The primary oscillator can either XT - crystal oscillator, HS - high speed oscillator or EC - external clock. The oscillator used in this board is an external oscillator. It can be used with Phase-Locked Loop (PLL) to operate the device at higher speeds. Device operating frequency F_{CY} in such a case can be calculated as follows:

$$F_{OSC} = F_{IN} \times \frac{(PLLDIV + 2)}{(PLLPRE + 2) \times 2(PLLPOST + 1)}$$

$$F_{CY} = \frac{F_{OSC}}{2}$$

The VCO frequency, F_{VCO} is given by

$$F_{VCO} = F_{IN} \times \frac{(PLLDIV + 2)}{(PLLPRE + 2)}$$

For example, to operate the micro-controller at the speed of 60MIPS(Million Instructions Per Second) using the onboard 12Mhz external clock, we can use the following settings: $PLLPRE = 0$, $PLLDIV = 38$ and $PLLPOST = 0$. Thus, we get $F_{CY} = 60MIPS$. And, $F_{VCO} = 240Mhz$, which is within the acceptable range of $120Mhz < F_{VCO} < 340Mhz$.

These values can be written to the $CLKDIV$:Clock divisor and $PLLFBD$:PLL feedback divisor registers. The oscillator can be controlled on the fly using the $OSCCON$:Oscillator control register. The register also has read-only status bits to read the clock source selection status, PLL lock status, etc..

I/O Pin configuration

PIC24E micro-controller has parallel input/output ports shared with all pins expect power, master clear and oscillator pins. All ports are configured as inputs upon reset. The data direction of a port is set in the $TRISx$ register (where x is the port name). The individual bits of the register can also be accessed directly. Setting a $TRISx$ bit '1' configures the corresponding pin as an input, and setting the bit value '0' configures the corresponding pin as an output. To read the value at a port, its corresponding $PORTx$ register is read. To write a value to an output port, the value is written to the corresponding $LATx$ register. The ports can also be configured in open-drain output, which is controller by the $ODCx$ register.

For ports which are capable of handling analog signals, the $ANSELx$ register is used to contol its operation. Clearing the register configures the pin as digital and setting the register configures the pin as analog. The pins are all analog by default, since the

default value of $ANSELx$ is $0xFFFF$. Also, for analog operation, the corresponding $TRISx$ bit of the pin must be set.

5.1.3 Peripheral Pin Select

Peripheral PIn Select(PPS) is one of the designer-friendly features on the PIC24E. In a high pin-count device with many peripherals, such as the PIC24, it becomes inconvenient and difficult to design the hardware when the peripherals are fixed to certain pins, which may not be ideally placed. This required the application to be modified around the pin placement. The PPS features allows the users to map the input/output of most digital peripherals to any of the PPS enabled pins. This offers greater flexibility in design, and now the pin placement can be modified to suit the application. The PPS configuration is entirely done in software, and the pins can be reconfigured on the fly if needed.

The PPS enabled pins are either labelled as RPn or $RPIn$, where n is the port number. A RP pin can either be remapped as an input or an output, whereas a RPI can only be remapped as an input. The available peripherals are digital peripherals such as SPI, UART, external interrupt, external timer, input capture, etc..

THe PPS feature of a pin is controlled through two special function registers - one for mapping input, and one for mapping outputs. Inputs are mapped on basis of the peripheral. The appropriate values for input mapping can be found in table 11-1 and table 11-2 in the PIC24E data-sheet[3]. For example, to assign SPI clock signal (SCK1) to the pin RPI86 as an input, the following code is used:

$$RPINR20bits.SCK1R = 0x56;$$

Outputs are mapped on the basis of the pins. Table 11-3 in the PIC24E data-sheet[3] lists the values of output selection for re-mappable pins. For example, to configure RP87 as SPI serial data out (SDO), the following code is used:

$$RPOR6bits.RP87R = 5;$$

Since there are no enforced lock-outs between any of the PPS registers, it is possible to have any combination, including one-to-many and many-to-one. Though it is possible

to do so in code, it might not be the best thing to do, from an electrical stand point. Hence, care must to taken to use the PPS feature sensibly.

5.1.4 ADC module

The PIC24E has a successive approximation analog to digital converter. As described in the previous chapter, it has up to 32 analog input pins: AN0-AN31, which are connected to four sample & hold amplifiers: CHO-CH3 through multiplexers. The ADC module also supports direct memory access (DMA), which helps in reducing load on the CPU and increasing the performance. The configuration and operation of the ADC module can be summarized by discussing the ten control and status registers associated with the module, as under:

ADxCON1: ADCx Control Register 1

The ADCx Control Register 1 controls turning of the ADC module on or off, selecting the 10 bit operation mode or the 12 bit operation mode, selecting the data output format, ADC clock source selection, whether the conversion is done automatically or manually and enabling of the ADC module. It also has a status bit which is set when the ADC conversion cycle is complete.

ADxCON2: ADCx Control Register 2

The ADCx Control Register 2 has options for selecting the high and low voltage references and selecting the channels to be scanned. It also has an option to select the buffer fill mode: either to be always filled form the starting address or if the buffer must be filled in halves based on the interrupts received. If the later option is selected, the status of the buffer can be observed in the buffer fill status bit.

ADxCON3: ADCx Control Register 3

ADCx Control Register 3 is mainly for ADC conversion clock selection: which can be 1-256 times the CPU cycle time period and clock source selection: which can either be derived from system clock or can be the internal RC clock.

ADxCON4: ADCx Control Register 4

ADCx Control Register 4 is for configuring the direct memory access (DMA). It has a bit for enabling or disabling the use of DMA. If DMA is used, the register also has options for allocating $2^0 - 2^7$ word(s) of buffer for each analog input in the DMA channel.

ADxCHS123: ADCx Input Channel 1, 2, 3 Select Register

The ADCx Input Channel 1, 2, 3 Select Register is used to select negative input and positive input for sample A and sample B.

ADxCHS0: ADCx Input Channel 0 Select Register

ADCx Input Channel 0 Select Register is the same as ADCx Input Channel 1, 2, 3 Select Register, but is used to configure channel 0.

ADxCSSH: ADCx Input Scan Select Register

ADCx Input Scan Select Register is made up of two 16 bit registers, with each bit being used for selecting a particular analog input pin for scanning.

ANSELy: Analog/Digital Pin Selection Register

ANSELy register is used to configure a pin as analog or digital. It is used along with the TRIS registers for digital applications.

Detailed and complete bit information for each register can be found in the device data-sheet[3]. The 'x' in the register names refers to ADC1 or ADC2.

5.1.5 PWM/Output Compare module

Output compare module in the PIC24E can be used to generate PWM signals. The module compares the values of the output compare timer with the value in one or two compare registers. If the values match, the state of the output pin changes. The module can either generate a a single output or an output sequence. Each output compare channel is implemented by the following five registers:

OCxCON1: Output Compare x Control Register 1

The Output Compare x Control Register 1 is used to select one of the available eight clock sources, enable or disable fault inputs, select trigger status mode and the mode of operation of the output compare module. It also has status bits for indicating the occurrence of faults in PWM.

OCxCON2: Output Compare x Control Register 2

Output Compare x Control Register 2 register is used to select various option related to faults, such as fault mode, fault out bit and fault output state. The register also has a bit to enable or disable cascaded timer operation. Timer trigger status and synchronization source selection bits are part of this register. The register also has an option to tri-state the OCx pin.

OCxR: Compare Register and OCxRS: Secondary Compare Register

Compare register and the secondary compare register store the value that needs to be compared with the timer, which needs to match to change output state.

OCxTMR: Internal Time Base Register

The Internal Time Base Register stores the current value of the timer. The value of this register is compared with the value stored in the Compare register to produce output.

The output compare module can be operated in four different modes, but for this particular application, it needs to be operated in the simple Pulse-Width Modulation (PWM) mode. Setting the value of $OCM < 2:0 >$ in $OCxCON1$ to be 110 or 111 for operating the module in Edge-Aligned PWM or Center-Aligned PWM respectively. In Center-Aligned PWM mode the output set high when the timer matches the value in the compare register and is set low when the timer matches the value in the secondary compare register. In Edge-Aligned PWM mode, output set high when the timer starts and is set low when the timer matches the value in the compare register.

For example, to operate the module in Edge-Aligned PWM mode, first the clock source is selected. Then, T_{CY} is determined and based on that the pulse ON value is calculated and written to $OCxR$. Then, the period of the PWM is calculated based on T_{CY} and written to $OC1RS$. Finally, $OCM < 2:0 >$ in $OCxCON1$ is set to 110 to select the mode of operation.

5.1.6 Serial communication

Serial communication offers link between the PIC24E and FT4232H. PIC24E supports various digital serial communication protocols such as SPI, I2C, UART and ECAN. UART and SPI protocols have been explored in this project. Since PIC24E has the Peripheral Pin Select (PPS) feature, there is no hardware change that is needed to switch between the two protocols.

UART

Universal Asynchronous Receiver Transmitter (UART) is one of the simplest serial communication protocols. It is a full-duplex interface, which can be used to implement RS-232, RS-485, etc.. UART is essentially operated using two registers: $UxMODE$ and $UxSTA$.

$UxMODE$, the UARTx mode register, has bits for enabling and disabling the module, specify which pins in the module are enabled and a bit for enabling or disabling auto-baud select. It is used to specify essential parameters like parity & data bit selection and number of stop bits. It also has a bit for choosing between the standard 16x clock of the high-speed 4x clock.

$UxSTA$, the UARTx status and control register is used to check the status of the transmission buffer being empty and the receive buffer having data available. It contains the bit for enabling transmission. It also has various fault status bits and interrupt configuration bits.

The data to be transmitted is written to $UxTXREG$: UARTx Transmit Register. The received data can be read from $UxRXREG$: UARTx Receive Register. The baud

rate is generated based on the value stored in the $UxBRG$: UARTx Baud Rate Register. The value of UxBRG is calculated using the following formula:

$$UxBRG = \frac{F_{CY}}{16 \times Baudrate} - 1; \text{ for 16x clock}$$

$$UxBRG = \frac{F_{CY}}{4 \times Baudrate} - 1; \text{ for 4x clock}$$

To set up the UART module for communication, first the number of data bit, number of stop bit and parity must be specified in $UxMODE$. Then, the value of the baud rate register must be calculated and be written to $UxBRG$. Then, the module needs to be enable by setting $UARTEN$, after which the transmission can be enabled by setting $UTXEN$. Loading the data into $UxTXREG$ starts the transmission. To receive data, $URXDA$ must be scanned to check if it is set, which means that data has been received. The received data can be read from $UxRXREG$. Interrupts can also be used for receiving data.

SPI

Serial Peripheral Interface (SPI) is a synchronous serial interface. Each SPI module consists of four pins: SDIx: Serial Data Input, SDOx: Serial Data Output, SCKx: Clock Input or Output and $\overline{\text{SSx}}$: Slave Select.

The SPI module has four Control and Status registers. $SPIxSTAT$: SPIx Status and Control Register indicates the status of receive and transmit buffers and contains a bit that enables or disables the module. $SPIxCON1$: SPIx Control Register1 which has a bit to select mast/slave mode, and various options for configuring the clock. $SPIxCON2$: SPIx Control Register2 has options related to framed SPI operation. $SPIxBUF$: SPIx Data Receive / Transmit Buffer Register is made of Transmit Buffer register ($SPIxTXB$) and the Receive Buffer register ($SPIxRXB$); which respectively are used to write the data to be transmitted and read the data that was received, in the standard mode.

Since the FTDI chip operates only as a SPI master, the PIC24E micro-controller needs to be programmed to run as a slave. To configure the SPI module in slave configuration,

first the $SPIxBUF$ is cleared. Then, SPIx Interrupt Flag Status ($SPIxIF$) is cleared. Then, SPIx Event Interrupt is enabled by setting the ($SPIxIE$), after which appropriate priority is set up for the interrupt. Then, $SPIxCON1$ is configured to enable slave mode. As the last step, in $SPIxSTAT$ register, the receive overflow flag is cleared and the SPIx module is enabled, which starts the operation of the module. The data to be transmitted or received can be written or read from the $SPIxBUF$ respectively.

5.2 Programming the FT4232H USB controller

FT4232H, the Quad High Speed USB to Multipurpose UART/MPSSE chip from FTDI, has four ports, all of which can be used for UART communication, and two of which can also be used for Multi-Purpose Synchronous Serial Engines (MPSSE). MPSSE supports synchronous serial protocols such as JTAG, I2C and SPI. The high-speed USB device supports up to 480Mb/s. FT4232H supports data transfer rates up to 12Mbit/s in the UART mode. The MPSSE is capable of operating at speeds up to 30Mbits/s[4].

FTDI offers free drivers for Windows, Linux and Mac operating systems. These drivers must be installed before proceeding with programming or using the FTDI device. If the device is used in the UART mode, the Virtual COM Port (VCP) driver must be installed. If the device is used in the MPSSE mode, the D2XX Direct driver must be installed. VCP drivers emulate the standard PC serial port, and thus creates a virtual serial port for each module connected. The D2XX driver is used when direct access to the USB device is needed. The device can be accessed using a DLL interface or by using one of the libraries provided in user's program.

FT4332H does not have any built-in program memory and the USB protocol, MPSSE and UART modules are implemented in hardware in the FT4332H. Hence, the chip doesn't require to be programmed directly to perform a specific function. However, it needs to be configured properly to perform the required action. The configuration data is stored in an EEPROM attached to the chip.

FTDI provides a free utility called FT Prog, which can be used to select various configuration options and can be used to flash the EEPROM. Figure 5.3 shows a screenshot

of the FT Prog utility.

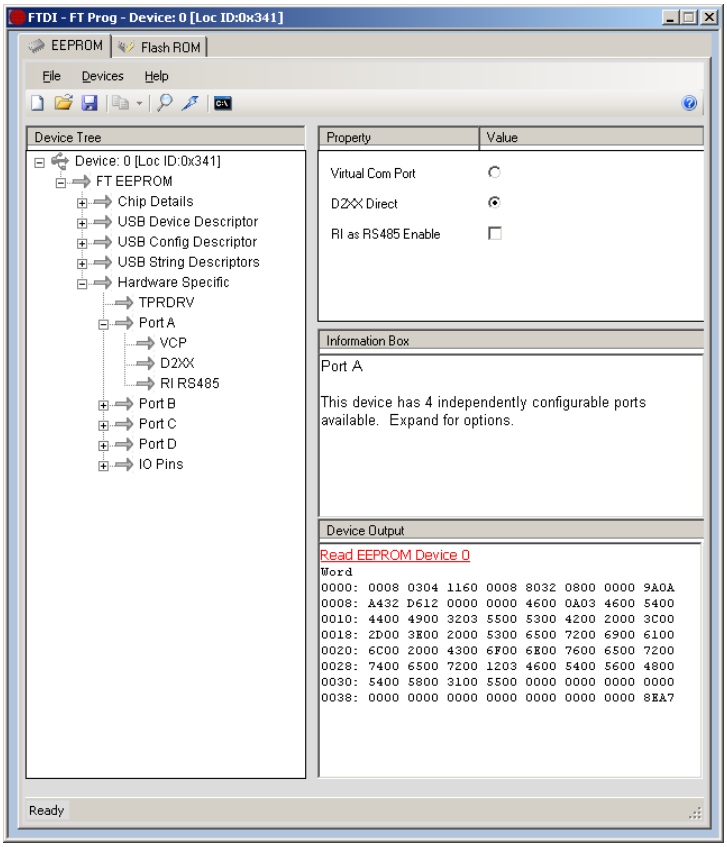

Figure 5.3: FTDI FTProg utility

FT Prog utility offers the option to configure various features of the device, such as the device descriptor, the Vendor ID (VID) and Product ID (PID) and other basic device parameters. The utility also can configure each port of the device to be operated in VCP mode or D2XX mode. When using the bit-banging mode, the I/O ports can also be configured bitwise. The utility can also be used to read data from a device for further identification or debugging. It also has the option of saving the chip configuration in a file on the PC, for future use or reference. A command-line variant of the utility is available for batch programming.

5.3 Programming the computer

The computer needs to be programmed to communicate properly with the controller board, and to properly simulate the control system in real time. The communication between the board and the computer is via USB channel. Since FT4232H does not have any programming memory, the operating instructions must be given from the computer. Thus, based on the mode of operation of the FTDI chip, the programming on the computer side would differ. The control system designed in MATLAB/Simulink® is configured to run in real-time using the Real-Time Windows Target blockset.

5.3.1 Using FT4232H in UART mode

To operate FT4232H in UART mode, no special programming is required. When a port on the FT4232H is configured as a VCP using FT Prog utility, the port is automatically configured as a UART port, and is mapped on the computer as a COM port (serial port). The Virtual COM Port can then be accessed like any other COM port. UART configuration details such as baud rate, number of stop bits, number of start bits, flow control, etc. are inherited from the configuration of the Virtual COM Port on the computer. This eliminates the need for additional programming.

5.3.2 Using FT4232H in MPSSE mode

Operating the FTDI chip in MPSSE mode, for serial communication in SPI protocol between the micro-controller and FT4232H, requires additional programming. The MPSSE port is configured as to operate in D2XX using the FT Prog utility. The port is then accessed using the LibMPSSE-SPI library provided by FTDI.

The user application runs on top of the LibMPSSE-SPI library, which in turn runs on top of the D2XX driver API. The D2XX driver directly communicates with the FTDI chip, which has USB protocol implemented in hardware. The FTDI chip communicates with the micro-controller using the MPSSE in SPI mode.

The LibMPSSE-SPI library is written in C, and can be accessed from user programs written in any compatible language. The Application Programming Interface (API)

consists of six control APIs and two data transfer APIs. The library includes SPI functions, which are used for configuring and operating the SPI channel; and GPIO functions, which can be used for general purpose input/output operations. The library also has library infrastructure functions, which are generally not used in user programs. Usage example can be found in the application note that is included in the library package[5].

Once the SPI channel is set up, the data stream must be converted into a format suitable for MATLAB/Simulink®'s Real-Time Windows Target. The acceptable data streams that can be used are serial port and UDP port. Hence, the data from the MPSSE module must to converted to a virtual serial port or a virtual UDP port. Implementation of this approach is out of the scope of this thesis.

5.3.3 MATLAB/Simulink® programming

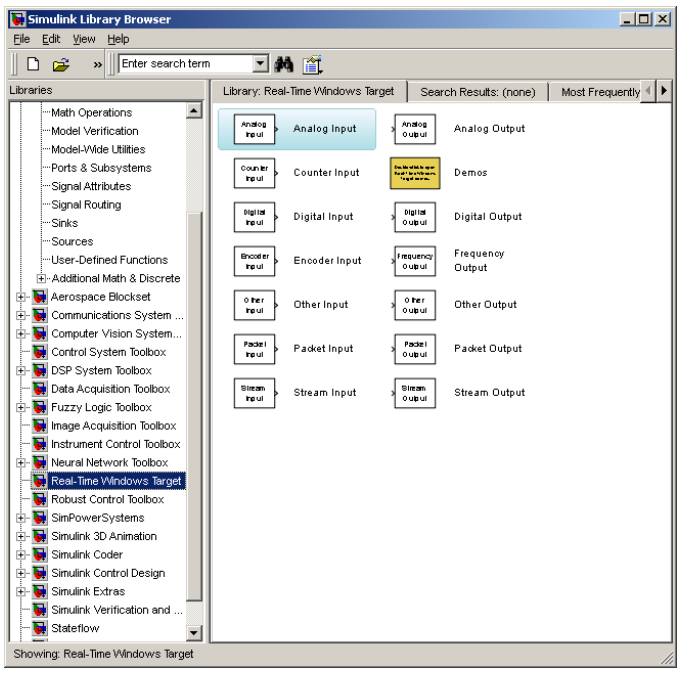

Figure 5.4: MATLAB/Simulink® Real-Time Windows Target block set

The control system designed in MATLAB/Simulink® uses the Real-Time Windows

Target block set for input and output functions. Other components of the system are imported from regular block sets. Figure 5.4 shows the blocks available in the Real-Time Windows Target block set.

Stream input and stream output blocks can be used as inputs and outputs for the control system respectively. The blocks must then be configured to use the appropriate board. Let us consider configuring the analog input block while the FTDI chip is used in UART mode.

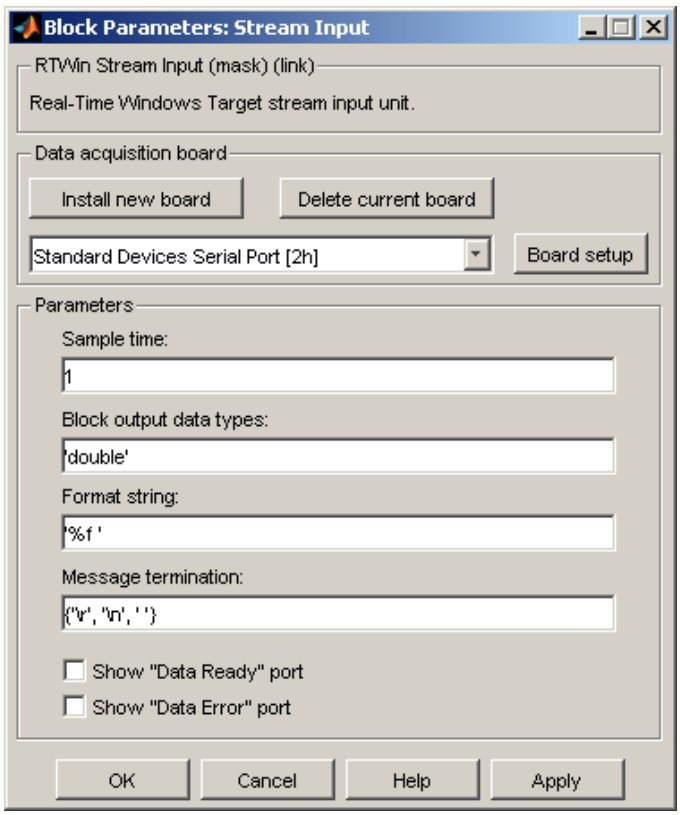

Figure 5.5: Stream input block parameters

The block parameters as shown in figure 5.5 can be used to configure the sample time, block output data type, formatting string, termination character and status reports. To use the block, first a new board must be installed. A standard serial port is chosen as the board type. The port configuration dialog that opens up has configuration options

for setting up the UART communication. As shown in figure 5.6, options such as which serial port has to be used, the number of data bits, parity, number of stop bits, baud rate and the type of flow control can be configured in the dialog box.

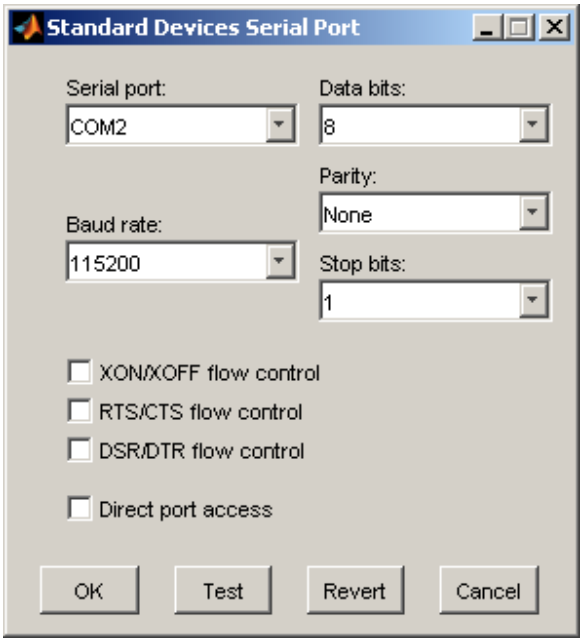

Figure 5.6: Board setup

Once the board is set up, the communication link can be tested by clicking the "Test" button in the dialog box.

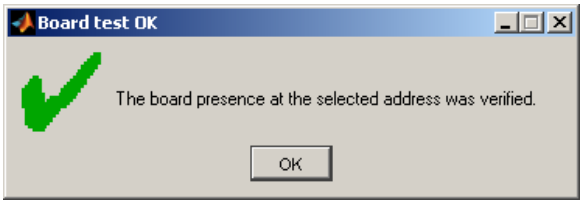

Figure 5.7: Board verification

If the board was installed and configured properly, a positive test verification as shown in the picture above will be shown.

Once the board is set up as discussed above, rest of the control system can be designed using regular blocks, the only difference being the change of input sources and output sinks to the input/output block in the Real-Time Windows Target block set.

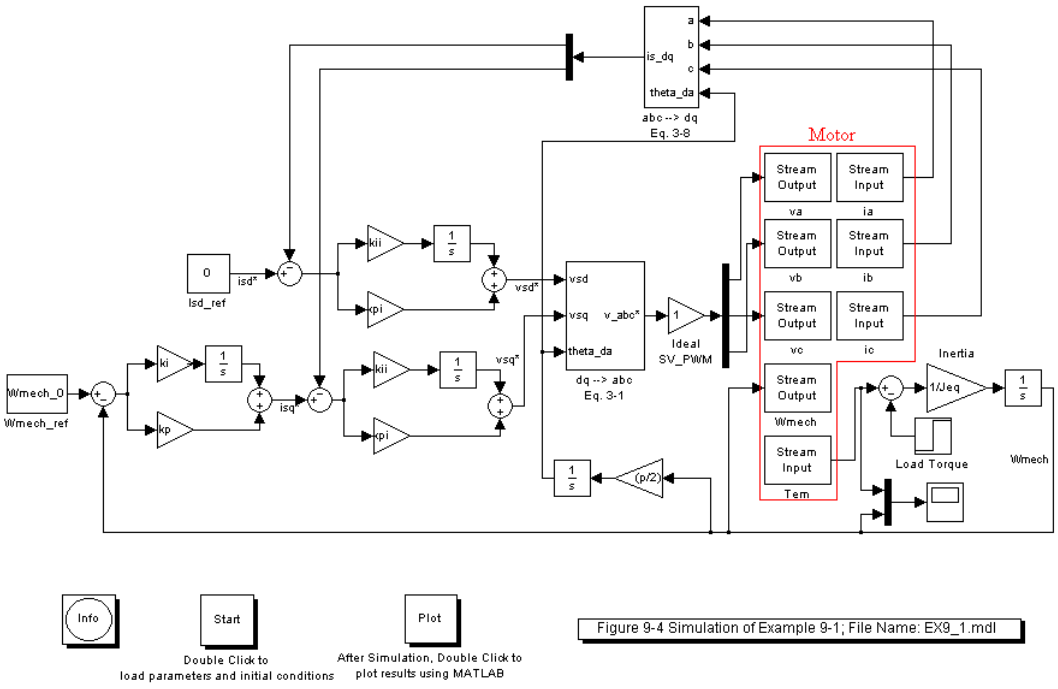

Figure 5.8: Sample control system

Figure 5.8 shows a sample control system designed using this approach[2], with the motor being replaced with stream outputs and sensor signals being fed in as stream inputs.

Chapter 6

Results

The hardware-in-the-loop controller board was designed, laid out and assembled. Figure 6.1 shows a picture of the assembled board.

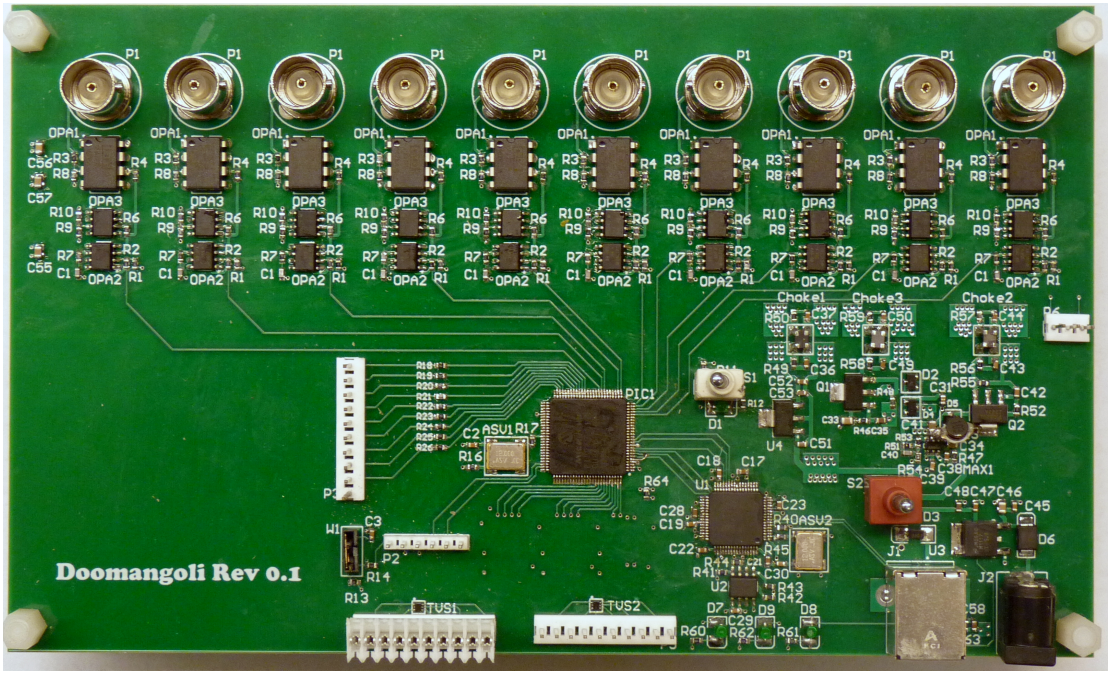

Figure 6.1: Assembled board

Basic features of the board, such as connectivity, operation of power supplies, etc. were tested and found to be normal.

The USB communication channel was established, operating the FTDI chip in UART mode. The ADC values could be read from the analog input channels using HyperTerminal, a serial communication. A sample set of output values as recorded in HyperTerminal is shown in figure 6.2

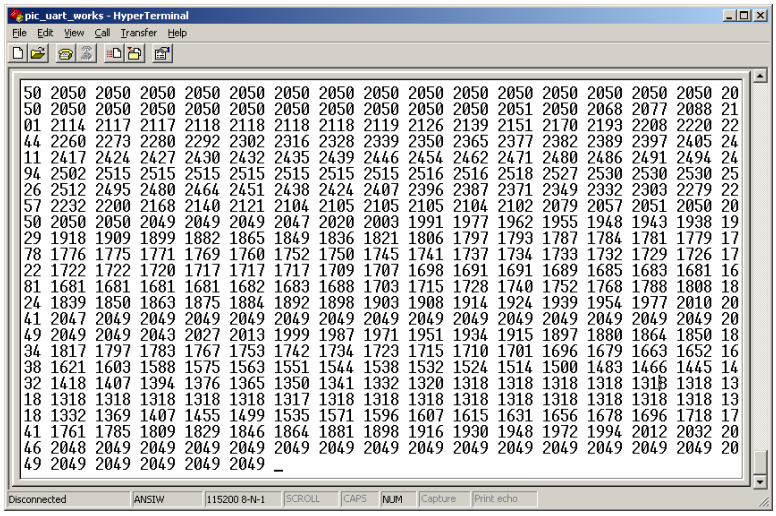

Figure 6.2: Sample ADC output in HyperTerminal

Figure 6.3 shows a sample PWM output, with the signals being generated by the microcontroller at a time period of 1ms and a duty ratio of 0.7. The output was observed to be at the 5V voltage level as required.

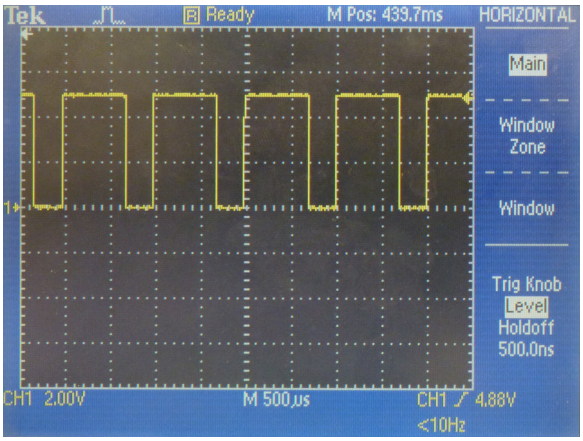

Figure 6.3: Sample PWM output

The FTDI chip was also programmed to run in the MPSSE SPI mode, but the desired results could not be obtained. The communication signals, viewed on an oscilloscope, appeared as expected, however, the chip could not read the signals and respond as expected. There might have been errors in configuring the SPI channel, which could not be detected and debugged. Also, further programming on the computer side was required for the board's complete functioning as a hardware-in-the-loop converter.

Thus, the controller hardware was built and tested satisfactorily, and minimal firmware to demonstrate essential features of the board was developed.

Chapter 7

Conclusion and Future Work

This thesis presents the design and implementation of a hardware-in-the-loop converter for electric drive applications. The control system is designed in MATLAB/Simulink® using the Real-Time Windows Target block set. The controller board that was designed forms a communication link between the power board and the computer running the control system in real time. The controller board also reads the analog signals from the sensors on the power board and generates PWM signals for driving the gates on the power board. The system was built and the essential features of the controller were verified. It was learnt that the concept is feasible. There have been issues with the communication link between the USB controller and the micro-controller, that needs improvement.

Future work would mainly involve optimizing the micro-controller firmware and programming the computer to allow the use of high-speed SPI communication between the micro-controller and the FTDI chip. The system must be put into further rigorous tests to match the speed and bandwidth requirements for motor drives. Further work might also involve updating the PCB layout and design to match new footprints and make minor changes in the schematics.

There are concerns with the real-time capabilities of Windows operating system and the Real-Time Windows Target block set. The effectiveness of the current approach for electric drive applications can be verified only by actual testing. If the system is not

able to support the required performance, various other approaches can be considered for utilizing the controller that has been developed. One such approach would be using Simulink Coder for compiling the Simulink models into C code, which can then be compiled for the micro-controller and entirely or partly be executed by the micro-controller. This would reduce the load on the computer, which could then be used only as a data visualization device. Another approach could be using a dedicated real-time machine as the target, to which the controller board could be connected. MATLAB/Simulink®'s xPC Target could be used for this purpose. The target machine for this purpose need not be extremely powerful, and hence the desired cost reduction might still be achieved.

Building upon this controller in one of the directions suggested, a set up could be developed for teaching electric drives lab in universities, which would enable the students to design a control system for motors and test them instantly, without the need for expensive industry grade hardware.

References

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

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

[3] Microchip Technology Inc., *High-Performance, 16-bit Digital Signal Controllers and Microcontrollers*, PIC24EP512GU810 Data Sheet, December 2009 [Revised August 2011].

[4] Future Technology Devices International Ltd, *FT4232H Quad High Speed USB To Multipurpose UART/MPSSE IC*, FT4232H Datasheet, 4th November 2008 [Revised 17th November 2010].

[5] Future Technology Devices International Ltd, *User Guide For libMPSSE - SPI*, FT4232H Application Note, 1st August 2011 [Revised 12th December 2011].

[6] Pete Wilson, P.E., Texas Instruments, *High-Voltage Signal Conditioning for Low-Voltage ADCs*, Application Report, June 2004.

[7] Maxim Integrated Products, *USB-Powered Bipolar Supply*, MAX1896 Application Note, Aug 18th, 2009.

Appendix A

PIC24E Firmware

A.1 Configuration File

```
// PIC24EP512GU810 Configuration Bit Settings


_FOSCSEL(
    FNOSC_PRI &
                // Initial Oscillator Source Selection bits
    IESO_OFF
                // Two-speed Oscillator Start-up Enable bit
);

_FOSC(
    POSCMD_EC &
          // Primary Oscillator Mode Select bits

    OSCIOFNC_OFF &
          // OSC2 Pin Function bit

    IOL1WAY_OFF &
```

```
        // Peripheral pin select configuration

    FCKSM_CSDCMD
        // Clock Switching Mode bits
);

_FWDT(
    WINDIS_OFF &
        // Watchdog Timer Window Enable bit

    FWDTEN_OFF
        // Watchdog Timer Enable bit
);

_FICD(
    ICS_PGD2
        // ICD Communication Channel Select bits

);
```

A.2 PWM Generation

```
#define FCY 6000000
#define TestLed LATGbits.LATG15
#define TrisTestLed TRISGbits.TRISG15
#define PWM1 LATAbits.LATA6
#define TrisPWM1 TRISAbits.TRISA6

/*
 * File:   newmain.c
 * Author: raja0128
 *
 * Created on May 17, 2012, 3:08 PM
```

```
 */

#include <p24EP512GU810.h>
#include "config.h"
#include "hardware.h"
#include <libpic30.h>

int t;

void delay_ms(int t)
{
    int i=0;
    while(i<t){
        __delay_us(97);
        i++;
    };
}

void pwm(int d)
{
    PWM1=1;
    delay_ms(d);
    PWM1=0;
    delay_ms(10-d);
}
int main() {
    TrisTestLed = 0;
    TrisPWM1 = 0;
    int d = 7; //duty ratio *10, freq 1kHz
     while(1){
         pwm(d);
    }
```

```
    return 0;
}
```

A.3 Reading ADC values using UART

```
#define FCY 6000000
#define BaudRate 115200
#define TestLed LATGbits.LATG15
#define TrisTestLed TRISGbits.TRISG15



#include <p24EP512GU810.h>
#include "config.h"
#include "hardware.h"
#include <libpic30.h>


long int data;

void send_long(long int data)
{
    int i;
    int divisor=1000;

    for(i=0;i<4;i++)
    {
        while(U1STAbits.UTXBF == 1);
        U1TXREG = (int) (data/divisor+ 48);
        data=data%divisor;
        divisor/=10;
    }
```

```
    while(U1STAbits.UTXBF == 1);
    U1TXREG = ' ';
}

void UARTInit(){
    RPINR18bits.U1RXR = 118; //map to RP118
    RPOR14bits.RP120R = 0x1; //1 is the UART1 TX output

    U1MODE=0; //clear all U1MODE register
    U1MODEbits.BRGH = 1; //highest baudrate select

    U1BRG =(FCY/(BaudRate*4))-1 ;

    U1STA = 0; //clear all U1STA register

    U1MODEbits.UARTEN = 1; // Enable UART1
    U1STAbits.UTXEN = 1; //Enable transmit
    IFS0bits.U1RXIF = 0; //Clear UART1 Received interrupt flag

}

void ADCInit()
{
    //Turn on, auto sample start, auto-convert
   AD1CON1 = 0x80E4;

    AD1CON4bits.ADDMAEN = 0;        //no DMA
    AD1CON1bits.AD12B = 1;          //12 bit mode
    AD1CON2bits.CHPS = 0;           //select channel 0
    AD1CON2bits.VCFG = 0;           //Voltage ref. AVdd, AVss
    AD1CON3bits.ADRC = 1;           //Clock - Internal RC
    AD1CON1bits.FORM = 0;           //Output format: Unisigned Int
```

```
    AD1CON2bits.SMPI = 0;          //Generates interrupt after
                                   //completion of every sample
                                   //or conversion operation
    ANSELBbits.ANSB10 = 1;         //RB10/AN10/Pin34 configured
                                   //as an analog input
    AD1CHS0bits.CH0SA = 10;        //AN10 for CH0 +ve input
    AD1CHS0bits.CH0NA = 0;         //Vref- for CH0 -ve input
    AD1CSSL = 0;                   //No scanned inputs
    AD1CSSH =0;                    //No scanned inputs
    AD1CON1bits.ADON = 1;          //module is in active mode,
                                   //fully powered & functional
}

void ADCGetVal()
{
    while (!AD1CON1bits.DONE);
    data = ADC1BUF0;
}

int main() {
     data = 0;
     UARTInit();
     ADCInit();

     while(1){
          __delay_ms(10);
          ADCGetVal();
          send_long(data);
     }
     return 0;
}
```

A.4 SPI Communication

```
#define FCY 6000000
#define TestLed LATGbits.LATG15
#define TrisTestLed TRISGbits.TRISG15

void assign_pins()
{
    // Configure input/output pins////
    __builtin_write_OSCCONL(OSCCON & ~(1<<6));
                        //unlock PPS registers

    //Configure inputs
    RPINR21bits.SS1R=0x33;
                        //assign SPI Slave Select 1
                        //(SS1) to pin RPI51
    RPINR20bits.SCK1R=0x56;
                        //assign SCK1 to pin RPI86
    RPINR20bits.SDI1R=0x31;
                        //assign SDI1 to pin RPI49

    //Configure output pins
    RPOR6bits.RP87R=5;
                        //assign SDO1 to pin RP87
    __builtin_write_OSCCONL(OSCCON | (1<<6));
                        //lock PPS registers

}

/*
 * File:   newmain.c
 * Author: raja0128
 *
```

```
 * Created on May 17, 2012, 3:08 PM
 */

#include <p24EP512GU810.h>
#include "config.h"
#include "hardware.h"
#include <libpic30.h>

void write_SPI(short command)
{
        short temp;
        temp = SPI1BUF;
        // dummy read of the SPI1BUF register
        //to clear the SPIRBF flag
        SPI1BUF = command;
        // write the data out to the SPI peripheral
        while (!SPI1STATbits.SPITBF)
                        // wait for the data to be sent out
                ;
}
int main() {
    assign_pins();
    TrisTestLed = 0;
    // 2. init the SPI peripheral ////
        SPI1BUF = 0;
        IFS0bits.SPI1IF = 0;
                        // Clear the Interrupt flag
        IEC0bits.SPI1IE = 0;
                        // Disable the interrupt

        // SPI1CON1 Register Settings
        SPI1CON1bits.DISSDO = 0;
```

```
                    // SDOx pin is used by the module
        SPI1CON1bits.MODE16 = 1;
                    // Communication is 16 bits wide
        SPI1CON1bits.SMP = 0;
                    //SMP bit must be cleared when SPIx
                    //module is used in Slave mode.
        // SPI Mode 1
        SPI1CON1bits.CKE = 0;
                    //(CPHA=1)Serial output data changes
                    //on transition from Idle clock
                    //state to active clock state
        SPI1CON1bits.CKP = 0;
                    // (CPOL=0) Idle state for clock
                    // is a low level; active state
                    // is a high level
        SPI1CON1bits.SSEN=1;
                    // Slave Select pin is used by module
        SPI1CON1bits.MSTEN = 0;
                    // Master mode disabled
        SPI1STATbits.SPIROV=0;
                    // No Receive Overflow has occurred
        SPI1STATbits.SPIEN = 1;
                    // Enable SPI module
    while(1){
    TestLed = 1;
    write_SPI(7);
    __delay_ms(100);
    TestLed = 1;
    __delay_ms(100);
    }
    return 0;
}
```