Robot Brain Board: A Microcontroller Board Design for Robotics-Oriented Motor Controls

A Design Project Report
Presented to the Engineering Division of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering (Electrical)

by
Ko Ihara
Project Advisor: Professor Andy Ruina
Degree Date: May, 2006

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title:

Brain Board: A Microcontroller Board Design for Robotics-Oriented Motor Control

Author:

Ko Ihara

Abstract:

Controlling a complex system such as a bipedal walking robot requires a sophisticated control algorithm with numerous feedback inputs and control outputs. The shortcomings of many off-the-shelf microcontroller kits include large physical size, power inefficiency, small number of digital ports, and poor computation capability. A printed circuit board was designed around Freescale MC56F8347 16-bit microcontroller to address the shortcomings of commercially-available robotics-oriented microcontroller boards. The physical pin configuration on the board makes it easy to develop an inexpensive custom daughter board, allowing the control "brain" of the robot to be specifically tailored to different actuator/sensor configurations. The board was successfully populated and tested to verify all specified functionalities.

Report Approved by

Project Advisor: _____Date: _____

Executive Summary

Controlling a complex system such as a bipedal walking robot requires a sophisticated control algorithm with numerous feedback inputs and control outputs. The shortcomings of many off-the-shelf microcontroller kits include bulky size, power inefficiency, small number of digital ports, and poor computation capability. The problem caused by poor computation capability of an off-the-shelf Microchip 8-bit microcontroller-system has been observed in the Biorobotic Laboratory's *Marathon Walker*, a 2-dimensional bipedal walking robot, in which side-to-side stability is mechanically guaranteed by a wide foot size. More computation capability will not only be desirable, but necessary in the future generation of "true" 3-dimensional bipedal walking robot designs.

An H-bridge circuit board was designed around ST Microelectronics VNH2SP30 motor driver to replace bulky off-the-shelf H-bridges that have poor refresh rate, high on-resistance, and low current rating. Another printed circuit board, named *Brain Board*, was designed around Freescale MC56F83x7 series of 16-bit microcontrollers to address the shortcomings of commercially-available robotics-oriented electronics kits, and to add additional on-board functionality toward the design of "true" 3-dimensional bipedal robot.

The physical pin configuration on the board makes it easy to develop an inexpensive custom daughter board, allowing the control "brain" of the robot to be specifically tailored to different actuator/sensor configurations. For example, the pulse-width modulation and digital output pins could be directly plugged into the receptacles on a daughter board that has H-bridge motor driver chips populated on it. Such customization would help to reduce the amount of external wiring required. The board was successfully populated and tested to verify all specified functionalities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The 2-dimensional bipedal “Marathon Walker” robot designed at Cornell University’s Biorobotics Laboratory takes advantage of gravity to realize a pendulum-like walking locomotion. While power-efficient in principle, the walking cycle of the robot is inconsistent and unstable. A major factor that hinders a stable walking locomotion is the low refresh rate of the control variables in the microcontroller.

The main electronic components of most robots include the main processing unit, actuators, and feedback sensors. The signal processing can be handled by a single microcontroller or a digital signal processor (DSP). Ideally, the processing unit would include such on-chip peripherals as analog-to-digital (A/D) and digital-to-analog (D/A) converters, pulse-width modulation (PWM) generators, quadrature decoders, and duplex data transmitters. Actuators on a robot may include various types of servos and DC motors. The feedback sensors may include tactile switches, potentiometers, accelerometers, gyroscopes, and quadrature encoders (usually embedded inside a motor). A/D conversion or quadrature-decoding of the feedback signals can be processed by the main processing unit, or off-chip integrated circuits (ICs).

Figure 1: 2-Dimensional Bipedal Walker

The main processing on the *Marathon Walker* robot is handled by the Innovation First, Inc.’s *Robot Controller*, which is a Microchip PIC18F8520 microcontroller-based system that runs at a core clock frequency of 40 MHz. Unfortunately, the *Robot Controller* suffers from a poor temporal resolution of the control: the main proportional-derivative (PD) control loop takes

over 10 ms to complete, which limits the control refresh rate to under 100 Hz. This system uses two PIC18F8520 microcontrollers, but the Innovation First's documentation on how the two chips communicate and interact is not publicly available, so it is difficult to debug a software-related problem when it arises. The potential performance of the robotics control could be greatly improved by switching the main controller to a faster digital signal processor, programming a more sophisticated control algorithm, and upgrading off-the-shelf hobbyist-level components to customized high-performance electronics. In the Biorobotic Lab's upcoming design of a 3-dimensional, "true bipedal" walking robot, a faster processing unit is not only desirable, but necessary to perform control algorithms for the added degrees of freedom.

2. PROJECT STATEMENT

Build a controller board for a walking robot that is energy efficient, capable of fast computations and provides a three-dimensional orientation and roll information on-board. The specifications are as follows:

- All active board components shall be powered by a single battery pack of supply voltage ranging from 7.5V to 25V.
- The physical board size shall be 3.0" x 2.3" or smaller.
- The board shall maximize power efficiency.
- The controller on the board shall make use of analog feedback signals of both 3.3V and 5V ranges.
- The controller on the board shall use feedback data from various sensors (switches, analog voltage, potentiometers, optical encoders, RX-232, and serially-transferred data).
- The controller on the board shall control multiple robotic actuators (DC motors, RC servos, solenoids) independently.

3. THEORY OF OPERATION

The main controller of the robot must be able to interpret numerous sensor inputs that come in various signal formats, and drive the actuators accordingly. This section describes the common formats of signals used in robotics applications.

3.1. GENERAL-PURPOSE INPUT/OUTPUT (GPIO) PORTS

Digital input/output signals are used to send or receive digital information. In the context of robotics control, a microcontroller could send a binary output signal to an H-bridge motor driver to specify a direction of output current (forward or reverse), or to drive or retract solenoid shafts. Configured as inputs, the microcontroller digital ports can receive signals from such binary sensors as mechanical or optical switches.

3.2. ANALOG-TO-DIGITAL CONVERTER (ADC)

An ADC is an electronic circuit that converts an analog input voltage to discrete digital numbers. The ADC has a given input voltage range, and an ADC with n-bit resolution evenly divides that analog voltage range into 2^n discrete "counts." For instance, a 10-bit ADC with an input voltage range of 0V-5V would express the input analog voltage as a digital number between 0 and 1,023: an input voltage of 3V would be expressed as 614, assuming perfect accuracy and zero noise. Some sensors that output analog voltage feedback signals are gyroscopes, accelerometers, and potentiometers mechanically coupled to the joint of two materials.

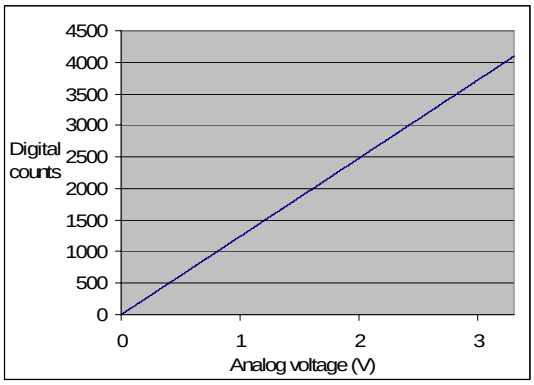

Figure 2: Conversion Graph for a 12-bit ADC w/ 3.3V Range

3.3. QUADRATURE DECODER

A rotary quadrature encoder is a digital electronic device used to convert the angular position of a shaft to a digital signal. The encoder usually consists of a circular disk that is mechanically coupled to a shaft. It has a series of radial slots cut into it, so that when a light generated by a light emitting diode (LED) passes through the slot, a photodetector such as a photodiode would generate an electrical pulse. A quadrature signal consists of two signals, always 90 degrees offset in phase as the shaft turns, so that a simple hardware such as that described by the finite state machine in Figure 3 can count the number of positive or negative transitions.

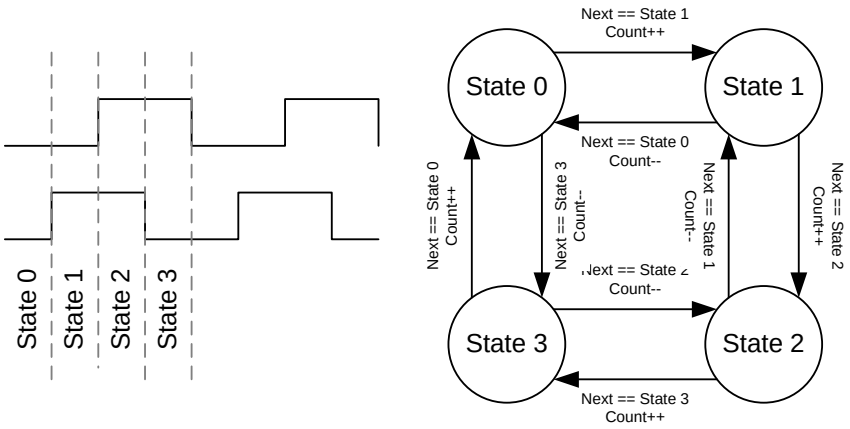

Figure 3: Quadrature Decoding

Rotary encoders are often coupled to motor shafts, and can have as many as, or greater than, 2,048 state transitions per revolution.

3.4. PULSE WIDTH MODULATION (PWM)

Pulse width modulation is a digital means of controlling an analog component. A pulse width modulated signal is a constant-frequency, adjustable duty cycle square pulse. In the context of robotics control, it is used to control the torque output of DC motors and the position of RC servos.

For example, a torque output of a DC motor could be controlled by adjusting the analog voltage level of the power supply. This task could also be accomplished digitally by switching the supply on and off at a high frequency with variable pulse width. In some H-bridge motor drivers, the pulse width can determine both the maximum torque and the direction of the DC motor.

The position of an RC servo axle can be controlled by sending a single pulse width between 0.5 ms (for 0 degree position) and 2.5 ms (for 270 degrees position). The position would reset unless the same pulse width is sent periodically, so setting up a PWM signal with one duty cycle would refresh the shaft position of an RC servo automatically.

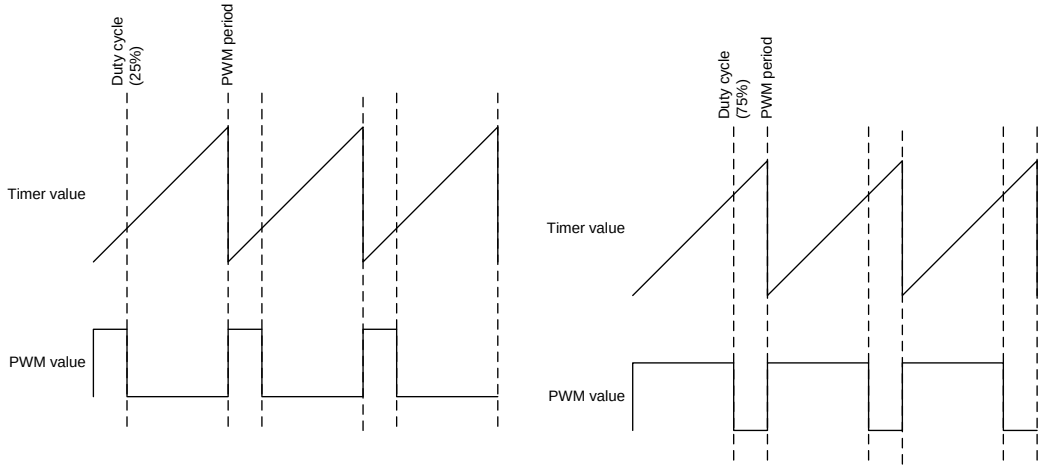

Figure 4: PWM Signal Examples

3.5. SERIAL PERIPHERAL INTERFACE (SPI)

SPI is a 4-wire duplex digital protocol for chip-to-chip communication. The four wires are generally specified as *RX* (receive), *TX* (transmit), *SCLK* (serial clock), and *CS* or *SS* (chip/slave select). In this communication protocol, the “master” chip, the “slave” chip, and the number of bits to transfer must be explicitly specified. The master sends a constant clock to the slave chip, and drives CS signal low to begin communication with the slave chip: for example, in a 16-bit SPI protocol, 16 bits of information would be sent or received in the first 16 rising- or falling- edges of the SCLK (the edge trigger specification may differ depending on the chip model). After the transfer is complete, the master would set the CS signal high. The CS signal could be driven low for longer than the number of SCLK cycles to receive or send all bits without corrupting the data transfer, since the important n-bits of information is transferred in the first n SCLK cycles.

SPI is an ideal chip-to-chip communication protocol in situations where the shared I/O ports are too valuable to be used in a parallel communication protocol. In a 16-bit transfer of data, a parallel communication protocol would require 16 I/O ports, plus any necessary control bits. SPI communication is considered a very fast inter-chip communication protocol, since the SCLK frequency could be driven as high as the master and the slave can operate at that frequency. In a microcontroller design, the microcontroller is often configured as the master, sends data/instruction to, and receives data from the slave peripherals on the SPI bus. Many peripheral ICs such as electronically-erasable programmable read-only memory (EEPROM) and ADCs use the SPI protocol.

4. H-BRIDGE BOARD DESIGN

The H-bridge driver used to drive the DC motors on the *Marathon Walker* robot is a rather bulky unit that has a significant on-resistance. A custom printed circuit board was designed around ST Microelectronics' VNH2SP30-E H-bridge IC. This H-bridge adjusts the amplitude and polarity of the output current based on the duty cycle of the input PWM signal, and the values of the two "direction" input signals.[1] Some of the benefits from using VNH2SP30 include small surface-mount package, high current capability (absolute maximum of 40A), high refresh rate (compatible with 20 kHz PWM signals), simple interface, low on-resistance, and built-in current measurement output. The PCB was designed with CadSoft's EAGLE Layout Editor. Refer to Appendices A and B for schematics and layout, respectively.

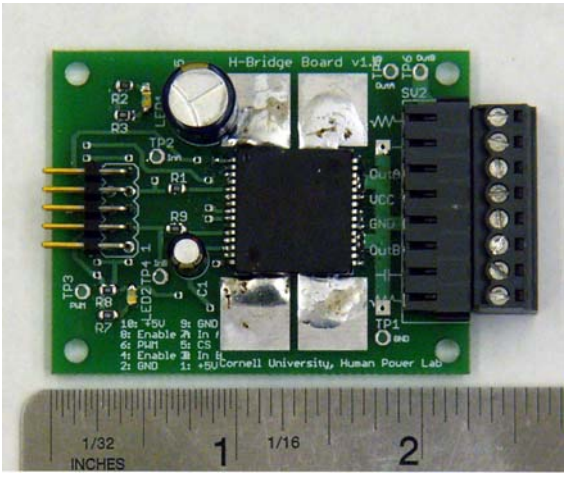

Figure 5: Populated H-Bridge Board

5. *BRAIN BOARD* MICROCONTROLLER PCB DESIGN

The two microcontroller boards previous used and evaluated in the Biorobotics Lab are Innovation First, Inc.'s *Robot Controller* and New Micro, Inc.'s *IsoPod* microcontroller board. The main goals of the *Brain Board* design are to carry all the desirable design aspects, to

[1] STMicroelectronics. "VNH2SP30-E Data Sheet." 30 April, 2006. http://www.st.com/stonline/products/literature/ds/10832/vnh2.htm.

eliminate all the shortcomings from previously evaluated system, and to add new functionalities toward the design of a true 3-dimensional bipedal walking robot.

Innovation First's *Robot Controller* is an 8-bit, dual Microchip PIC18F8520-based board. It is a popular off-the-shelf controller designed for robotics applications, and its header pins are configured to facilitate wiring with various sensors and actuators. The software is programmed in C language, under Microchip's proprietary *MPLABS* environment. There are several shortcomings with this system: the 8-bit microcontrollers' performance is rated at 10 million instructions per second (10 MIPS) at maximum, and this does not provide enough computation capability to run a single loop of proportional-derivative control algorithm for *Marathon Walker* under 10 ms. It has a poor minimum specified PWM refresh period of 2 ms. As mentioned before, the documentation on how the two microcontrollers communicate is not available, and the system appears much like a black box during software debugging. Lastly, the supply voltage regulation is inefficient, and the physical size is rather bulky at 3.4"x4.6"x0.75".[2]

New Micro, Inc.'s *IsoPod* system is a microcontroller board for Freescale's DSP56F805 microcontroller. Compared to Microchip PIC18F8520, DSP56F805 is a much faster processor, with a native 16-bit support and 40 MIPS rating. The software is programmed in hybrid C/assembler in Metrowerks *Codewarrior* environment, which provides a set of tools to facilitate fast software development. The main shortcomings of this system are the inefficient linear voltage regulation, small number of unshared GPIO pins, relatively small 64 kB program flash memory (compared to other models in Freescale's 56800E family), and ADC of only 8 channels compatible with only 3.3V-range inputs.[3]

The *Brain Board* design addresses all aforementioned design issues, and makes several improvements that makes it specifically geared towards advanced robotics-oriented actuator controls. The key hardware features that differentiate *Brain Board* from *Robot Controller* and *IsoPod* are listed below.

- Efficient supply voltage regulation through switching voltage regulator.

---

[2] Innovation First, Inc. "IFI Robotics – Mini Robot Controller." 30 April, 2006. http://www.ifirobotics.com/edu-rc.shtml.

[3] New Micros, Inc. "IsoPod V2." 30 April, 2006. http://www.newmicros.com/cgi-bin/store/order.cgi?form=prod_detail&part=IsoPod_V2&id=HiGR7s40vl3e02T11IEMo165h3tX5C7C.

- Generation compatibility with Freescale MC56F8347, MC56F8357, and MC56F8367, with up to 512 kB of program memory.
- 16-bit core with a 60 MIPS maximum rating.
- 32 ADC channels: 16 channels compatible with 5V range, the other 16 channels compatible with 3.3V range.
- Up to 7 hardware quadrature decoders.
- Large number of dedicated GPIO ports, over 40 when external memory access mode is not used.
- 3-dimensional orientation and roll information available via on-board accelerometers and gyroscopes.
- Small physical size of 3.0”x2.3”.
- A pin configuration facilitates the development of inexpensive daughter boards that makes the control system more customized to specific robotics sensor/actuator architecture.

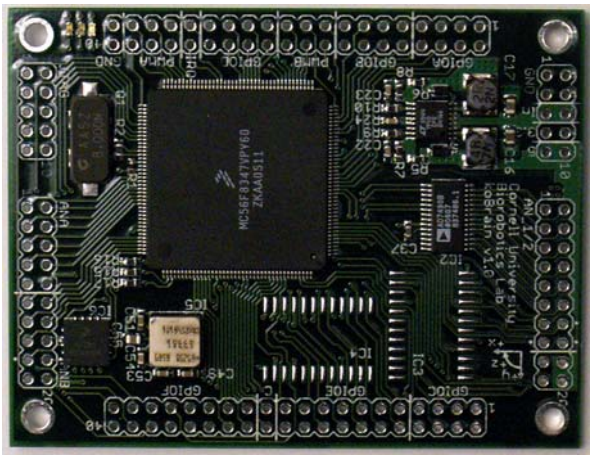

Figure 6: Populated Brain Board (Top)

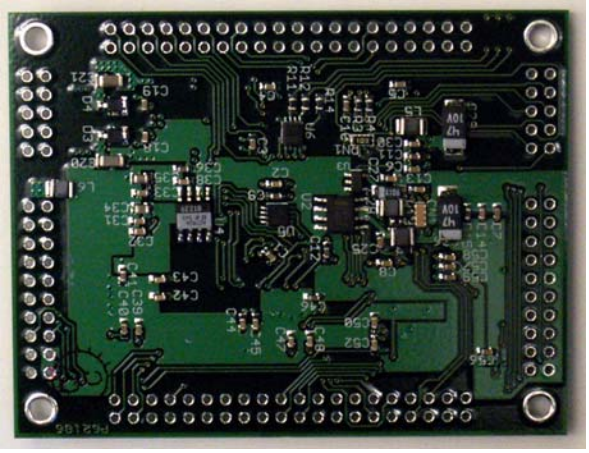

Figure 7: Populated Brain Board (Bottom)

5.1. COMPONENT SELECTION

Much time was invested into selecting electronic components for the *Brain Board* after the design specifications were defined and the shortcomings of previously-used systems were identified. During parts research, the emphasis was particularly placed on design risk reduction, power efficiency, small size, and high computation performance.

5.1.1. Microcontroller

The evaluation of the Freescale-based *IsoPod* system during fall, 2005, found the DSP56F805 core to be more than fast enough for a PD- and PID-based control algorithm used for *Marathon Walker*. The set of tools in Metrowerks Codewarrior helped to make the software development process very fast. After reviewing the *IsoPod* system, we found the Freescale products very favorable.

For the microcontroller core of the *Brain Board*, Freescale MC56F8347[4] was chosen for the following reasons. It is a native 16-bit machine that has a Harvard architecture, characterized

[4] Freescale Semiconductor. "56F8347 and 56F8147 Data Sheet." 30 April, 2006. http://www.freescale.com/files/dsp/doc/data_sheet/MC56F8347.pdf.

by separate instruction and data memories. It has a program flash memory of 128 KB, much larger than any compiled microcontroller software programmed in the lab, and the set of hardware peripherals specifically geared toward motor controls: 12 independent PWM channels, 16 channels of 12-bit ADC, 16 8-bit timers of which 7 could be configured as quadrature decoders, and two sets of SPI interface signals. The complete list of the microcontroller's hardware functionalities on the *Brain Board* is included in Appendix F.

The Brain Board, designed initially for MC56F8347, will also be compatible with MC56F8357 and MC56F8367 microcontrollers, which have larger memories and will be available in summer 2006. A unique feature of Freescale's 56800E family of microcontrollers is that they have an internal phase-locked loop (PLL) clock multiplier: for example, even though MC56F83x7 series' logic core runs on 120 MHz, the external crystal clock only needs to be 8 MHz, because the internal PLL multiplies the external input clock frequency. This feature isolates the high-frequency design risks from the printed circuit board (PCB) design.

Even though there are faster microcontrollers/processors for embedded applications than MC56F82x7 series, they do not have nearly as many motor-control-oriented hardware peripherals, or unshared GPIO pins. A possible alternative was to use a faster microcontroller core that communicates with a field-programmable gate array (FPGA) that is programmed to perform all the hardware peripheral functions. However, having a single-chip processing core would reduce the necessary physical board size, hardware complexity, chance of failure, and debugging complexity. For the reasons stated above, Freescale MC56F8347 was chosen as a single-chip processing core of the *Brain Board.*

5.1.2. Voltage Regulator

The actuators (DC motors, RC servos, etc.) are usually driven by a single battery pack ranging between 9V-20V; usually of lithium-ion, Nickel-Metal-Hydride, or Nickel-Cadmium types that have low internal resistance. It is desirable to use the same battery pack to power the digital logic that controls the motion of the robot as well, rather than have a separate power source. However, the voltage level must be reduced down to 5V or 3.3V that most digital logics run on.

A common means of regulating the voltage level of the power supply is a linear voltage regulator such as 7805. Most linear regulators have low dropout voltage (the minimum required difference between input and output voltage); and are simple to use, requiring connections only to input voltage, reference ground, and output regulated voltage. Unfortunately, linear regulators are very inefficient, and all the power from voltage regulation is dissipated as heat: for instance, if a 9V power supply is regulated via a linear regulator to run a 5V digital logic that requires 100mA, approximately 400mW of power is wasted in voltage regulation. This is not acceptable since the research robots in the Biorobotics Lab use higher-voltage battery packs, and power efficiency is one of the important agendas of the Lab.

Switching regulators offer a higher-efficiency solution to voltage regulation. It consists of an integrated circuit that switches the load current on and off at a high frequency, and a set of capacitors and inductors to stabilize the output voltage. Although most switching regulators can achieve efficiency over 80%, it is not a single-chip solution and the external circuitry is necessarily complex. An external circuit design will impose considerable risk on a hardware designer who does not have prior experience with switching power supplies.

Considering the benefits and design risks involved, I decided to use Linear Technology LT1940[5] switching regulator IC, which achieves about 85% regulation efficiency. This IC has a dual output and is thus capable of providing both 5V and 3.3V supplies from one IC. Compared to other switching regulator ICs from other companies such as National Semiconductors, LT1940 requires comparatively simple external circuit, and the output voltage levels are determined by simple voltage dividers. The theory of operation and design processes are thoroughly documented by the company, which greatly facilitate the circuit design.

---

[5] Linear Technology. "Datasheet: LT1940/LT1940L – Dual Monolithic 1.4A, 1.1MHz Step-Down Switching Regulator." 30 April, 2006.
http://www.linear.com/pc/productDetail.do?navId=H0,C1,C1003,C1042,C1032,C1064,P2241.

5.1.3. Analog-To-Digital Converter (ADC)

The Freescale MC56F83x7 microcontroller has a 16-channel ADC, but it only has a 3.3V input range. The number of ADC channels may not be enough for complex robot architecture with numerous feedback sensors, and many sensors run on a 5V supply. Therefore, an external ADC that can accept 5V input voltage range and communicate with the microcontroller is necessary.

Analog Devices AD7490[6] ADC was chosen for the board design among the company's other offerings for the following reasons: this IC has the largest number of input channels (16) of Analog Devices' ADCs that communicate on SPI interface. SPI is a 4-wire serial communication protocol, and depending on the master microcontroller capabilities, it can communicate at serial clock frequencies well over 1 MHz. SPI interface is much more desirable than parallel interface in the context of the *Brain Board* design, since it keeps greater number of I/O ports free on the microcontroller, and the communication rate is still very fast. AD7490 has 12-bit resolution, which makes the measurement precise to approximately 1.25mV for a 0V-5V analog input range. Lastly, the maximum sampling rate of AD7490 is 1,000,000 samples per second (1 Msps), which is much faster than is necessary.

5.1.4. Accelerometer

For a true 3-dimensional bipedal robot, it would be necessary to obtain 3-dimensional tilt feedback information. A 3-axis accelerometer can provide tilt information with respect to gravity, assuming no external jolt that can introduce noise. Companies such as Freescale, Analog Devices, and Kionix manufacture single-chip 3-axis accelerometers. Freescale MMA7260Q[7] was chosen for the Brain Board design because of selectable acceleration range, and larger pin pitch that facilitates the board population process. While accelerometers from Analog Devices and Kionix have constant acceleration range (usually between ±1g and ±10g), the MMA7260Q

---

[6] Analog Devices, Inc. "AD7490 Data Sheet, Rev. A." 30 April, 2006. http://www.analog.com/UploadedFiles/Data_Sheets/400753325AD7490_a.pdf.

accelerometers allows the user to select among ±1.5g, ±2g, ±4g, and ±6g depending on the digital input values on its two "g-select" pins. The physical pins on this IC have a pitch of 1 mm, as opposed to 0.65 mm and 0.5 mm pitches for Analog Device and Kionix ICs, respectively. A wider pin pitch reduces the risk of component placement errors during board population.

5.1.5. Gyroscope

For a true 3-dimensional bipedal robot, it would also be helpful to have a set of on-board 3-axis gyroscopes that provide the roll information in three coordinate axes. Gyroscope ICs are rather complex micro electro-mechanical sensors (MEMS), and only Analog Devices and Kionix are well-known manufacturers of them. There is no single-chip device for 3-dimensional gyroscope measurement, except for MEMSense's *AccelRate3D*[8], a 0.7"x0.7"x0.4" unit that can provide 3-axis accelerometer and gyroscope measurements. However, this unit costs around $1,000 each, and is a rather bulky block. Analog Devices gyroscopes have an axis of rotation perpendicular to the chip surface, while Kionix gyroscopes have an axis of rotation parallel to the chip surface. Therefore, it is possible to obtain 3-dimensional gyroscope signals by using two Kionix gyros and one Analog Devices gyro. For this reason, Kionix KGF01 and Analog Devices ADXRS401[9] gyros were selected.

5.2. DESIGN PROCESS

Both the H-bridge and the *Brain Board* were designed using CadSoft's EAGLE PCB layout editor[10]. To minimize cost, the H-bridge board was designed with 2 layers, while the

---

[7] Freescale Semiconductor. "MMA7260Q Data Sheet." 30 April, 2006.
http://www.freescale.com/files/sensors/doc/data_sheet/MMA7260Q.pdf.
[8] MemSense. "AccelRate3D Data Sheet." 30 April, 2006.
http://www.memsense.com/downloads/datasheets/AccelRate3D_SMT_Datasheet_revD1.pdf.
[9] Analog Devices, Inc. "ADXRS401 Data Sheet, Rev. 0." 30 April, 2006.
http://www.analog.com/UploadedFiles/Data_Sheets/279107307ADXRS401_0.pdf.
[10] CadSoft Online. "CadSoft Online: Home of the EAGLE Layout Editor." April 30, 2006.
http://www.cadsoftusa.com/.

Brain Board has 4 layers with two dedicated internal power and ground planes for uniform power distribution.

First, a custom library of all necessary components was prepared in EAGLE. Each custom “device” in a library consists of a schematic symbol, and a corresponding physical layout “land.” Figure 8 and Figure 9 show the schematic symbol and the corresponding land for Freescale 56F83x7 device. Custom devices were created for all electronic components used that are not included in EAGLE’s default library.

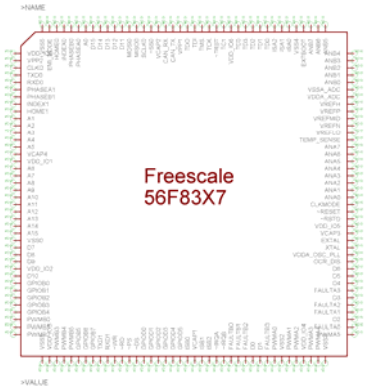

Figure 8: Schematic Symbol for Freescale 56F83x7

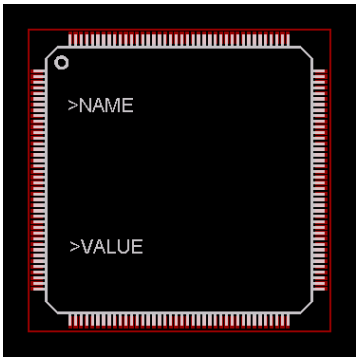

Figure 9: Physical Lands for Freescale 56F83x7

After all custom devices were created in the device library, pages of circuit schematics were drawn. In this process, the schematic symbols from the device libraries were laid out on several pages; and nets, or electrical connection among IC pins, are drawn. Specifications and recommended circuits from all IC datasheets were carefully reviewed for schematic creation. Refer to Appendix C for the *Brain Board* schematics.

After the schematics are drawn, the EAGLE performs “schematic capture” and places all the components’ layout lands in the layout drawing, with lines called “airwires” showing which pins are supposed to be physically connected together by metal traces. The positions, orientations, and layer (top or bottom layer) must be manually defined for each component land. Different component lands cannot overlap one another, and much effort was spent on land placement, so that it would be easy to draw traces between pads. Traces for different signal cannot overlap each other, so multiple metal layers, separated by electrically-insulating layers were used. For the *Brain Board*, four layers were used, with top and bottom layers as component and routing layers, and two internal layers as power and ground planes.

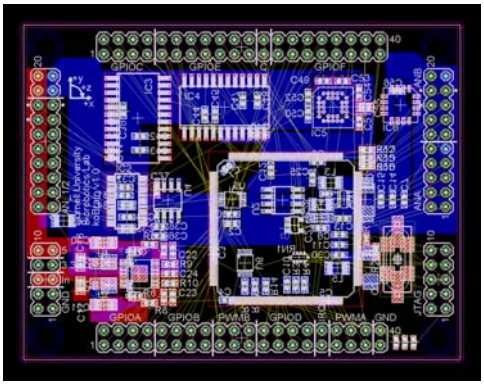

Figure 10: Brain Board Layout with Airwires

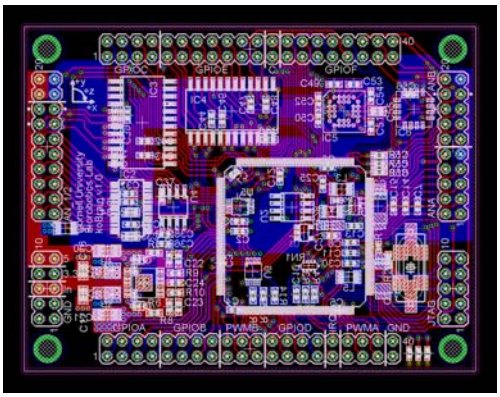

Figure 11: Brain Board Layout after Traces Drawn

On the Brain Board, “pseudo planes” were created for analog reference voltage signals. This can be observed in the top and bottom (red and blue, respectively) layers in Figure 11.

Because of very high trace density, the signal traces were all laid out manually, without any help of EAGLE's autorouter feature.

After all airwires were replaced with signal traces, a computer-aided manufacturing (CAM) file was generated from the board layout. The CAM file format must be compatible with the manufacturing system used by the printed circuit boardhouse. For this project, the board manufacturing was contracted to Advanced Circuits, and the compatible CAM file in GERBER 274-X format, generated by EAGLE, was electronically sent to the company for manufacturing.

It would be very difficult, if not impossible, so solder all components on the *Brain Board* by hand. Mistakes in circuit board population would be costly, as some components such as gyroscopes costs over $50. The board population was contracted to MPL Incorported, a company that specializes in circuit board assembly. For board assembly, MPL used the GERBER 274-X files to make a stencil, a thin metal foil with lands cut out, so that it could be placed over the printed circuit board and solderpaste can be poured precisely on the land pads. All the components were then placed on the solderpaste-covered lands and placed into a reflow soldering oven.

6. TESTING

Functional testing was performed for both H-Bridge board and the *Brain Board*, to verify functionalities and to evaluate performance.

6.1. H-BRIDGE BOARD TESTING

New Micros *IsoPod* system was used to test the H-bridge board. The test experiment uses the angular position feedback from the DC motor's embedded quadrature encoder to make the DC motor follow a time-based profile programmed on the *IsoPod's* DSP56F805 microcontroller.

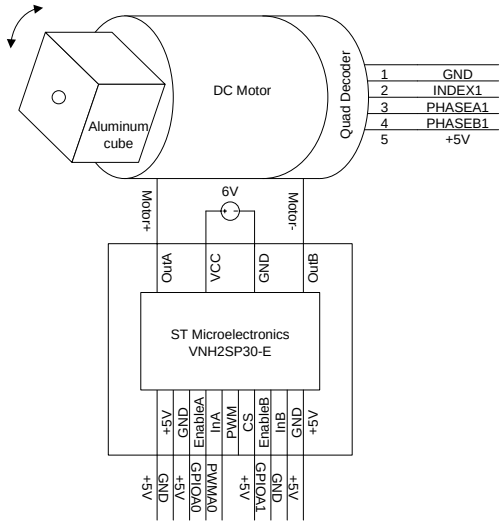

Figure 12: H-Bridge Board Test Circuit

Figure 13 shows the time-based performance of the PID control algorithm for a "step" target profile. As show in the figure the system has a rise and fall times of approximately 50 ms to make a -360-to-360 degree turn. The maximum overshoot is less than 1%.

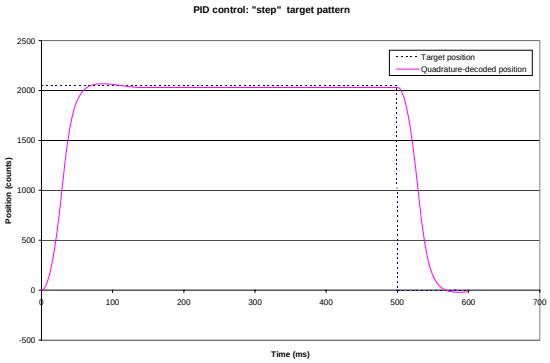

Figure 13: H-Bridge Test Result for "Step" Target Profile

The PID control's performance is even better when it is programmed to follow a smoother target profile. For a periodic "zigzag" target profile, the temporal delay of the measured position from the target position is under 10 ms. This experiment simulates the performance of the H-bridge board for a DC motor with very low natural damping, such as a DC motor with a light load or with very high gear ratio.

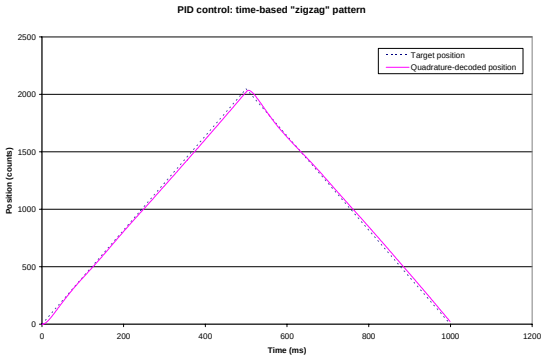

Figure 14: H-Bridge Test Results for "Zigzag" Target Profile

6.2. BRAIN BOARD TESTING

With MC56F8247 running, the 3.3V output of the switching voltage regulator was measured with a voltmeter to provide an average of 3.31V, with an RMS noise of 0.2mV. The

5V output side was measured to have an average output of 4.99V, with a negligible RMS noise not measurable by the voltmeter or the oscilloscope.

GPIO ports PORTA7:0, PORTB7:0, PORTC7:0, PORTD7:0, PORTE10, PORTE9, PORTE7:0, PORTF15:0 have all been tested and validated for both input and output functionalities by connecting each pin to the oscilloscope.

PWM functionality on ports PWMA5:0 and PWMB5:0 were tested and verified with an oscilloscope for a constant 20 kHz frequency operation. With a PWM peripheral clock of 60 MHz, the duty cycle value was adjustable from 0 to 3000, across 0% to 100% duty cycle. Therefore, the precision on the duty cycle is 0.033% when the PWM peripherals are set up for 20 kHz frequency.

The on-chip ADC on MC56F8347 was measured to have a standard deviation of 1 count across 1,000 measurements of a stable potentiometer wiper voltage of 1.645V, and less than 1 count across 1,000 measurements of MMA7260Q accelerometer outputs.

3.3Va - Vgnda1 (V)	3.28			
	ANA0	**ANB5**	**ANB6**	**ANB7**
Signal source	Potentiometer wiper	MMA7260Q X-axis	MMA7260Q Y-axis	MMA7260Q Z-axis
Average count (0 - 4095)	2016	2106	1635	1981
Standard deviation	1	0	0	0
Voltmeter reading (V)	1.645	1.675	1.291	1.575
Measurement discrepancy	1.90%	0.68%	1.40%	0.72%

Table 1: MC56F8347 ADC Performance

The off-chip AD7490 ADC was measured to have a standard deviation of less than 1 count across 1,000 measurements of a stable potentiometer wiper voltage of 2.51V, and also a standard deviation of less than 1 count across 1,000 measurements of a stable ADXRS401 gyroscope output (average value of 1913 counts). No SPI synchronization error was observed.

5V - Vgnda2 (V)	4.99	
	AN0	**AN15**
Signal source	Potentiometer wiper	ADXRS401
CS-to-SCLK sync error	0	0
Average count (0 - 4095)	2058	1913
Standard deviation	0	0
Voltmeter reading (V)	2.51	2.33
Measurement discrepancy	0.11%	0.02%

Table 2: AD7490 ADC Performance

7. ERRATA

Even though there was no single mistake that caused a complete loss of any functionality on the Brain Board, there is one modification that must be made to activate the 3-axis accelerometer functionality on the board. There is one silkscreen typo and a hardwiring error caused by a typo on the manufacture datasheet.

7.1. TOP-SIDE SILKSCREEN TYPO

On the top side, there is a silkscreen for *R24*, although this is really supposed to be *C24*.

7.2. MMA7260 SLEEP PIN ERROR

The Sleep pin on the MMA7260 IC is left open, while it should be connected to logic 1 for correct functionality. This problem can be resolved by shorting pin 12 to pin 2 or 3. Optionally, the Sleep pin could be tied to one of the unshared GPIO pins to make the sleep feature controllable by software.

7.3. MMA7260Q G-SELECT ERROR

g-Select1 and g-Select2 pins of Freescale MMA7260Q IC are hardwired to logic 0 and 1, respectively, for ±2g acceleration range. However Freescale's MMA7260Q datasheet, revision 2.0 (2/2006), has an error in the g-select table, and this hardwire configuration sets the IC in ±4g mode. I pointed this error after testing, and this typo has been corrected in datasheet 3.0 (4/2006). In the future revisions of the Brain Board it may be desirable to tie g-Select pins to two of the unshared GPIO pins to make the acceleration range software-selectable.

8. ACKNOWLEDGEMENTS

I would like to first thank Professor Andy Ruina of Theoretical and Applied Mechanics for granting a green light on the Brain Board development, and for trusting me with a rather high-risk PCB design. This project would not have been alive if it were not for his approval.

Secondly, I would like to thank Jason Cortell, the Biorobotics Lab Manager, for his priceless support throughout the design specification, parts research, and board development process. I owe much to Jason for the success and timely completion of this project.

I would like to acknowledge all the corporate sponsors who provided the component samples for this project, including Freescale Semiconductor; Analog Devices, Inc.; Central Semiconductor; and Kionix, Inc. My special thanks goes to Kionix, Inc. for their generous gifts and extensive support.

Lastly, I would like to thank my teammates Andre Harrison, Mike Tosto, Matt Haberland, Sarah Bates, Greg Stiesberg, and Justin Webb, for pointing out the shortcomings in Innovation First *Robot Controller* system. Their inputs were important to avoid repeating the past mistakes in the Brain Board design.

APPENDIX A: H-BRIDGE BOARD SCHEMATIC

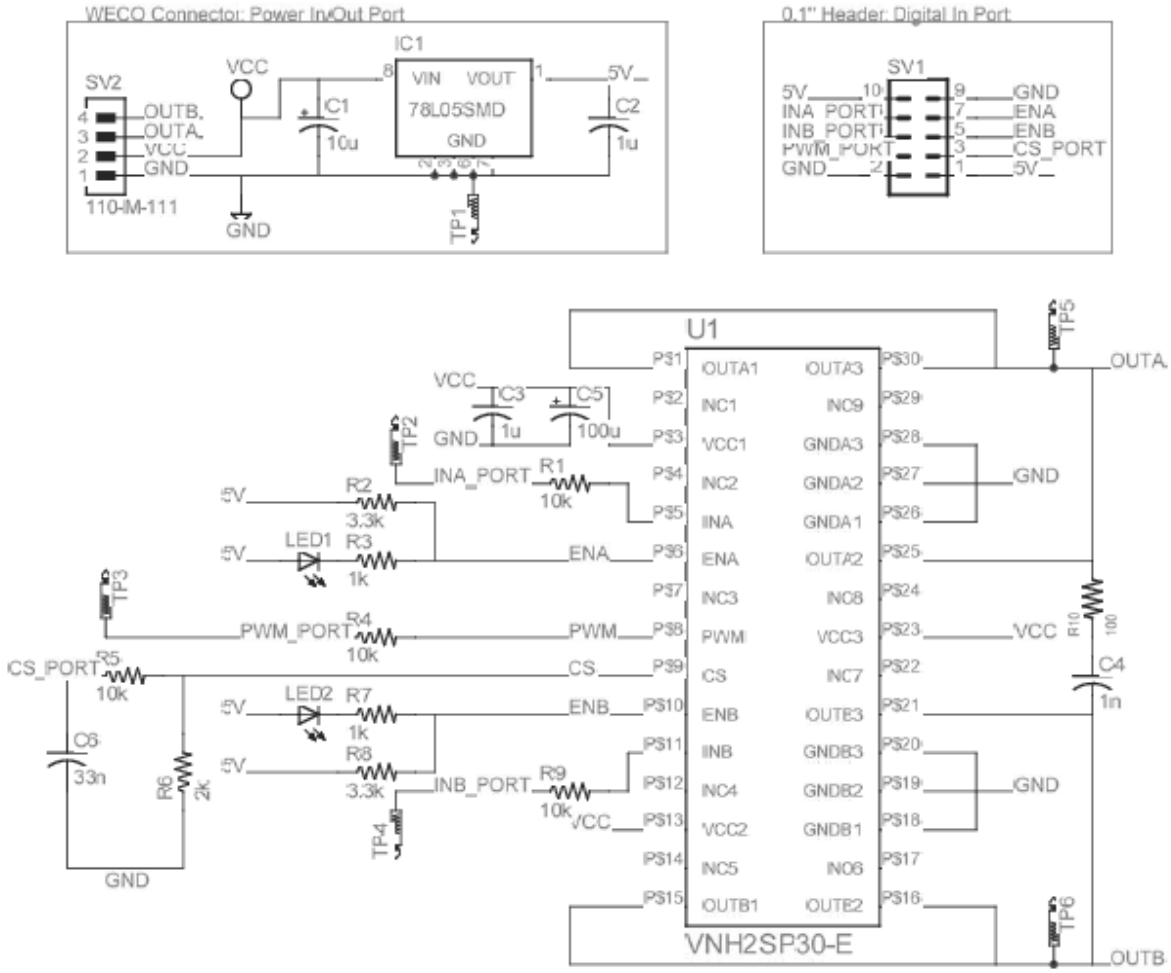

Figure 15: H-Bridge Board Schematic

APPENDIX B: H-BRIDGE BOARD LAYOUT

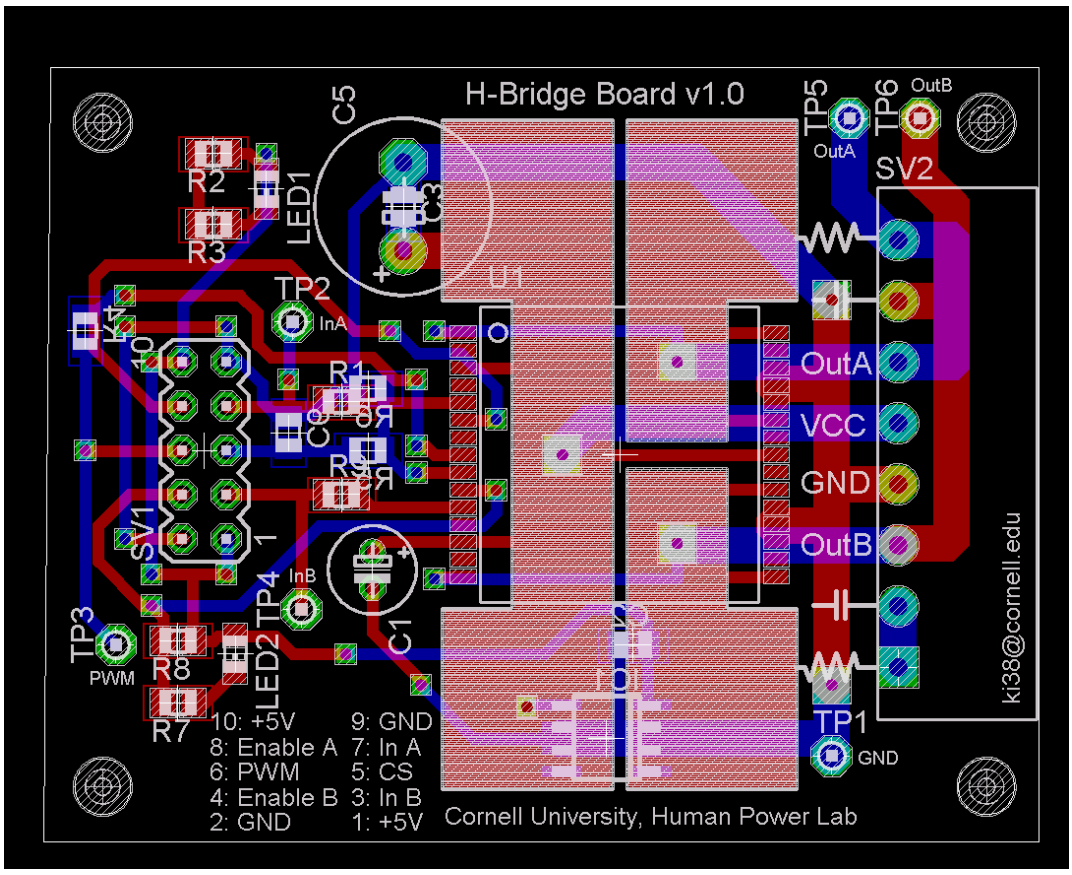

Figure 16: H-Bridge Board Layout

APPENDIX C: BRAIN BOARD SCHEMATICS

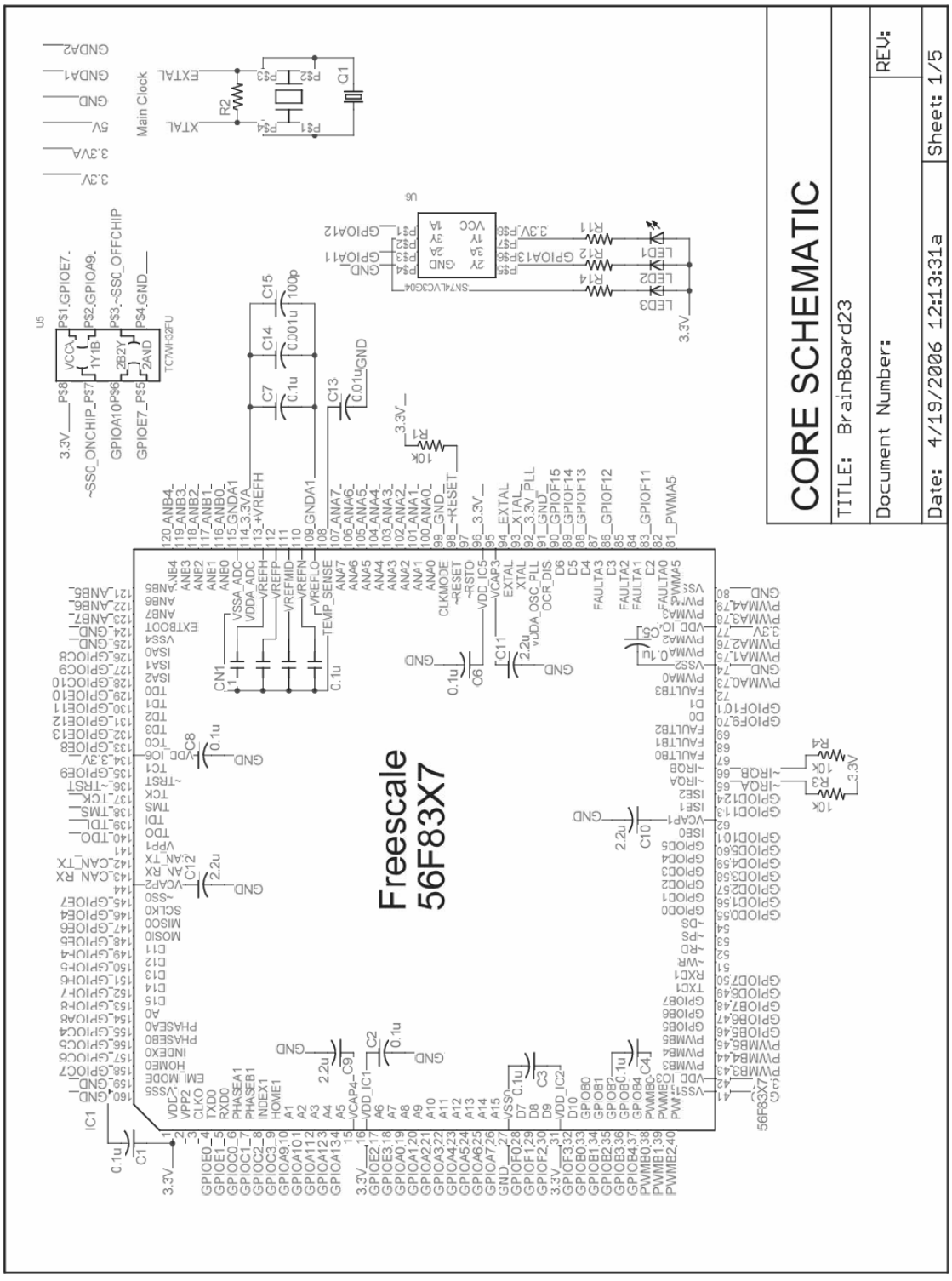

Figure 17: Core Schematic

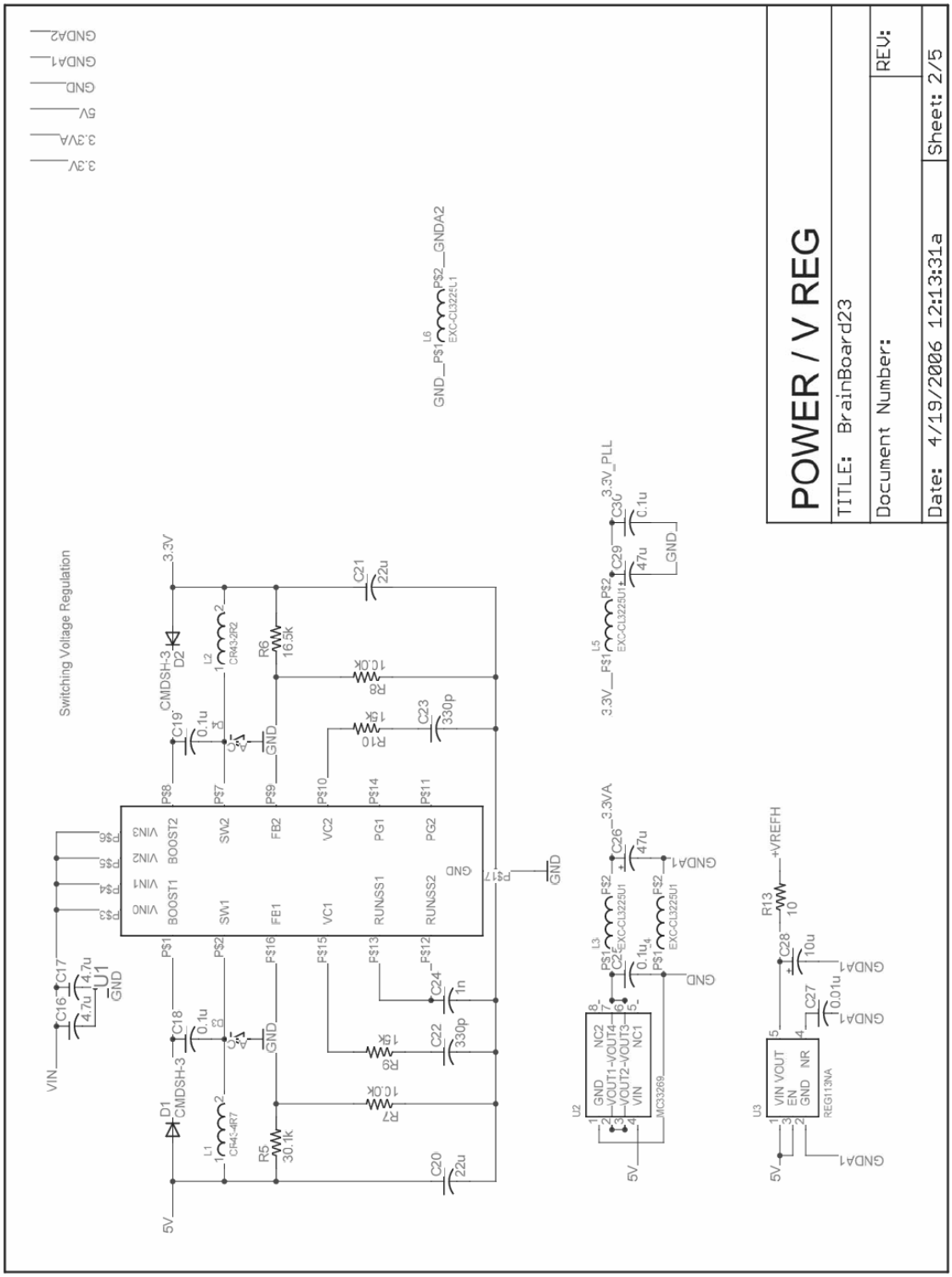

Figure 18: Voltage Regulator Schematic

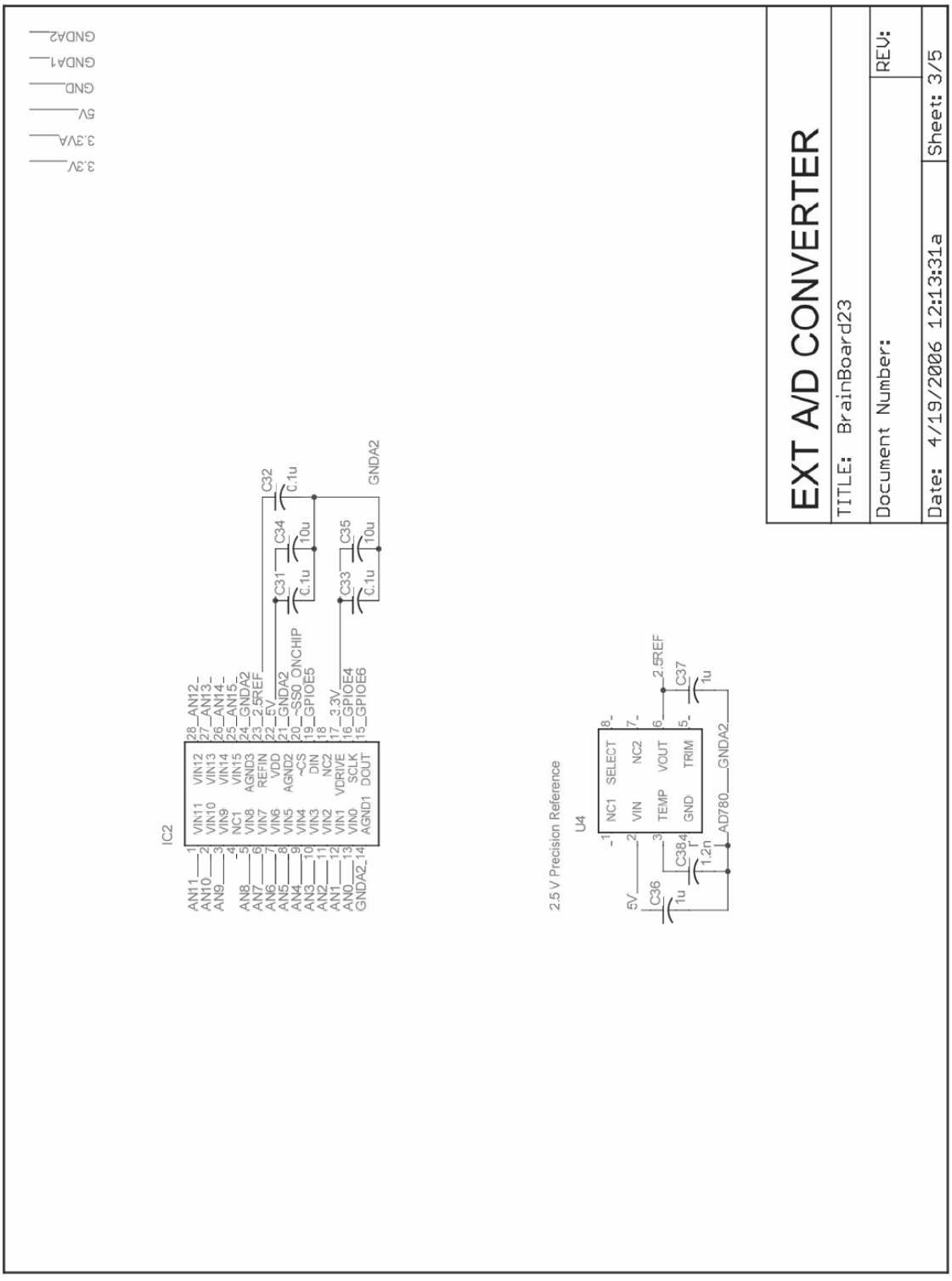

Figure 19: External ADC Schematic

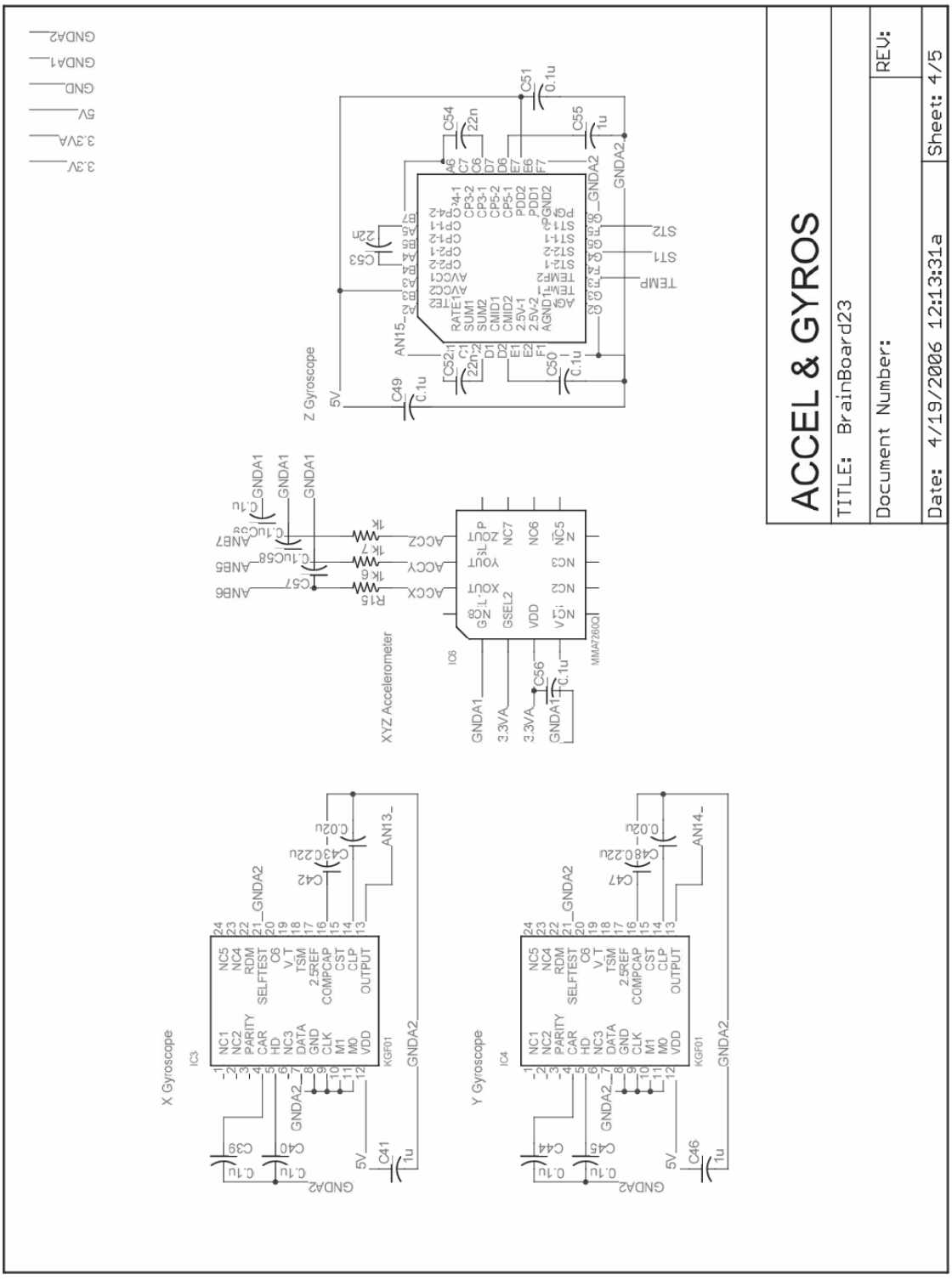

Figure 20: Accelerometer/Gyroscope Schematic

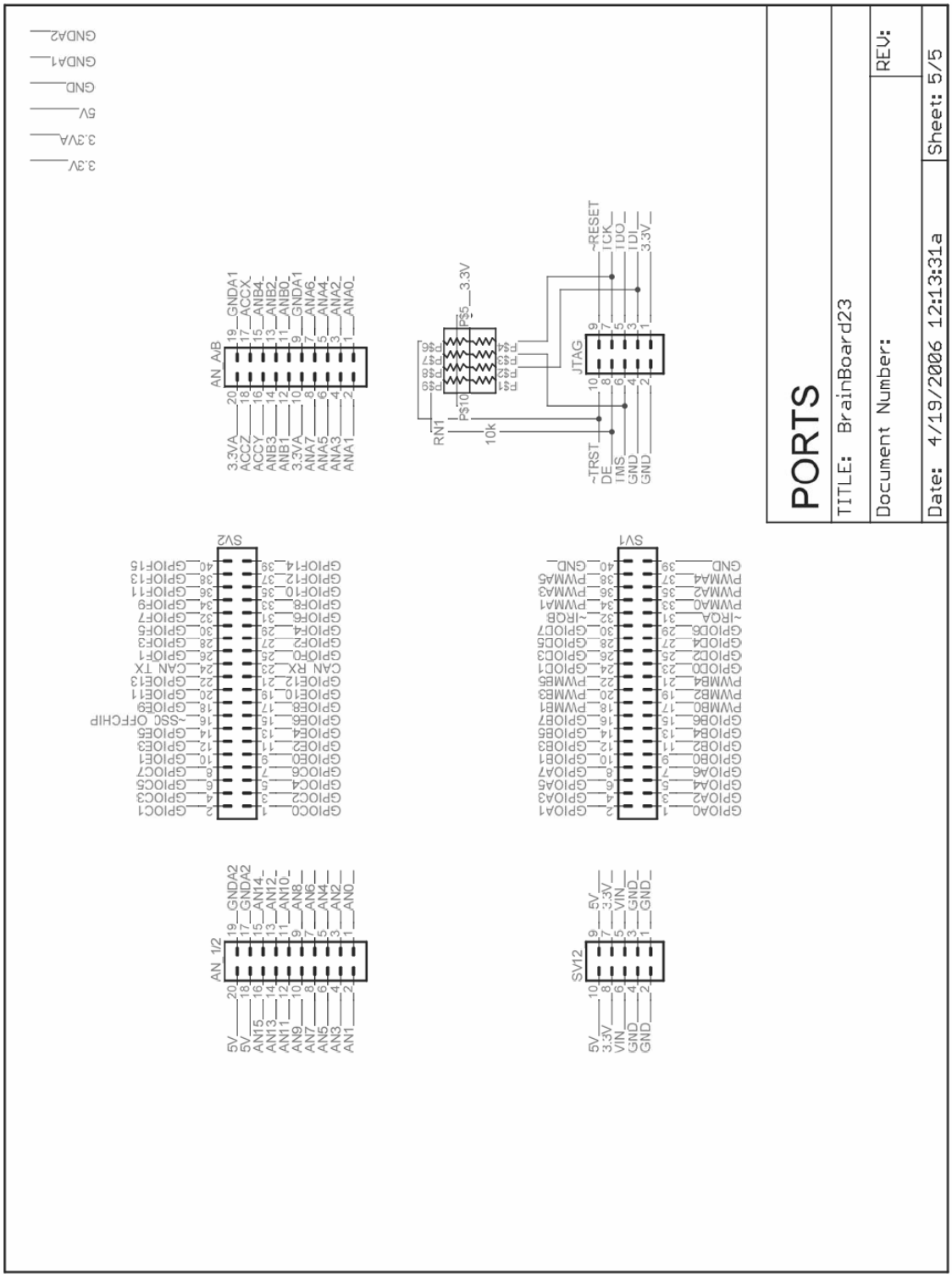

Figure 21: Port Schematic

APPENDIX D: BRAIN BOARD LAYOUT

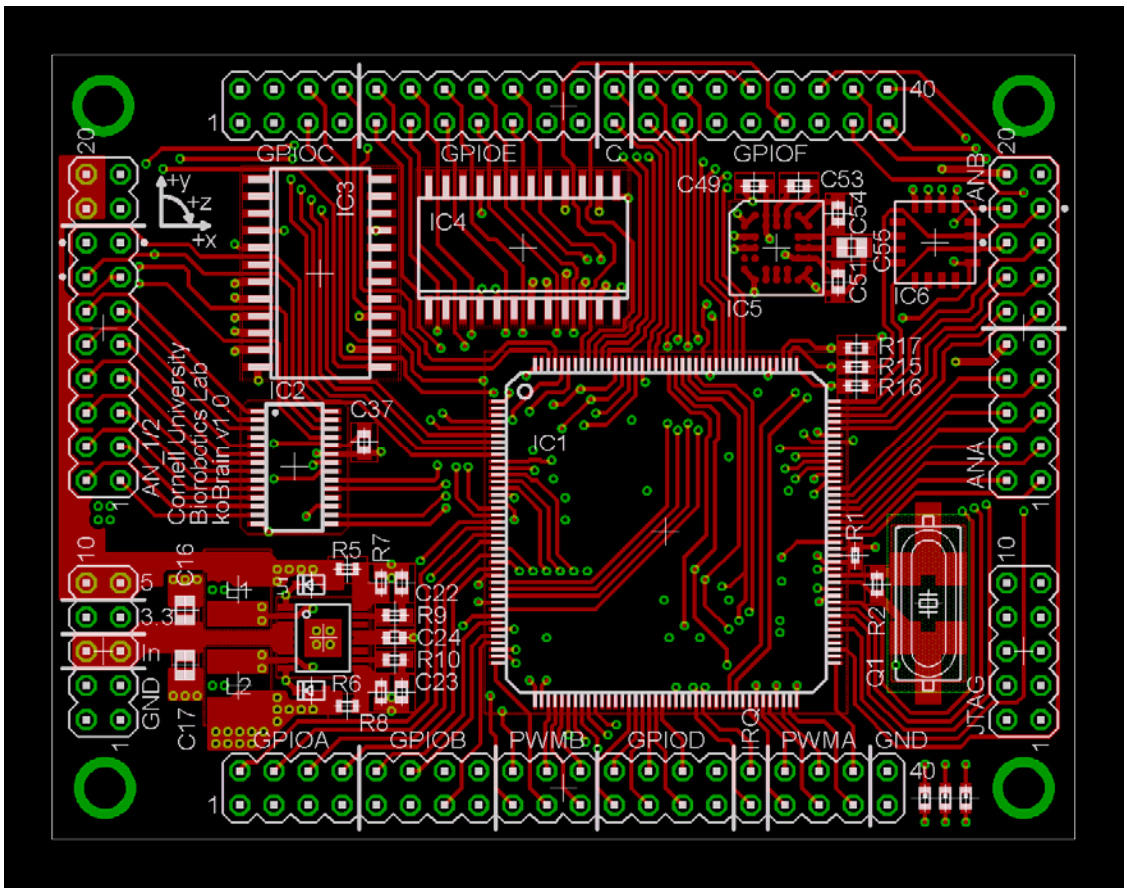

Figure 22: Microcontroller Board Layout (Top)

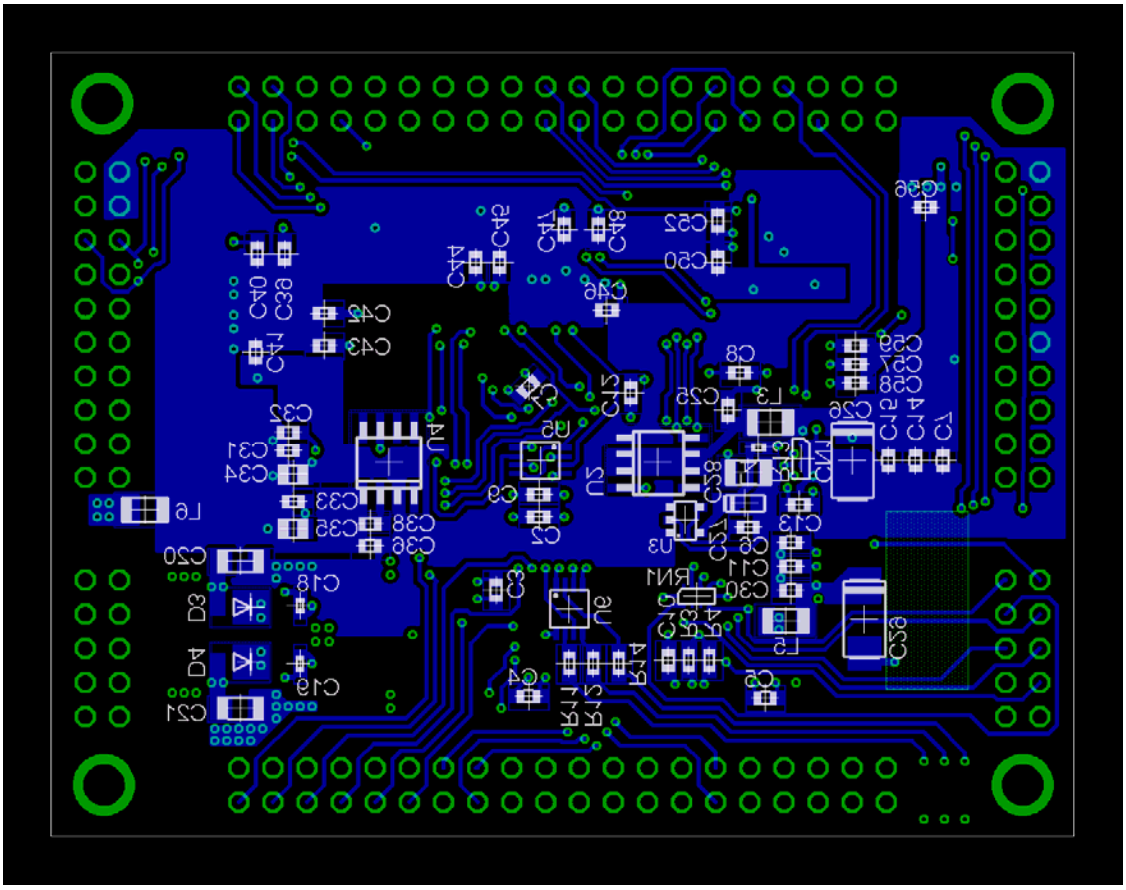

Figure 23: Microcontroller Board Layout (Bottom)

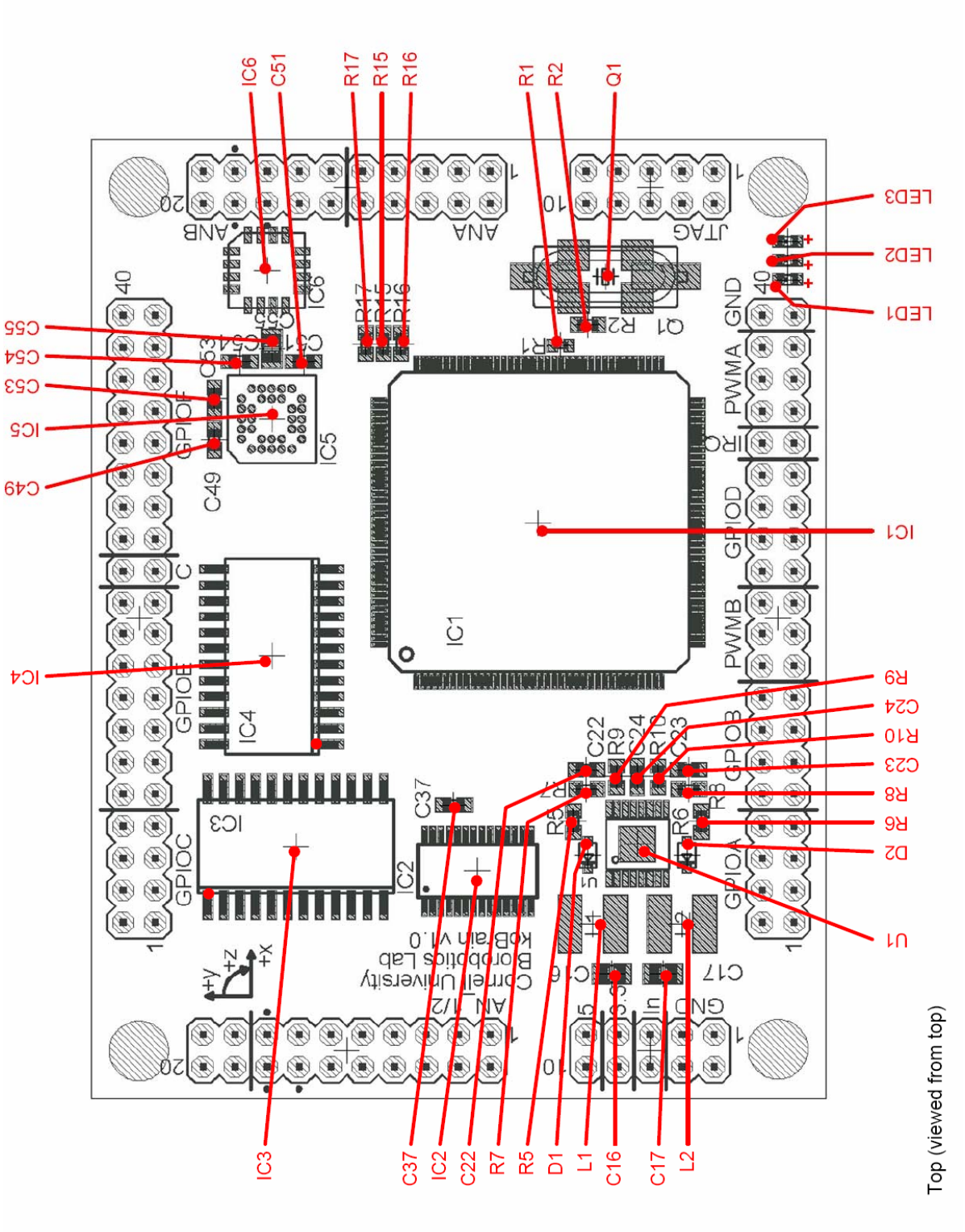

Figure 24: Component Placement (Top)

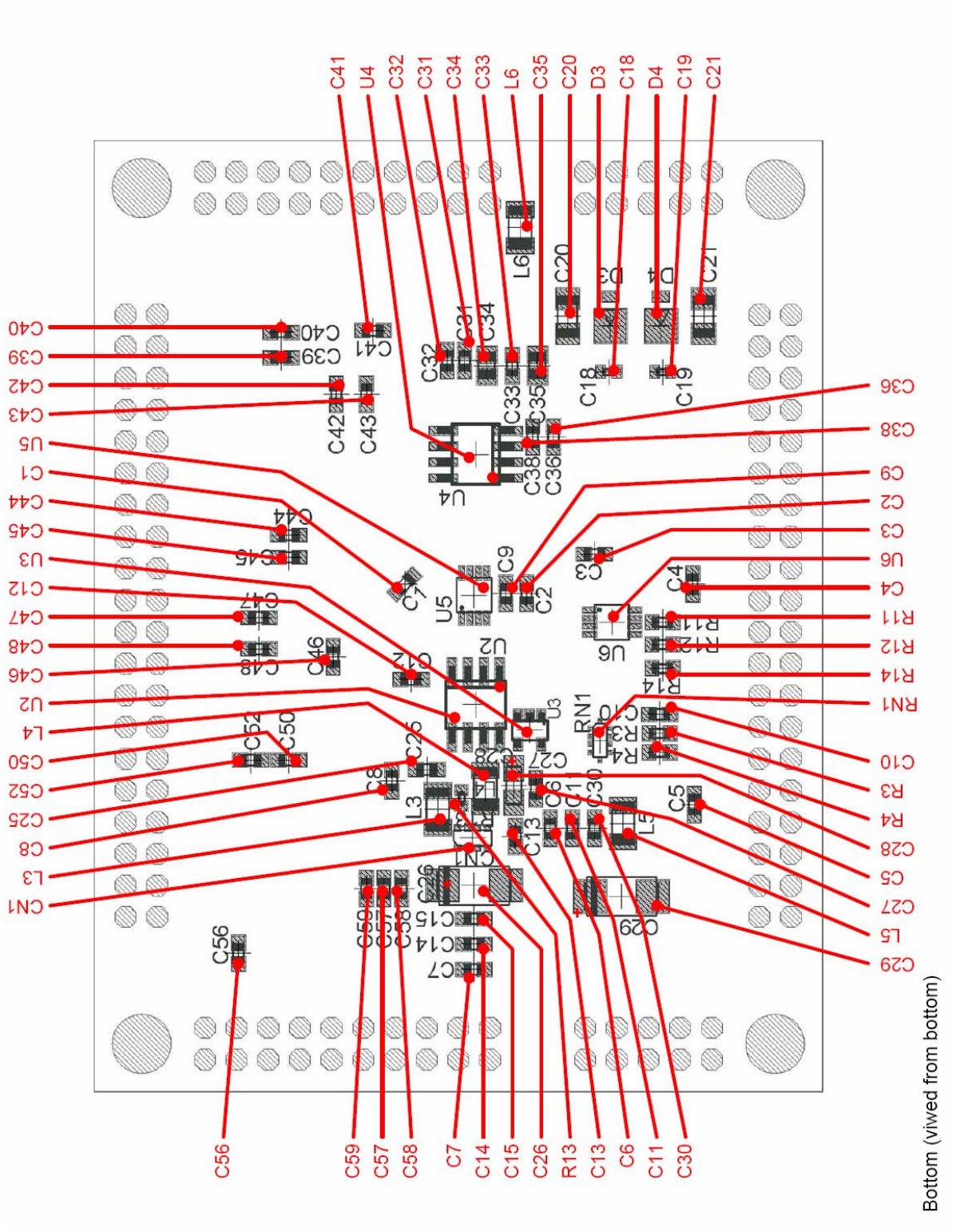

Figure 25: Component Placement (Bottom)

APPENDIX E: BRAIN BOARD BILL OF MATERIALS

Quantity	Number	Type	Digikey part number
26	C1-8, C18-19, C25, C30-33, C39-40, C44-45, C49-51, C56-59	Ceramic 0.1uF cap (0603, X7R)	PCC2398CT-ND
4	C9-12	Ceramic 2.2uF cap (0603, X5R)	PCC2397CT-ND
2	C13, C27	Ceramic 0.01uF cap (0603, X7R)	PCC1784CT-ND
2	C14, C24	Ceramic 1nF cap (0603, X7R)	PCC1772CT-ND
1	C15	Ceramic 0.1nF cap (0603, NPO)	PCC101ACVCT-ND
2	C16-17	Ceramic 4.7uF cap (0805, X5R)	PCC2321CT-ND
2	C20-21	Ceramic 22uF cap (1206, X5R)	PCC2332CT-ND
2	C22-23	Ceramic 330pF cap (0603, X7R)	PCC1949CT-ND
2	C34-35	Ceramic 10uF cap (0805, X5R)	PCC2403CT-ND
4	C36-37, C41, C46	Ceramic 1uF cap (0603, X5R)	PCC2224CT-ND
1	C38	Ceramic 1.2nF cap (0603, X7R)	PCC1773CT-ND
2	C42, C47	Ceramic 0.22uF cap (0603, X5R)	PCC1749CT-ND
5	C43, C48, C52-54	Ceramic 22nF cap (0603, X7R)	PCC1767CT-ND
1	C55	Ceramic 1uF cap (0805, X5R)	PCC2319CT-ND
1	C28	Tantalum 10uF cap (EIA A)	493-2351-1-ND
1	C26, C29	Tantalum 47uF cap (EIA C)	493-2362-1-ND
1	CN1	Ceramic 0.1uF cap network (1206)	P11148CT-ND
1	R1	Thick film 10k res (0402)	P10.0KLCT-ND
1	R2	Thick film 1M res (0603)	P1.00MHCT-ND
4	R3-4, R7-8	Thick film 10k res (0603)	P10.0KHCT-ND
1	R5	Thick film 30.1k res (0603)	P30.1KHCT-ND
1	R6	Thick film 16.5k res (0603)	P16.5KHCT-ND
2	R9-10	Thick film 15k res (0603)	P15.0KHCT-ND
1	R13	Thick film 10 res (0402)	P10.0LCT-ND
3	R11-12, R14	Thick film 330 res (0603)	P332HCT-ND
3	R15-17	Thick film 1k res (0603)	P1.00KHCT-ND
1	RN1	10k res network (EXB-D)	U9103CT-ND
1	L1	Sumida CR43-4R7 inductor	308-1123-1-ND
1	L2	Sumida CR43-2R2 inductor	308-1119-1-ND
3	L3-6	Panasonic EXC-CL3225U1	P9811CT-ND
2	D3, D4	UPS140 diode	UPS140E3CT-ND
2	D1, D2	CMDSH-3 diode	
1	Q1	8MHz crystal (CS20)	300-8126-1-ND
1	IC1	Freescale 56F8347 LQFP-160	
1	IC2	Analog Devices AD7490 (TSSOP28)	AD7490BRUZ-ND
2	IC3-4	Kionix KGF01 SOIC-24	DO NOT POPULATE
1	IC5	Analog Devices ADXRS401 BGA-32	ADXRS401ABG-ND
1	IC6	Freescale MMA7260Q QFN-16	MMA7260Q-ND
1	U1	Linear Technology LT1940 (TSSOP-16)	
1	U2	ON Semiconductor MC33269 (3.3V, SOIC8)	MC33269D-3.3OS-ND
1	U3	Burr-Brown REG113NA (3.3V, SOT)	REG113NA-3.3/3KCT-ND
1	U4	Analog Devices AD780 (SOIC8)	AD780BR-ND
1	U5	Dual OR gate (SSOP-8)	TC7WH32FUTCT-ND
1	U6	Triple inverter (SSOP8)	296-13278-1-ND
1	LED1	0603 LED (green)	160-1443-1-ND
1	LED2	0603 LED (yellow)	160-1449-1-ND
1	LED3	0603 LED (red)	160-1442-1-ND

Table 3: Microcontroller Board Bill of Materials

APPENDIX F: MC56F8347 PORT FUNCTIONALITIES

Group	Port	Header	Alternate functionality	Notes
GPIOA	GPIOA0	1	A8	**Address Bus** - A8 - A15 specify eight of the address lines for external program or data memory accesses. **Port A GPIO** - these eight GPIO pins can be individually programmed as input or output pins. After reset, the default state is Address Bus. To deactivate the internal pull-up resistor, clear the appropriate GPIO bit in the GPIOA_PUR register. GPIOA9 - active-low enable to gate GPIOE7 output to Pin 8 GPIOA10 - active-low enable to gate GPIOE7 output to ADC's CS pin GPIOA11 - tied to the yellow LED GPIOA12 - tied to the green LED GPIOA13 - tied to the red LED
	GPIOA1	2	A9	
	GPIOA2	3	A10	
	GPIOA3	4	A11	
	GPIOA4	5	A12	
	GPIOA5	6	A13	
	GPIOA6	7	A14	
	GPIOA7	8	A15	

Table 4: Header Description (PORTA Group)

Group	Port	Header	Alternate functionality	Notes
GPIOB	GPIOB0	9	A16	**PORT B GPIO** - these four GPIO pins can be programmed as input or output pins. After reset, the startup state of GPIOB0 - GPIOB3 is determined as a function of EXTBOOT, EMI_MODE and the Flash security setting. To deactivate the internal pull-up resistor, clear the appropriate GPIO bit in the GPIOB_PUR register. **Address Bus** - A16 - A19 specify one of the address lines for external program or data memory accesses. **Clock Outputs** - can be used to monitor the prescaler_clock, SYS_CLK, SYS_CLK2 or oscillator_clock on GPIOB4 through GPIOB7, respectively.
	GPIOB1	10	A17	
	GPIOB2	11	A18	
	GPIOB3	12	A19	
	GPIOB4	13	A20, prescaler_clock	
	GPIOB5	14	A21, SYS_CLK	
	GPIOB6	15	A22, SYS_CLK2	
	GPIOB7	16	A23, oscillator_clock	

Table 5: Header Description (PORTB Group)

Group	Port	Header	Alternate functionality	Notes
GPIOC	GPIOC0	1	PHASEA1, TB0, SCLK1	**PORT C GPIO** - these eight GPIO pins can be programmed as input or output pins. **PHASEA0, PHASEB0** - quadrature decoder 0, phase A and B (respectively) inputs for decoder 0. **INDEX0** - quadrature decoder 0, INDEX input. **HOME0** - quadrature decoder 0, HOME input. **PHASEA1, PHASEB1** - quadrature decoder 1, phase A and B (respectively) inputs for decoder 1. **INDEX1** - quadrature decoder 1, INDEX input. **HOME1** - quadrature decoder 1, HOME input. **TA0, TA1, TA2, TA3** - timer A: channels 0, 1, 2, and 3, respectively. **TB0, TB1** - timer B: channels 0 and 1, respectively. **SCLK1, MISO1, MOSI1, ~SS1** - SPI 1 serial clock, MISO, MOSI, and slave select, respectively.
	GPIOC1	2	PHASEB1, TB1, MOSI1	
	GPIOC2	3	INDEX1, TB2, MISO1	
	GPIOC3	4	HOME1, TB3, ~SS1	
	GPIOC4	5	PHASEA0, TA0	
	GPIOC5	6	PHASEB0, TA1	
	GPIOC6	7	INDEX0, TA2	
	GPIOC7	8	HOME0, TA3	

Table 6: Header Description (PORTC Group)

Group	Port	Header	Alternate functionality	Notes
GPIOD	GPIOD0	1	~CS2	**PORT D GPIO** - these eight GPIO pins can be programmed as input or output pins. **Chip Select** - ~CS2 - ~CS7 may be programmed within the EMI module to act as chip selects for specific areas fo the external memory map. **TXD1** - SCI1 trasmit data output. **RXD1** - SCI1 receive data input.
	GPIOD1	2	~CS3	
	GPIOD2	3	~CS4	
	GPIOD3	4	~CS5	
	GPIOD4	5	~CS6	
	GPIOD5	6	~CS7	
	GPIOD6	7	TXD1	
	GPIOD7	8	RXD1	

Table 7: Header Description (PORTD Group)

Group	Port	Header	Alternate functionality	Notes
GPIOE	GPIOE0	9	TXD0	**Address Bus** - A6 - A7 specify eight of the address lines for external program or data memory accesses. **Port E GPIO** - these fourteen GPIO pins can be individually programmed as input or output pins. **SCLK0, MISO0, MOSI0, ~SS0** - SPI 0 serial clock, MISO, MOSI, and slave select, respectively. **TC0, TC1** - timer C: channels 0, and 1, respectively. **TD0, TD1, TD2, TD3** - timer D: channels 0, 1, 2, and 3, respectively. **TXD1** - SCI1 trasmit data output. **RXD1** - SCI1 receive data input.
	GPIOE1	10	RXD0	
	GPIOE2	11	A6 (address bus)	
	GPIOE3	12	A7 (address bus)	
	GPIOE4	13	SCLK0	
	GPIOE5	14	MOSI0	
	GPIOE6	15	MISO0	
	GPIOE7	16	~SS0	
	GPIOE8	17	TC0	
	GPIOE9	18	TC1	
	GPIOE10	19	TD0	
	GPIOE11	20	TD1	
	GPIOE12	21	TD2	
	GPIOE13	22	TD3	

Table 8: Header Description (PORTE Group)

Group	Port	Header	Alternate functionality	Notes
GPIOF	GPIOF0	25	D7 (data bus)	**Data Bus** - D0 - D15 specify part of the data for external program or data memory accesses. **Port F GPIO** - these sixteen GPIO pins can be individually programmed as input or output pins. After reset, the default state is Data Bus. To deactivate the internal pull-up resistor, clear the appropriate GPIO bit in the GPIOF_PUR register.
	GPIOF1	26	D8 (data bus)	
	GPIOF2	27	D9 (data bus)	
	GPIOF3	28	D10 (data bus)	
	GPIOF4	29	D11 (data bus)	
	GPIOF5	30	D12 (data bus)	
	GPIOF6	31	D13 (data bus)	
	GPIOF7	32	D14 (data bus)	
	GPIOF8	33	D15 (data bus)	
	GPIOF9	34	D0 (data bus)	
	GPIOF10	35	D1 (data bus)	
	GPIOF11	36	D2 (data bus)	
	GPIOF12	37	D3 (data bus)	
	GPIOF13	38	D4 (data bus)	
	GPIOF14	39	D5 (data bus)	
	GPIOF15	40	D6 (data bus)	

Table 9: Header Description (PORTF Group)

Group	Port	Header	Notes
PWMA	PWMA0	33	**PWMA0-5** - six PWMA output pins.
	PWMA1	34	
	PWMA2	35	
	PWMA3	36	
	PWMA4	37	
	PWMA5	38	

Table 10: Header Description (PWMA Group)

Group	Port	Header	Notes
PWMB	PWMB0	17	**PWMB0-5** - six PWMB output pins.
	PWMB1	18	
	PWMB2	19	
	PWMB3	20	
	PWMB4	21	
	PWMB5	22	

Table 11: Header Description (PWMB Group)

Group	Port	Header	Notes
ANA	ANA0	1	**ANA0 - 3** - analog inputs to ADC A, channel 0.
	ANA1	2	**ANA4 - 7** - analog inputs to ADC A, channel 1.
	ANA2	3	Inputs should be between 0-3.3V
	ANA3	4	
	ANA4	5	
	ANA5	6	
	ANA6	7	
	ANA7	8	
	GNDA1	9	Analog ground
	3.3VA	10	Analog 3.3V

Table 12: Header Description (ANA Group)

Group	Port	Header	Notes
ANB	ANB0	11	**ANB0 - 3** - analog inputs to ADC B, channel 0.
	ANB1	12	**ANB4 - 7** - analog inputs to ADC B, channel 1.
	ANB2	13	Inputs should be between 0-3.3V
	ANB3	14	ANB5 - ANB7 inputs are filtered through RC circuit.
	ANB4	15	ANB5 - tied to accelerometer output in X direction.
	ANB5	16	ANB6 - tied to accelerometer output in Y direction.
	ANB6	17	ANB7 - tied to accelerometer output in Z direction.
	ANB7	18	
	GNDA1	19	Analog ground
	3.3VA	20	Analog 3.3V

Table 13: Header Description (ANB Group)

Group	Port	Header	Notes
AN_1/2	AN0	1	**AN0 - 15** - analog inputs to Analog Devices AD7490 (IC2). Inputs should be between 0-5V. AN13 is tied to gyroscope output in X direction. AN14 is tied to gyroscope output in Y direction. AN15 is tied to gyroscope output in X direction.
	AN1	2	
	AN2	3	
	AN3	4	
	AN4	5	
	AN5	6	
	AN6	7	
	AN7	8	
	AN8	9	
	AN9	10	
	AN10	11	
	AN11	12	
	AN12	13	
	AN13	14	
	AN14	15	
	AN15	16	
	GNDA2	17	Analog ground
	5V	18	5V reference
	GNDA2	19	Analog ground
	5V	20	5V reference

Table 14: Header Description (AN_1/2 Group)

Group	Port	Header	Notes
CAN	CAN_RX	23	**FlexCAN Receive Data** - this is the CAN input. This pin has an internal pull-up resistor. To deactivate the internal pull-up resistor, set the can bit in the SIM_PUDR register.
	CAN_TX	24	**FlexCAN Receive Data** - CAN output

Table 15: Header Description (CAN Group)

Group	Port	Header	Notes
IRQ	~IRQA	31	**External Interrupt Request A and B** - The ~IRQA and ~IRQB inputs are asynchronous external interrupt requests during Stop and Wait mode operation. During other operating modes, they are synchronized external interrupt requests, which indicate an external device is requesting service. They can be programmed to be level-sensitive or negative-edge triggered.
	~IRQB	32	

Table 16: Header Description (IRQ Group)

Group	Port	Header	Notes
JTAG	3.3V	1	3.3V digital reference
	GND	2	Digital ground
	TDI	3	**Test Data Input** - this input pin provides a serial input data stream to the JTAG/EOnE port. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor. This pin is connected to an external 10k pull-up resistor.
	GND	4	Digital ground
	TDO	5	**Test Data Output** - this tri-stateable output pin provides a serial output data stream from the JTAG/EOnCE port. It is driven in the shift-IR and shift-DR controller states, and changes on the falling edge of TCK. This pin is connected to an external 10k pull-up resistor.
	TMS	6	**Test Mode Select Input** - this input pin is used to sequence the JTAG TAP controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor. This pin is connected to an external 10k pull-up resistor.
	TCK	7	**Test Clock Input** - this input pin provides a gated clock to synchronize the test logic and shift serial data to the JTAG/EOnCE port. The pin is connected internally to a pull-down resistor. This pin is connected to an external 10k pull-up resistor.
	DE	8	**Debug Event** - this is an active-low debug event signal, connected to an external 10k resistor
	~RESET	9	**Reset** - this input is a direct hardware reset on the processor. When ~RESET is asserted low, the device is initialized and placed in the reset state. A schmitt trigger input is used for noise immunity. When the ~RESET pin is deasserted, the initial chip operating mode is latched form the EXTBOOT pin. The internal reset signal will be deasserted synchronous with the internal clocks after a fixed number of internal clocks. This pin is internally pulled-up.
	~TRST	10	**Test Reset** - as an input, a low signal on this pin provides a reset signal to the JTAG TAP controller. To ensure complete hardware reset, ~TRST should be asserted whenever ~RESET is asserted. The only exception occurs in a debugging environment when a hardware device reset is required and the JTAG/EOnCE module must not be reset. This pin is internally pulled up.

Table 17: Header Description (JTAG Group)

Group	Port	Header	Notes
Power	GND	1	Digital ground reference
	GND	2	Digital ground reference
	GND	3	Digital ground reference
	GND	4	Digital ground reference
	Vin	5	Power supply input +
	Vin	6	Power supply input +
	3.3V	7	3.3V output
	3.3V	8	3.3V output
	5V	9	5V output
	5V	10	5V output

Table 18: Header Description (Power Group)

APPENDIX G: H-BRIDGE BOARD TEST PROGRAM

Robo.c

```
/** ###################################################################
**     Filename  : Robo.C
**     Project   : Robo
**     Processor : 56F805
**     Version   : Driver 01.07
**     Compiler  : Metrowerks DSP C Compiler
**     Date/Time : 10/23/2005, 10:44 PM
**     Abstract  :
**         Main module.
**         Here is to be placed user's code.
**     Settings  :
**     Contents  :
**         No public methods
**
**     (c) Copyright UNIS, spol. s r.o. 1997-2004
**     UNIS, spol. s r.o.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
** ###################################################################*/
/* MODULE Robo */

/* Including used modules for compilling procedure */
#include "Cpu.h"
#include "GPIO_A.h"
#include "GPIO_B.h"
#include "GPIO_E.h"
#include "PWM1.h"
#include "AS1.h"
#include "FC1.h"
#include "QD1.h"

/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

#include <stdlib.h>           // Custom addition: standard library
#include <stdio.h>            // Custom addition: standard i/o
#include <math.h>             // Custom addition: mathematic protocols
#include <string.h>           // Custom addition: handles strings
#include <float.h>            // Custom addition: floating point

#include "CustomFunctions.h"  // Custom functions library

void main(void)
{
  int i;
  char strTarget[6], strPosition[6], strDuty[6];

  // PID variables
  long  target  = 0;
  long  err[]   = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  long  dErr    = 0;
  long  iErr    = 0;
  float Kp = 0.2;
  float Ki = 0.01;
  float Kd = 5;

  // Quadrature decoder variables (module QD1)
  TStateValues QDcounter;
  long  position       = 0;    // Position (from quadrature decoder QD1)
  long  prevPosition= 0;       // Position in the previous program loop
  int   revolution     = 0;    // Number of revolution
  int   prevRevolution = 0;    // Number of revolution in the prev program loop
  int   posPerRev  = 2048;     // 2048 positions in 1 revolution
```

```
  // PWM-related variables (module PWM1)
  float dutyPercent = 0;       // PWM duty cycle (in %)

  // Time-related variables (module FC1, free counter)
  word  prgmLoopUS     = 0;    // Microseconds to complete a single program loop
  word  tArithmeticEnd;        // Time marker (in us) at end of arithmetic operation
  word  tSerialEnd;            // Time marker (in us) at end of serial communication
  word  tenthUS = 0;           // 1/10 microseconds in the free counter
  int   countMS = 0;           // Counts the number of milliseconds
  int   period  = 1000;        // Number of milliseconds in one "cycle"

  // Mathematical constants
  const double PI = 3.141592653;

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/


  /* Write your code here */

  FC1_Reset();                 // Reset the free counter
  QD1_SetPosition(0);          // Reset QD counter
  QD1_SetRevolution(0);        // Reset QD counter
  GPIO_A_ClrBit(0);            // Initialize H-bridge polarity
  GPIO_A_SetBit(1);

  while(TRUE)
  {
        FC1_GetCounterValue(&tenthUS);
        tArithmeticStart = tenthUS/10;

        // STEP 1: calculate new target
        target = Target_Step(countMS, period);
        //target = Target_ZigZag(countMS, period);
        //target = Target_Sine(countMS, period);

        // STEP 2: obtain latest position feedback
        for(i=10; i>0; i--)
        {
              err[i]=err[i-1];                                          // Shift the err
history by 1
        }
        prevPosition = position;             // Save previous position
        prevRevolution = revolution;         // Save previous revolution

        QD1_GetCounters(&QDcounter);                 // Save latest QD info into QDcounter
        position = (long)((&QDcounter)->Position);   // Extract new position
        revolution = (int)(position/posPerRev);      // Calculate new revolution

        err[0] = target - position;          // Calculate new error
        dErr = err[0]-err[1];                // Calculate d(err) for derivative
        iErr = err[0];
        for(i=1; i < 10; i++)
        {
              iErr += err[i];                // Calculate i(err) for integral
        }

        // STEP 3: calculate new duty cycle (in %)
        dutyPercent = (float)(Kp*err[0] + Kd*dErr + Ki*iErr);
        SetDutyPercent(fabsf(dutyPercent));
        if(dutyPercent < 0)           // For negative duty cycle, reverse polarity
        {
              GPIO_A_SetBit(0);              // GPIOA[1:0] = 2b'01 for negative polarity
              GPIO_A_ClrBit(1);
        }
        else
        {
              GPIO_A_ClrBit(0);              // GPIOA[1:0] = 2b'10 for positive polarity
              GPIO_A_SetBit(1);
        }

        FC1_GetCounterValue(&tenthUS);
        tArithmeticEnd = tenthUS/10;  // Time marker at end of arithmetic operation
```

```
        // STEP 4: parse number into character arrays and transmit over serial port
        ParseLong(strTarget, target);
        Print(strTarget);
        Print("\t");
        ParseLong(strPosition, position);
        Print(strPosition);
        Print("\r\n");

        if (countMS<period)   countMS++;     // If countMS hasn't reached period, add 1
        else                  countMS=0;     // If countMS has reached period, reset it

        FC1_GetCounterValue(&tenthUS);       // Get the number of 0.1 microseconds in counter
        prgmLoopUS = tenthUS/10;             // How many microsecounds did the prgm loop take?
        while (tenthUS <= 10000)             // Has 1 ms passed in free counter?
        {
               FC1_GetCounterValue(&tenthUS);// No, get counter value again and keep waiting
        }
        FC1_Reset();
        // Yes, exit the loop and reset the counter
  }

  // END MY CODE

}

/* END Robo */
/*
** ###################################################################
**
**     This file was created by UNIS Processor Expert 2.95 [03.58]
**     for the Freescale 56800 series of microcontrollers.
**
** ###################################################################
*/
```

CustomFunctions.h

```
// Prototypes
int     ParseLong(char *s, long number);
void Print(char *);
long Target_Step(int countMS, int period);
long Target_ZigZag(int countMS, int period);
//long Target_Sine(int countMS, int period);
void SetDutyPercent(float percent);

// Custom Functions

/** ###################################################################
**      Function       : ParseLong
**      Input          : char *s
**                           long number
**      Output         : integer 0
**
**      ParseLong takes a long and converts it into a 6-byte character
**      string. The formatting is as follows:
**      " #####" for positive number, and
**      "-#####" for negative number.
** ###################################################################*/
int     ParseLong(char *s, long number)
{
        int absNum = abs(number);     // Absolute value of the argument long
        int digit;
        int div = 10000;
        int i;

        if(number < 0)        // If number is negative number,
        {                             //  insert a '-' sign
               s[0] = '-';
        }
        else                          // Else,
        {                             //  insert a blankspace ' ' character
               s[0] = ' ';
        }
        for(i=0; i<5; i++)
```

```
	{
		digit = (absNum/div)%10;
		switch(digit)
		{
			case 1:
				s[i+1] ='1';
				break;
			case 2:
				s[i+1] ='2';
				break;
			case 3:
				s[i+1] ='3';
				break;
			case 4:
				s[i+1] ='4';
				break;
			case 5:
				s[i+1] ='5';
				break;
			case 6:
				s[i+1] ='6';
				break;
			case 7:
				s[i+1] ='7';
				break;
			case 8:
				s[i+1] ='8';
				break;
			case 9:
				s[i+1] ='9';
				break;
			case 0:
				s[i+1] ='0';
				break;
			default:
				s[i+1] = ' ';
				break;
		}
		div = div/10;
	}
	s[6] = '\0';		// Insert a void character
	return 0;
}

/** ###################################################################
**	Function	: Print
**	Input		: char String[]
**
**	Print takes a character string, and sends it out via
**	asynchronous transmitter AS1. Use this funciton in this manner:
**	Print("Hello World!"); or
**	Print(message);			where message is a character array
** ###################################################################*/
void Print(char String[])
{
	int	i;
	int N = strlen(String);

	for(i=0; i<N; i++)
	{
		AS1_SendChar(String[i]);		// Send String[i] across the SCI bus
		while(AS1_GetCharsInTxBuf() != 0);	// Wait until Tx buffer cleared
	}
}

/** ###################################################################
**	Function	: Target_Step
**	Input		: int countMS
**			  int period
**	Output		: long target
**
**	Target_Step takes countMS (millisecond counter) and period (in ms)
**	and generates a "step" target profile that switches between 2048
**	and 0 every period/2.
** ###################################################################*/
```

```
long Target_Step(int countMS, int period)
{
	long target;
	if(countMS < period/2)
	{
		target = 2048;
	}
	else
	{
		target = 0;
	}
	return target;
}

/** ###################################################################
**	Function	: Target_ZigZag
**	Input		: int countMS
**				  int period
**	Output		: long target
**
**	Target_ takes countMS (millisecond counter) and period (in ms)
**	and generates a linear "zigzag" target profile that rises from 0 to
**	between 0<countMS<period/2 and decreases from 2048 to 0 betwen
**	period/2<countMS<period.
** ###################################################################*/
long Target_ZigZag(int countMS, int period)
{
	long target;
	if(countMS < period/2)
	{
		target = (long)2048*2*(((double)countMS)/((double)period));
	}
	else
	{
		target = (long)2048*2*(1-((double)countMS)/((double)period));
	}
	target = target;
	return target;
}

/*long Target_Sine(int countMS, int period)
{
	long target;
	const double   PI = 3.141592653;
	target = (long)512*sin(2*PI*((double)countMS)/((double)period));

	return target;
}*/

/** ###################################################################
**	Function	: SetDutyPercent
**	Input		: float percent
**
**	SetDutyPercent takes a floating-point "percent" value (between 0 and
**	100) and sets the duty cycle of the PWM1 module. There is a "clamp"
**	feature which keeps the duty cycle between 0 and PWM frequency.
**	IMPORTANT NOTE: this function assumes a 20 kHz PWM frequency, or
**				a 400 clock ticks which translate to 50 us.
** ###################################################################*/
void SetDutyPercent(float percent)
{
	if(percent > 99.75)
	{
		PWM1_SetDutyTicks32(399);
	}
	else if(percent < 0.25)
	{
		PWM1_SetDutyTicks32(1);
	}
	else
	{
		PWM1_SetDutyTicks32((unsigned long)(400*percent/100));
	}
}
```

APPENDIX H: PERIPHERAL SETUP FOR H-BRIDGE BOARD TEST

Bean name	GPIO_A
Port	GPIOA
Pull resistor	autoselected pull
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes

Bean name	GPIO_B
Port	GPIOB
Pull resistor	autoselected pull
Open drain	no open drain
Direction	Input

Bean name	GPIO_E
Port	GPIOE
Pull resistor	autoselected pull
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes

Table 19: Peripheral Setup for H-Bridge Board Test 1

Bean name	PWM1
PWM or PPG timer	PWModA0
Output pin	PWMA0
Counter	PWM_A
Interrupt service/event	Disabled
PWMA	
PWMA prescaler	1
Period	20 kHz
Starting pulse width	0.025 us
Initial polarity	high
Iterations before action/event	1
Safe peiod in modes	no
Bean uses entire timer	yes
Initialization	
Enabled in init code	yes
Events enabled in init	yes

Table 20: Peripheral Setup for H-Bridge Board Test 2

Bean name	AS1
Channel	SCI1
Interrupt service/event	Disabled
Settings	
Parity	none
Width	8bits
Stop bit	1
SCI output mode	Normal
Receiver	Disabled
Transmitter	Enabled
TxD	GPIOD6_TXD1
TxD pin signal	
Baud rate	230400 baud
Break signal	Disabled
Wakeup condition	Idle line wakeup
Transmitter output	Not inverted
Stop in wait mode	no
Initialization	
Enabled in init code	yes
Events enabled in init	yes

Table 21: Peripheral Setup for H-Bridge Board Test 3

Bean name	FC1
Timer	TMRD0_Compare
Counter	TMRD0
Interrupt service/event	Disabled
Prescaler	4
Period	52428 ticks
Same resolution in modes	yes
Same resolution in modes	no
Initialization	
Enabled in init code	yes
Events enabled in init	yes

Table 22: Peripheral Setup for H-Bridge Board Test 4

Bean name	QD1
Device	Quad_Decoder1
Interrupt service/event	Disabled
Phase A pin	PHASEA1_TB0
Phase B pin	PHASEB1_TB1
Index	INDEX1_TB2
Index edge	positive edge
Index initialization	no
Home	HOME1_TB3
Home edge	positive edge
Home initialization	no
Watchdog	Disabled
Bypass decoder	no
Reverse counting	no
SwitchMatrix 1	
Mode	Raw input
FIR value	0
Filter frequency (kHz)	0
Initialization	
Events enabled in init	yes

Table 23: Peripheral Setup for H-Bridge Board Test 5

APPENDIX I: BRAIN BOARD TEST PROGRAM

```
/** ###################################################################
**     Filename  : Brain01.C
**     Project   : Brain01
**     Processor : 56F8347
**     Version   : Driver 01.09
**     Compiler  : Metrowerks DSP C Compiler
**     Date/Time : 4/19/2006, 2:38 PM
**     Abstract  :
**         Main module.
**         Here is to be placed user's code.
**     Settings  :
**     Contents  :
**         No public methods
**
**     (c) Copyright UNIS, spol. s r.o. 1997-2004
**     UNIS, spol. s r.o.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
** ###################################################################*/
/* MODULE Brain01 */

/* Including used modules for compiling procedure */
#include "Cpu.h"
#include "Events.h"
#include "A.h"
#include "B.h"
#include "C.h"
#include "D.h"
#include "F.h"
#include "SPI0Sel.h"
#include "ChipSelect.h"
#include "LED.h"
#include "ANA.h"
#include "ANB.h"
#include "MSInt.h"
#include "Math.h"
#include "PulseA.h"
#include "PulseB.h"
#include "SPI.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include <math.h>

#define t1 1000;

#define PutYellow(a)          a ? LED_SetBit(0) : LED_ClrBit(0)
#define GetYellow()           LED_GetBit(0)
#define PutGreen(a)           a ? LED_SetBit(1) : LED_ClrBit(1)
#define GetGreen()            LED_GetBit(1)
#define PutRed(a)             a ? LED_SetBit(2) : LED_ClrBit(2)
#define GetRed()              LED_GetBit(2)
#define SetDutyPercentA(ch,val)      PulseA_SetDuty(ch,(int)(3000*(long)val/100))
#define SetDutyPercentB(ch,val)      PulseB_SetDuty(ch,(int)(3000*(long)val/100))

void task1(void);

unsigned int time1;

unsigned int upPWM, duty;
unsigned int ANAVal[8], ANBVal[8], ANVal[16];

extern unsigned int data;

int main(void)
{
```

```
      int i, j;
      unsigned int temp[1000];
      long sum, avg, stdev;

      unsigned int local[16];

/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
PE_low_level_init();
/*** End of Processor Expert internal initialization.                    ***/

/* Write your code here */
      time1 = t1;

      /* Initialization for PWM "sweep" test */
      duty = 0;             //Initial duty cycle value for PWM
      upPWM = 1;            //Variable for PWM "sweep" test - upcount from 0 to 100% first

      /* Initialization for SPI ADC */
      for(i=0; i<16; i++)    local[i] = 0;  //Initialize local probe array
      for(i=0; i<16; i++)    ANVal[i] = 0;  //Initialize count storage
      SPI0Sel_ClrBit(0);
      SPI0Sel_ClrBit(1);

      /* BEGIN A/C performance test for average and stdev */
      for (i = 0; i < 1000; i++)
      {
             ANB_Measure(1);
      //Perform A/D for all channels, ANB15:0
             ANB_GetChanValue16(0, &temp[i]);     //Save one channel value in temp[i]
             temp[i] >>= 4;
      }
      sum = 0;                      //Accumulator for A/D counts
      for (i = 0; i < 1000; i++)
             sum += temp[i];        //Sum A/D counts
      avg = sum/1000;               //Calc avg A/D count across 1000 samples
      sum = 0;                      //Sum (sample[i]-avg)^2
      for (i = 0; i < 1000; i++)
             sum += ((long)temp[i]-avg)*((long)temp[i]-avg);
      stdev = Math_mfr32Sqrt(sum<<1);
      stdev = stdev/999;            //Calc stdev across 1000 samples
      asm(NOP);
      /* END A/D performance test */

      //Send dummy conversion to AD7490
      ChipSelect_ClrVal();
      while(ERR_OK != SPI_SendChar(0xffff)) ;
      //Dummy conversions

      //Configure AD7490
      ChipSelect_ClrVal();
      while(ERR_OK != SPI_SendChar(0b1111111110010000)) ;  //Write to control reg

      /* BEGIN AD7490 SPI performance test for average and stdev */
      j = 0;
      for(i=0; i<16000; i++)
      {
             ChipSelect_ClrVal();
             while(ERR_OK != SPI_SendChar(0)) ;
             while(FALSE==ChipSelect_GetVal()) ;          //Wait until CS cleared by ISR

             if(((0xf000 & data)>>12)==15)
             {
                    temp[j] = data & 0x0fff;
                    j++;
             }
      }
      sum = 0;                      //Accumulator for A/D counts
      for(i = 0; i < j; i++)
             sum += temp[i];        //Sum A/D counts
      avg = sum/j;                  //Calc avg A/D count across j samples
      sum = 0;                      //Sum (sample[i]-avg)^2
      for(i = 0; i < j; i++)
             sum += ((long)temp[i]-avg)*((long)temp[i]-avg);
      stdev = Math_mfr32Sqrt(sum<<1);
      stdev = stdev/(j-1);          //Calc stdev across j-1 samples
```

```
        asm(NOP);
        /* END AD7490 SPI performance test for average and stdev */


  while(1)
  {
        if(time1 == 0) task1();                       //Run task1 every t1 milliseconds
        for(i=0; i<16; i++)    local[i] = ANVal[i];   //Get AD7490 results in local
  }
}

//-------------------------------------------------
void task1(void)
{
        char probe;
        time1 = t1;

        //AD7490 conversion
        ChipSelect_ClrVal();
        while(ERR_OK != SPI_SendChar(0)) ;
        ChipSelect_SetVal();

        /* Store ADC results from previous run */
        ANA_GetValue16(&ANAVal[0]);
        ANB_GetValue16(&ANBVal[0]);

        /* Start next ADC */
        ANA_Measure(1);
        ANB_Measure(1);

        /* LED blink test code */
        PutYellow(!GetYellow());
        PutGreen(!GetGreen());
        PutRed(!GetRed());

        B_PutVal(~B_GetVal());

        /* PWM sweep test code */
        duty += (upPWM != 0) ? 1.8 : -1.8;
        upPWM = (upPWM) ? !(duty >= 100) : (duty <= 0);
        SetDutyPercentA(0,duty);
  PulseA_Load();

        /* PORT A-F output test code */
        A_PutVal((unsigned char)~A_GetVal());
        B_PutVal((unsigned char)~B_GetVal());
        C_PutVal((unsigned char)~C_GetVal());
        D_PutVal((unsigned char)~D_GetVal());
        E_PutVal((unsigned char)~E_GetVal());
        F_PutVal(~F_GetVal());
}

/* END Brain01 */
/*
** ###################################################################
**
**     This file was created by UNIS Processor Expert 2.96 [03.65]
**     for the Freescale 56800 series of microcontrollers.
**
** ###################################################################
*/


/** ###################################################################
**     Filename  : Events.C
**     Project   : Brain01
**     Processor : 56F8347
**     Beantype  : Events
**     Version   : Driver 01.02
**     Compiler  : Metrowerks DSP C Compiler
**     Date/Time : 4/19/2006, 2:59 PM
**     Abstract  :
**         This is user's event module.
**         Put your event handler code here.
**     Settings  :
```

```
**     Contents  :
**         MSInt_OnInterrupt - void MSInt_OnInterrupt(void);
**
**     (c) Copyright UNIS, spol. s r.o. 1997-2004
**     UNIS, spol. s r.o.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
** ###################################################################*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"

extern int time1;
extern int ANVal[16];
unsigned int data;

/*
** ===================================================================
**     Event       :  MSInt_OnInterrupt (module Events)
**
**     From bean   :  MSInt [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the bean is enabled - "Enable" and the events are
**         enabled - "EnableEvent").
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers'
property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the
ISR)      */
void MSInt_OnInterrupt(void)
{
  /* Write your code here ... */
  if(time1 > 0)        time1--;
}

/*
** ===================================================================
**     Event       :  SPI_OnRxChar (module Events)
**
**     From bean   :  SPI [SynchroMaster]
**     Description :
**         This event is called after a correct character is
**         received.
**         DMA mode:
**         If DMA controller is available on the selected CPU and
**         the receiver is configured to use DMA controller then
**         this event is disabled. Only OnFullRxBuf method can be
**         used in DMA mode.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers'
property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the
ISR)      */
void SPI_OnRxChar(void)
{
  /* Write your code here ... */
  //unsigned int data;

  while(ERR_OK != SPI_RecvChar(&data)) ;
  ANVal[(0xf000 & data)>>12] = data & 0x0fff;
  ChipSelect_SetVal();
}

/*
```

```
** ===================================================================
**     Event       :  SPI_OnTxChar (module Events)
**
**     From bean   :  SPI [SynchroMaster]
**     Description :
**         This event is called after a character is transmitted.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers'
property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the
ISR)      */
void SPI_OnTxChar(void)
{
  /* Write your code here ... */

}

/*
** ===================================================================
**     Event       :  SPI_OnError (module Events)
**
**     From bean   :  SPI [SynchroMaster]
**     Description :
**         This event is called when a channel error (not the error
**         returned by a given method) occurs. The errors can be
**         read using <GetError> method.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
#pragma interrupt called /* Comment this line if the appropriate 'Interrupt preserve registers'
property */
                         /* is set to 'yes' (#pragma interrupt saveall is generated before the
ISR)      */
void SPI_OnError(void)
{
  /* Write your code here ... */
}


/* END Events */

/*
** ###################################################################
**
**     This file was created by UNIS Processor Expert 2.96 [03.65]
**     for the Freescale 56800 series of microcontrollers.
**
** ###################################################################
*/
```

APPENDIX J: PERIPHERAL SETUP FOR BRAIN BOARD TEST

Type	ByteIO
Bean name	A
Port	GPIOA_Low
Pull resistor	pull up
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes

Type	ByteIO
Bean name	B
Port	GPIOB
Pull resistor	pull up
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes

Type	ByteIO
Bean name	C
Port	GPIOC_Low
Pull resistor	pull up
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes

Type	ByteIO
Bean name	D
Port	GPIOD_LOW
Pull resistor	pull up
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes

Table 24: Peripheral Setup for Brain Board Test 1

Type	WordIO
Bean name	F
Port	GPIOF
Pull resistor	autoselected pull
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes

Type	BitIO
Bean name	ChipSelect
Pin for I/O	GPIOE7_SS0B
Pin signal	
Pull resistor	autoselected pull
Open drain	no open drain
Direction	Output
Initialization	
Init direction	Output
Init value	1
Safe mode	yes
Optimization for	Speed

Table 25: Peripheral Setup for Brain Board Test 2

Type	ADC
Bean name	ANA
A/D converter	ADCA
Sharing	Disabled
Interrupt service/event	Disabled
A/D channels	8
Channel0	
A/D channel (pin)	ANA0
Mode select	Single Ended
Channel1	
A/D channel (pin)	ANA1
Mode select	Single Ended
Channel2	
A/D channel (pin)	ANA2
Mode select	Single Ended
Channel3	
A/D channel (pin)	ANA3
Mode select	Single Ended
Channel4	
A/D channel (pin)	ANA4
Mode select	Single Ended
Channel5	
A/D channel (pin)	ANA5
Mode select	Single Ended
Channel6	
A/D channel (pin)	ANA6
Mode select	Single Ended
Channel 7	
A/D channel (pin)	ANA7
Mode select	Single Ended
Queue	Enabled
A/D prescaler	ADCA_ADCR2
A/D resolution	12 bits
Conversion time	1.7 us
Internal trigger	Disabled
Volt ref recovery time	100
Power up delay	13
Power saving mode	Disabled
Number of conversions	1
Initialization	
Enabled in init code	yes
Events enabled in init	yes

Type	ADC
Bean name	ANB
A/D converter	ADCB
Sharing	Disabled
Interrupt service/event	Disabled
A/D channels	8
Channel0	
A/D channel (pin)	ANB0
Mode select	Single Ended
Channel1	
A/D channel (pin)	ANB1
Mode select	Single Ended
Channel2	
A/D channel (pin)	ANB2
Mode select	Single Ended
Channel3	
A/D channel (pin)	ANB3
Mode select	Single Ended
Channel4	
A/D channel (pin)	ANB4
Mode select	Single Ended
Channel5	
A/D channel (pin)	ANB5
Mode select	Single Ended
Channel6	
A/D channel (pin)	ANB6
Mode select	Single Ended
Channel 7	
A/D channel (pin)	ANB7
Mode select	Single Ended
Queue	Enabled
A/D prescaler	ADCB_ADCR2
A/D resolution	12 bits
Conversion time	1.7 us
Internal trigger	Disabled
Volt ref recovery time	100
Power up delay	13
Power saving mode	Disabled
Number of conversions	1
Initialization	
Enabled in init code	yes
Events enabled in init	yes

Table 26: Peripheral Setup for Brain Board Test 3

Type	BitsIO
Bean name	SPI0Sel
Port	GPIOA_High
Pins	2
Pin0	
Pin	GPIOA9_A1
Pin1	
Pin	GPIOA10_A2
Pull resistor	autoselected pull
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0

Type	BitsIO
Bean name	LED
Port	GPIOA_High
Pins	3
Pin0	
Pin	GPIOA11_A3
Pin1	
Pin	GPIOA12_A4
Pin2	
Pin	GPIOA13_A5
Pull resistor	autoselected pull
Open drain	no open drain
Direction	Input/Output
Initialization	
Init direction	Output
Init value	0
Safe mode	yes
Optimization for	speed

Table 27: Peripheral Setup for Brain Board Test 4

Type	TimerInt
Bean name	MSInt
Timer	TMRA0_Compare
Counter	TMRA0_Compare
Interrupt service/event	Enabled
Interrupt	INT_TMRA0
Interrupt priority	medium priority
Interrupt preserve registers	yes
Prescaler	1
Interrupt period	1 ms
Same period in modes	yes
Bean uses entire timer	no
Initialization	
Enabled in init code	yes
Events enabled in init	yes

Table 28: Peripheral Setup for Brain Board Test 5

Type	PWMMC
Bean name	PulseA
Device	PWM_A
Align	edge-aligned mode
Mode of PWM Pair 0	independent
Mode of PWM Pair 1	independent
Mode of PWM Pair 2	independent
Top-Side PWM Pair 0 Polarity	Positive
Top-Side PWM Pair 1 Polarity	Positive
Top-Side PWM Pair 2 Polarity	Positive
Bottom-Side PWM Pair 0 Polarity	Positive
Bottom-Side PWM Pair 1 Polarity	Positive
Bottom-Side PWM Pair 2 Polarity	Positive
Write Protect	no
Output pads	Enabled
Enable in Wait mode	no
Enable in ENOnCE mode	no
Frequency	20 kHz
Same frequency in modes	no
PWMA	
PWMA prescaler	1
Reload	1
Hardware acceleartion	Disabled
Dead-time	0 us
Correction	Disabled
Interrupt service/event	Disabled
Channel 0	
Channel	PWModA0
Duty	50%
Output software control	no
Mask channel	no
Channel 1	
Channel	PWModA1
Duty	50%
Output software control	no
Mask channel	no
Channel 2	
Channel	PWModA2
Duty	50%
Output software control	no
Mask channel	no
Channel 3	
Channel	PWModA3
Duty	50%
Output software control	no
Mask channel	no
Channel 4	
Channel	PWModA4
Duty	50%
Output software control	no
Mask channel	no
Channel 5	
Channel	PWModA5
Duty	50%
Output software control	no
Mask channel	no

Table 29: Peripheral Setup for Brain Board Test 6

Type	PWMMC
Bean name	PulseB
Device	PWM_B
Align	edge-aligned mode
Mode of PWM Pair 0	independent
Mode of PWM Pair 1	independent
Mode of PWM Pair 2	independent
Top-Side PWM Pair 0 Polarity	Positive
Top-Side PWM Pair 1 Polarity	Positive
Top-Side PWM Pair 2 Polarity	Positive
Bottom-Side PWM Pair 0 Polarity	Positive
Bottom-Side PWM Pair 1 Polarity	Positive
Bottom-Side PWM Pair 2 Polarity	Positive
Write Protect	no
Output pads	Enabled
Enable in Wait mode	no
Enable in ENOnCE mode	no
Frequency	20 kHz
Same frequency in modes	no
PWMA	
PWMA prescaler	1
Reload	1
Hardware acceleartion	Disabled
Dead-time	0 us
Correction	Disabled
Interrupt service/event	Disabled
Channel 0	
Channel	PWModB0
Duty	50%
Output software control	no
Mask channel	no
Channel 1	
Channel	PWModB1
Duty	50%
Output software control	no
Mask channel	no
Channel 2	
Channel	PWModB2
Duty	50%
Output software control	no
Mask channel	no
Channel 3	
Channel	PWModB3
Duty	50%
Output software control	no
Mask channel	no
Channel 4	
Channel	PWModB4
Duty	50%
Output software control	no
Mask channel	no
Channel 5	
Channel	PWModB5
Duty	50%
Output software control	no
Mask channel	no

Table 30: Peripheral Setup for Brain Board Test 7

Type	SynchroMaster
Bean name	SPI
Channel	SPI0
Interrupt service/event	Enabled
Interrupt from input	INT_SPI0_RxFull
Interrupt input priority	medium priority
Interrupt input preserve registers	yes
Interrupt from output	INT_SPI0_TxEmpty
Interrupt output priority	medium priority
Interrupt output preserve registers	yes
Input buffer size	0
Output buffer size	0
Settings	
Width	16 bits
Input pin	Enabled
Pin	GPIOE6_MISO0
Output pin	
Pin	GPIOE5_MOSI0
Clock pin	
Pin	GPIOE4_CSLK0
Slave select pin	Disabled
Clock edge	rising edge
Shift clock rate	7.5 MHz
Empty character	0
Ignore empty char	no
Send MSB first	yes
Wired-OR mode	Disabled
Shift clock idle polarity	High
Fault mode	Disabled
Initialization	
Enabled in init code	yes
Events enabled in init	yes

Table 31: Peripheral Setup for Brain Board Test 8