Electrical Interfaces for Electromechanical and Energy Systems

by
Rachel M. Chaney
B.S. Electrical Engineering and Computer Science
Massachusetts Institute of Technology 2010

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology
June 2010

Author____
Department of Electrical Engineering and Computer Science
January 27, 2010

Certified by
Dr. Steven B Leeb
Professor of Electrical Engineering and Computer Science and & Mechanical Engineering
Thesis Supervisor

Accepted by_____
Dr. Christopher J. Terman
Chairman, Department on Graduate Theses

(this page intentionally left blank)

Electrical Interfaces for Electromechanical and Energy Systems
by
Rachel M. Chaney

Submitted to the Department of Electrical Engineering and Computer Science on January 29, 2010 in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

Abstract

The design, construction, and testing of a versatile robot driver circuit is described. The printed circuit board produced can be used as an interface between any two-motor robot and the R31-JP, an eight-bit microcontroller system. The circuit board implements sensing, Ethernet communications, and motor driving modules. These modules are composed of eight bit peripheral microcontroller chips such as the ADC0808 analog-to-digital converter and the 84C54 programmable timer. Additionally, a programmable logic chip, the GAL22V10 creates the signals necessary for driving both direct current and stepper motors. The robot driver printed circuit board was testing in simulation and in hardware. The results are listed in this document. This robot driver will allow for the use of wireless mobile robots that can be used in future iterations of the Microcomputer Project Laboratory, 6.115.

The design, construction, and testing of a sensor signal conditioning printed circuit board for use in a hardware emulator for a Gas Turbine Generator used on the US Navy's DDG-51 Class Destroyer is described. The system emulator project seeks to construct a scaled hardware model and perform control experiments to explore different shipboard power distribution systems. The synchronization of two generators is necessary to fulfill these goals to extend versatility and allow for the testing of new power distribution systems. The signal conditioning circuit uses digital logic to develop a frequency sensor, which reduces the required sampling rate. Additionally, analog amplifier circuits are used to condition the signals output by voltage and current transducers, preparing them for analog to digital conversion. These signals will be used to implement the synchronization and load-balancing algorithms described in this document. This scaled model for shipboard power distribution systems will be demonstrate and compare experimental power distribution systems that will lead to increases in the safety and efficiency of shipboard power distribution systems.

Thesis Supervisor: Steven B. Leeb
Professor of Electrical Engineering and Computer Science & Mechanical Engineering

(this page intentionally left blank)

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

Research being conducted at the Massachusetts Institute of Technology's Laboratory for Electromagnetic and Electronic Systems includes a focus on interfaces between electromechanical and electrical systems as well as between energy and electrical systems. These interfaces allow for electrical sensing control. This thesis describes two different interfaces that allow electrical control over electromechanical and energy systems.

The design, construction, and testing of a versatile robot driver circuit is described in Chapter 2. The board produced can be used as an interface between any two-motor robot and the R31-JP, an eight-bit microcontroller system. Section 2.1 contains an explanation of the circuits used to interact with a robot's motors and sensors; this section details the method of communication with the 8051 microcontroller used on the R31-JP as well as hardware concerns with sensors and motors. The software that allows the robot driver to transparently drive both direct current and stepper motors in explained in Section 2.2. This robot driver will allow for the use of wireless mobile robots that can be used in future iterations of the Microcomputer Project Laboratory, 6.115.

The design, construction, and testing of a sensor signal conditioning circuit for use in a hardware emulator for a Gas Turbine Generator used on the US Navy's DDG-51 Class Destroyer is discussed in Chapter 3. The system emulator is a long term project that seeks to integrate Non-Intrusive Load Monitoring and Zonal Electrical Distribution [1] with future shipboard power distribution systems by constructing a scaled hardware model and performing control experiments to explore different shipboard power distribution systems. This chapter discusses the synchronization of two matched generators to extend versatility and allow for the testing of new power distribution systems. The design of a sensor signal conditioning printed circuit board seeks to aid the implementation of software

controlled synchronizing. The printed circuit board will held the generator system to approximate the discussed control systems.

Chapter 4 discusses possible improvements for the robot driver and generator emulator projects.

2 Robot Driver

This section details the design, construction, and testing of a versatile robot driver meant to be used as an interface between a prefabricated two motor robot and an R31-JP, an eight-bit microcontroller system. The finished product is a printed circuit board that allows the robot to be controlled with 8051 assembly code which can be downloaded through a serial cable or over wireless Ethernet. The driver board is not specific to a particular robot, but is meant to be useful across a variety of platforms. The driver board will work for any robot that has two independent motors that can be controlled to propel and steer the vehicle. The type of robot intended for use with this board is shown in schematic form in Figure 1. A specific model that was used for prototyping and testing is shown in Figure 2. The robot driver provides an interface between the eight-bit microcontroller system and the motors. Additionally, it provides an Ethernet interface, a resonant-frequency sensor, and an analog-to-digital converter for connection with additional sensors. The connections of these circuit blocks are shown in the block diagram in Figure 3.

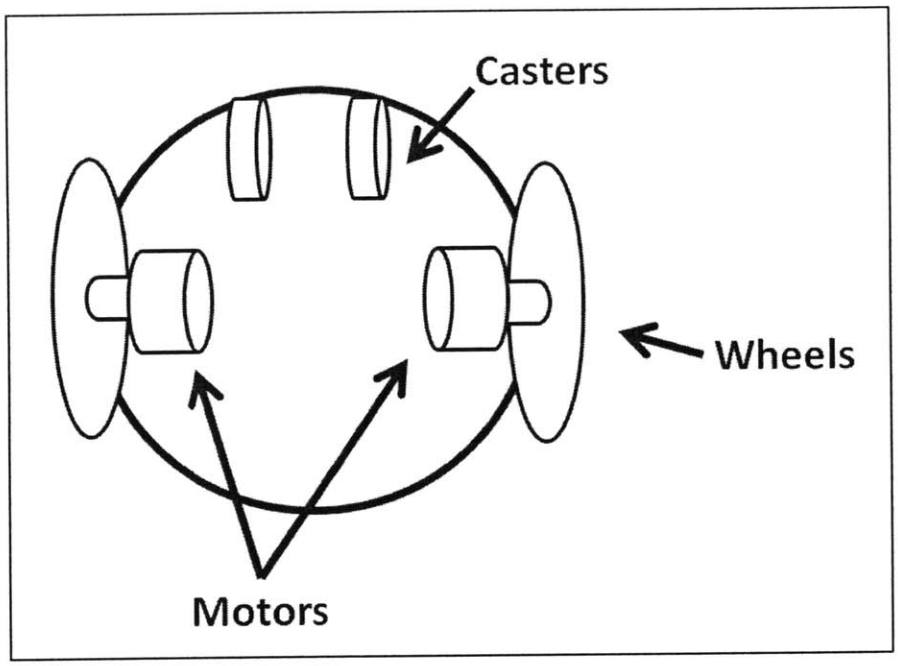


Figure 1 – Robot used to test the driver board.

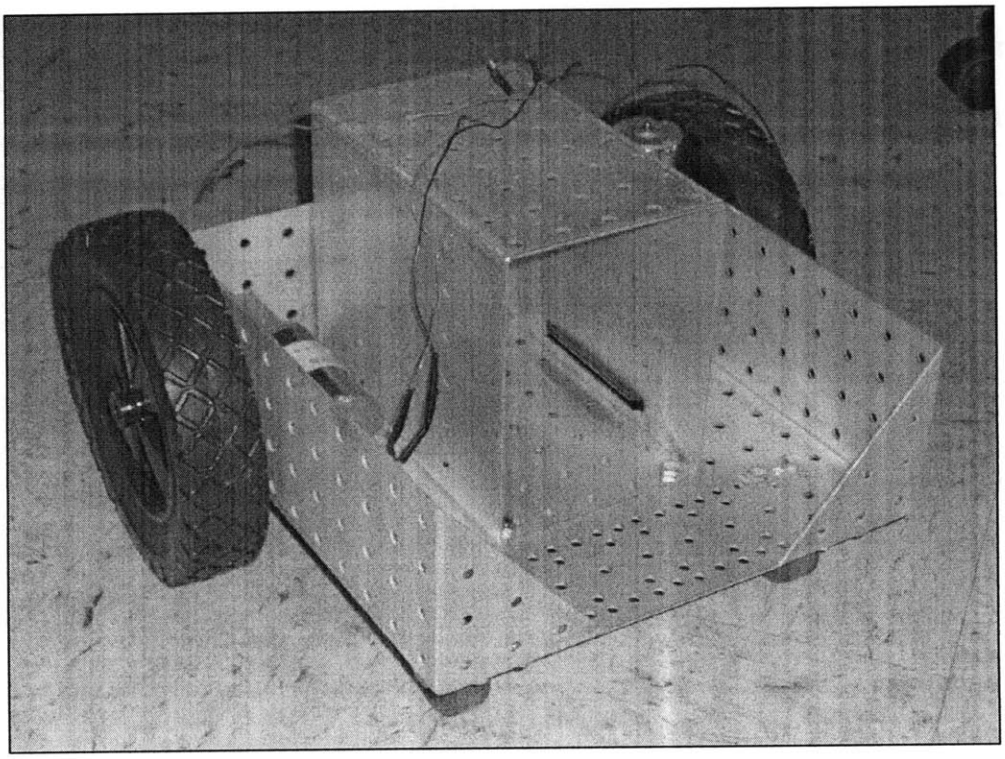


Figure 2 – Robot used for testing.

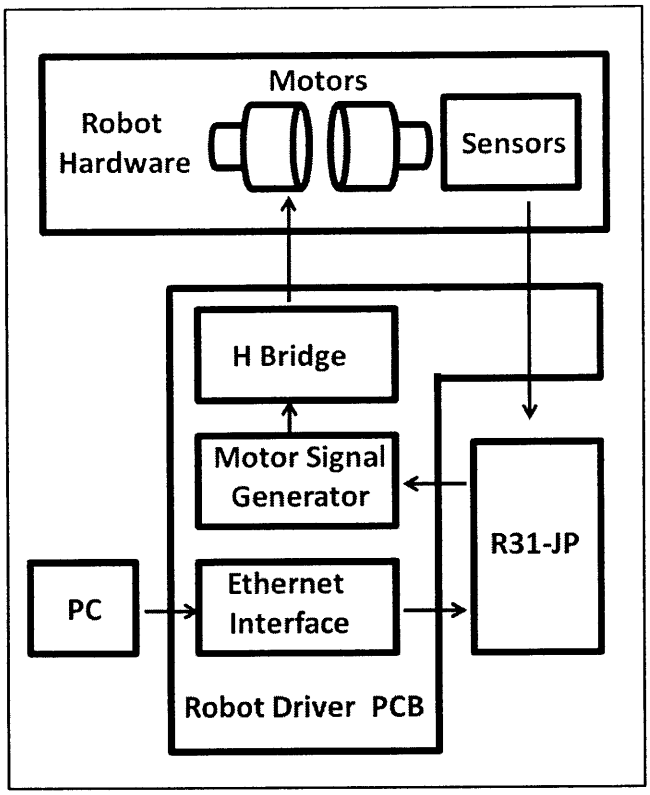


Figure 3 – Robot driver block diagram.

Both hardware and software are required to make this driver perform all the desired actions. The circuits that implement each block in this diagram are described in Section 2.1. The coded logic that implements the motor control systems is described in Section 2.2.

2.1 Circuit Explanation

This section explains the motivation and implementation of the circuits in the block diagram in Figure 3. The circuits are shown in their entirety in Appendix A. In order to understand the architecture of the robot driver board, it is necessary to understand the data flow implemented in the R31-JP, the 8051 microcontroller system shown in Figure 4 that interfaces with the robot board. The following sections detail data signal connections, locomotion interface, sensor interface, and Ethernet interface implemented with the robot driver.

2.1.1 Data Flow

The R31-JP is a particular eight-bit microcontroller system shown in Figure 4, which is comprised of an 8051 microcontroller along with a few auxiliary chips that add random-access and read-only memory as well as easy communication with computers through a serial port. The 8015 microcontroller has an eight bit data bus, an eight bit address bus, and a few other important signals for communicating with peripheral chips and computers. Figure 5 shows the distribution of these signals to the robot driver board.

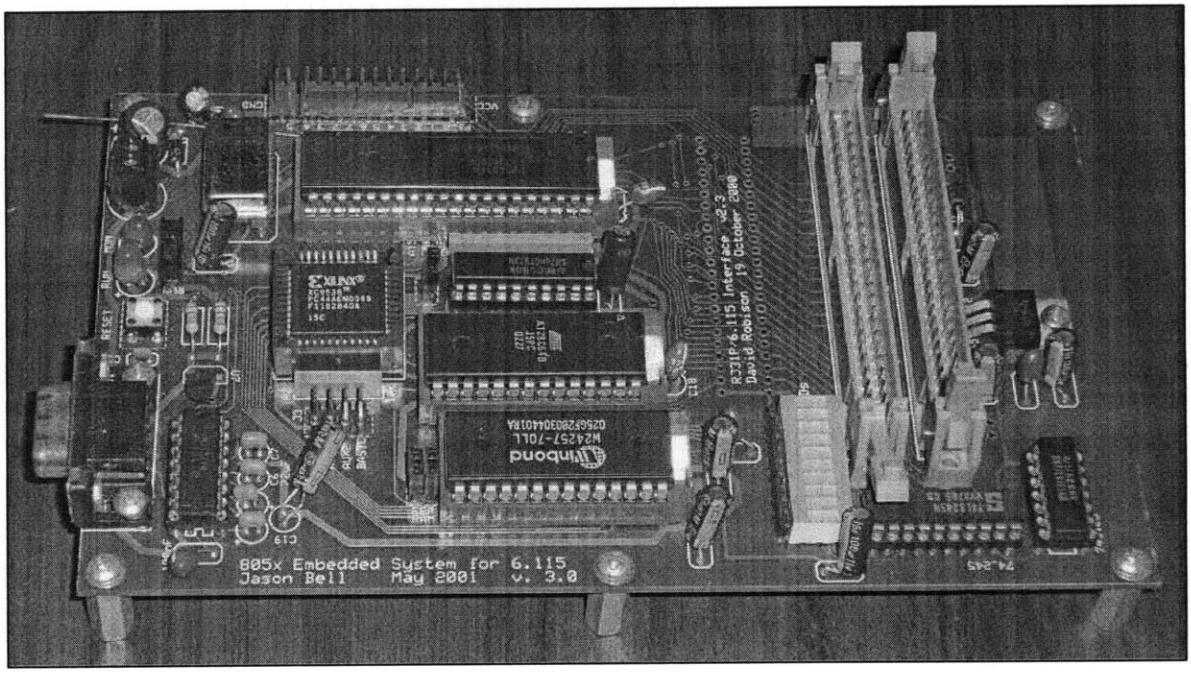


Figure 4 – R31-JP printed circuit board.

The data bus, shown as D0-D7, is an input / output bus that is shared between the 8051 microcontroller and the connected peripheral chips. It allows information to be shared between chips. */RD* and */WR* signals are used to indicate actions like a read or write to memory locations. The address bus, shown as A0-A7 contains the memory location of a desired byte of data. Eight bit memory locations are referred to with hexadecimal numbers for simplicity. The complete details of the memory structures in the R31-JP are beyond the scope of this document. A particularly relevant detail of the

memory structure is that all calls to read or write memory locations from FE00h to FEFFh are calls to virtual memory. Calling to these memory locations activates the */XSEL* signal, turning on peripheral chips connected to this signal outside the R31-JP board.

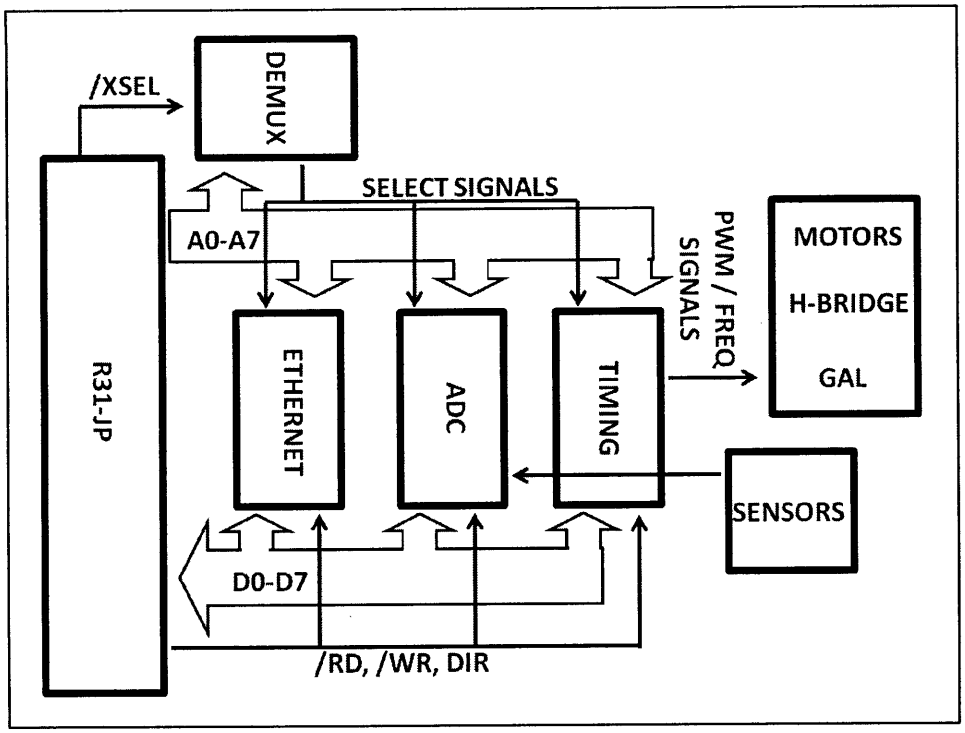


Figure 5 – Data flow in the robot driver circuit.

It is possible to make this behavior more specific by demultiplexing the address bus as shown in Figure 6. This configuration uses a three-to-eight demultiplexer, which allots different sections of available virtual memory to different peripheral chips. When A4-A6 are connected as shown in Figure 6, the allotment of virtual memory can be decoded by looking at Table 1. The motivation for using A4-A6 in particular is that they are the least significant address bits that are not used for addressing memory locations within any of the peripheral chips used. For reference, the CS8900A used in the Ethernet block requires four bits to address all of the internal memory locations, the ADC0808 requires three bits to address each of the eight analog-to-digital channels, and the 82C54 timer requires two bits to address all the control words and input numbers for the timers. This means that at least four address bits cannot be used in the demultiplexer. At maximum, a four-to-sixteen multiplexer can be used to create select

signals for sixteen peripheral chips. Only three chip select signals are required for the robot driver, so this scheme can be simplified. One three-to-eight demultiplexer can be used to create eight select signals. The three required for the locomotion, sensing, and communication blocks can be hardwired while the five extra signals are routed to connector J6 as shown in Figure 6 so they can be used in an outside circuit with additional peripheral chips. This demultiplexer increases the functionality of the R31-JP, allowing communication with the circuits used in the driver and with up to five additional peripheral chips.

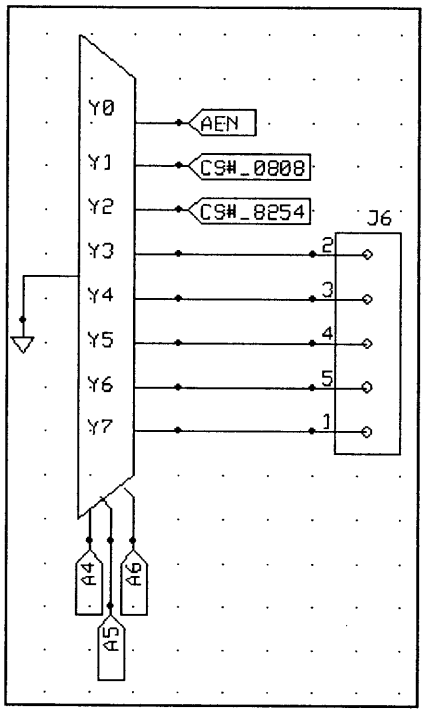


Figure 6 – Demultiplexing address bus.

Table 1 – Allotment of virtual memory locations.

Demux output	Selected Component	Alloted Memory Locations
Y0	CS8900A (Ethernet)	FE00h – FE0Fh
Y1	ADC0808 (ADC)	FE10h – FE1Fh
Y2	82C54 (Timer)	FE20h – FE2Fh
Y3	J6-pin 2	FE30h – FE3Fh
Y4	J6-pin 3	FE40h – FE4Fh

Y5	J6-pin 4	FE50h – FE5Fh
Y6	J6-pin 5	FE60h – FE6Fh
Y7	J6-pin 1	FE70h – FE7Fh

There are additional important signals that are routed from the microcontroller to the robot driver board. */WR* and */RD* are used to read from and write to the memory locations on the R31-JP and on the peripheral chips. The only signals that are not used for their typical purposes are *P3.3* and *P3.5*, two bits from port three of the 8051. This port can be used as part of an Input / Output bus for connecting to a peripheral chip. In the robot driver, they are dedicated signals that define motor direction. The use of these signals in controlling the motors is discussed in Section 2.1.4.

2.1.2 Sensor Interface

The ADC0808 is an eight channel analog-to-digital converter. Two channels are dedicated to reading the hardwired resonant-frequency simplifier. The remaining five channels are fed off-board using connector J3. These lines can be connected to additional analog inputs, such as phototransistors, thermistors, or wheel encoders. Example code in Figure 7 shows how to read information from the ADC0808.

```
Programmers Notepad - test_adc.asm
org 00h
ljmp start

org 100h
loop:
    lcall waiting        ; each channel need 100us between readings
    lcall read_adc
    mov P1, a            ; show adc reading on R31-JP LEDs
    sjmp loop

init:
    mov dptr, #0FE13h
    clr a
    movx @dptr, a        ; wake up the ADC
    mov R0, #01h
    mov R1, #01h
    ret

read_adc:
    mov dptr, #0FE13h
    movx a, @dptr        ; move channel 6 reading to acc
    ret

waiting:
    djnz R0, waiting
    mov R0, #01h
    djnz R1, waiting
    mov R1, #01h
    ret
```

Figure 7 – Example code for reading ADC0808 Channel.

The only hardwired sensing circuit on the driver is a resonant LC signal amplifier that allows the robot to sense when it is near a wire that is broadcasting the correct frequency. This legacy amplifier circuit, shown in Figure 8, was designed before this effort to create a robot driver board. The outputs of the amplifiers are connected to Channels 1 and 2 of the ADC0808; this allows them to be read by the R31-JP, such that they can be integrated into a locomotion feedback control system. There are two resonant-signal amplifier circuits; if the LC sensors are placed on either side of the robot, their readings can be used to center the robot over a broadcasting wire.

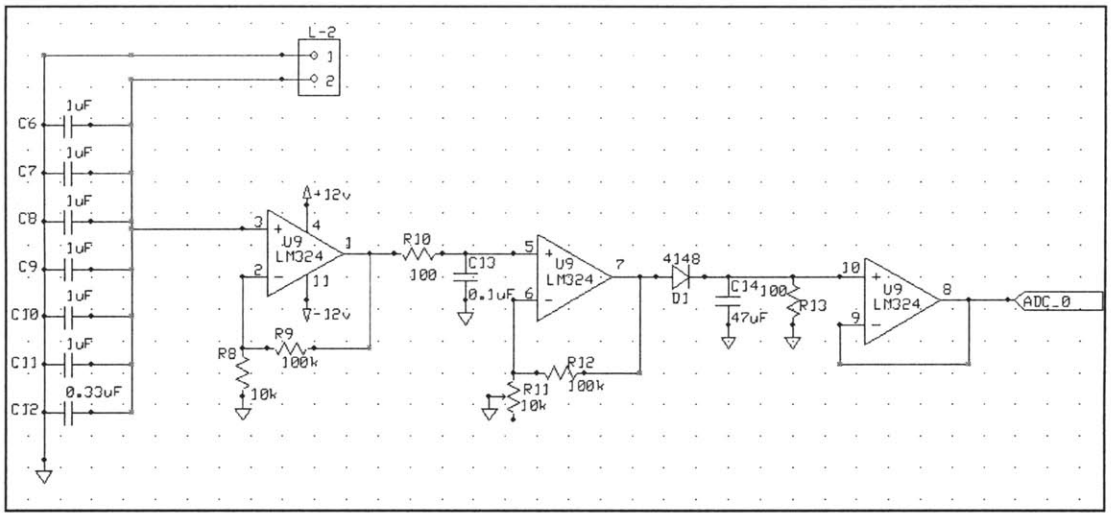


Figure 8 – Resonant signal amplifier circuit.

This amplifier converts a resonant-frequency sine wave into a larger magnitude triangle wave with the same fundamental frequency and a DC offset; the triangle wave reaches its minimum and maximum points when the sensed sine wave has a zero crossing as shown in Figure 9. The output signal is always positive and less than three volts, making it an ideal signal for analog-to-digital conversion.

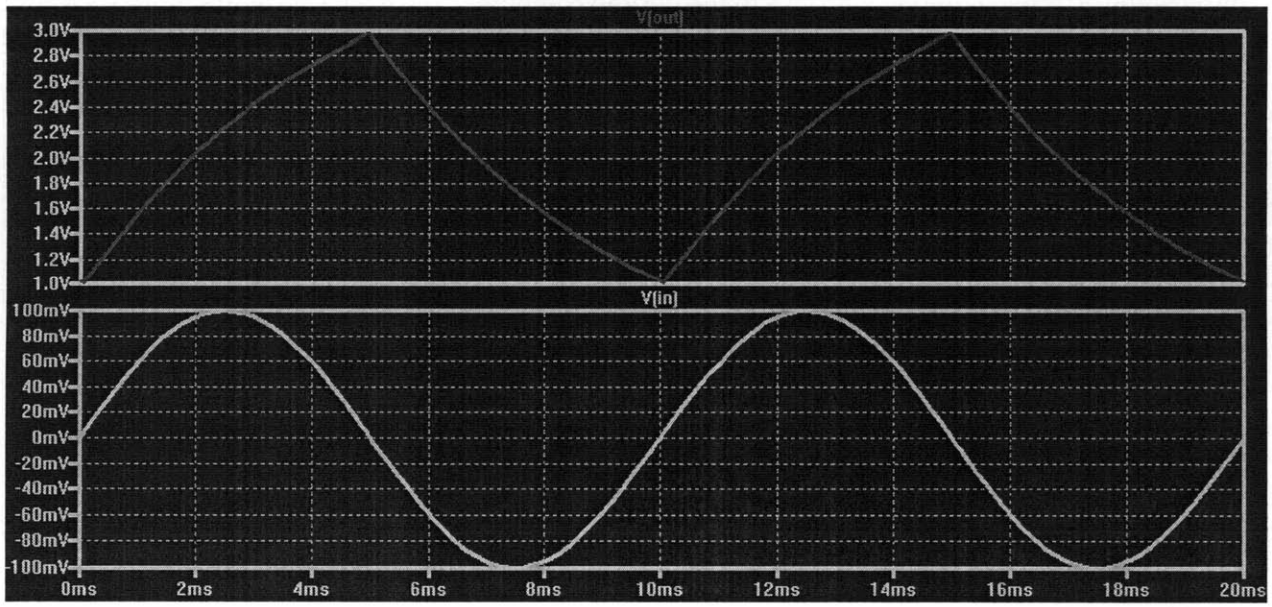


Figure 9 – Simulated waveforms for resonant-signal amplifier circuit.

After physical construction of the robot driver, this circuit was tested with an inductor of 1.0mH. An oscilloscope trace showing the correct functioning of the resonant-frequency amplifier is shown in _____.

___

2.1.3 Ethernet Interface

The Ethernet interface was designed by Sabrina Neuman with extensive reliance on [**2**]. The circuit schematic is shown in Appendix A. This functional block turns 8051 communications, which occur on parallel buses, into UDP packets which can be sent to and from a computer with an Ethernet interface. The Ethernet translator encoded in MINMON, the operating system of the R31-JP, must also translate from UDP packets to 8051 instructions, to allow for the downloading of programs onto the RAM of the R-31JP through the Ethernet connection. Additionally it must wrap 8051 assembly code instructions with UDP headers and footers such as sending and receiving address, and send the packets through the attached Ethernet interface to the desired computer. The main chip in this interface is the CS8900A, which requires connections to the data bus, address bus, and control lines discussed above.

Testing to ensure the Ethernet interface is fully functional has not yet been completed. The code for one test that has been performed is shown in Figure 10. This test does not utilize the Ethernet for communication since the computer is talking to the 8051, which is talking to the CS8900A. In this test just asks the CS8900A chip for its Product ID number, 630Eh.

Programmers Notepad - hiworld.asm

```
;=============================================================
;                Hello (CS8900A) world!
; Checks if the CS8900A can hear me
;-------------------------------------------------------------
; The CS8900A internal address 0000h is read with the PPPTR,
;   (Note: READ HIGH, THEN LOW! / WRITE LOW, THEN HIGH!)
; and should return the CS8900A Product ID Number: 630Eh
; The high and low byte are added, and the result returned on P1
;       63h + 0Eh = 71h
; OR    0110 0011b + 0000 1110b = 0111 0001b
;=============================================================
helloworld:
    mov dph, #0FEh
    mov dpl, #0Ah    ; set dptr to PPPTR low byte address
    mov a, #00h
    movx @dptr, a    ; send the low byte, #00h
    mov dph, #0FEh
    mov dpl, #0Bh    ; set dptr to PPPTR high byte address
    mov a, #00h
    movx @dptr, a    ; send the high byte, #00h
    mov dph, #0FEh
    mov dpl, #0Dh    ; set dptr to PPDataP0 high byte address
    movx a, @dptr    ; read the data high byte from PPD0h
        mov R1, a
    mov dph, #0FEh
    mov dpl, #0Ch    ; set dptr to PPDataP0 high byte address
    movx a, @dptr    ; read the data low byte from PPD0l
        add a, R1
        mov P1, a    ; show 63h + 0Eh = 71h on Port 1 LED's
godot:
    sjmp  godot      ; loop forever
```

Figure 10 – Testing code for Ethernet interface.

The Ethernet interface seems promising, but more testing is required to be sure that this circuit is fully functional. Preliminary code that may be useful for implementing the translation between assembly code and UDP packets is located in Appendix C [**3**]. Once the translation code is completed, the addition of a wireless adapter may allow for wireless communication with the robot and with the R31-JP.

2.1.4 Locomotion Circuit

The locomotion control circuit allows for independent control of two motors; both must be either stepper or direct current motors. The caster wheels located on the front end of the robot shown in Figure 1 are purely for balance and can turn freely in any direction. The motors are stationary and their shafts are secured to wheels so the velocity of the robot depends on the absolute and relative velocities of the two motors, which necessitates that the motors can rotate in both directions and are independently controlled. This suggests the use of an H-Bridge to interface between the electrical

circuit and the motors. The integrated circuit used to implement the H-Bridge is the NJM2670. Each IC contains two H-Bridges that are separated from the TTL inputs. A schematic for one of the H-Bridges in the chip is shown in Figure 11. The TTL inputs allow for easy connection to the logic signals produced by the 8051 microcontroller system and the peripheral chips.

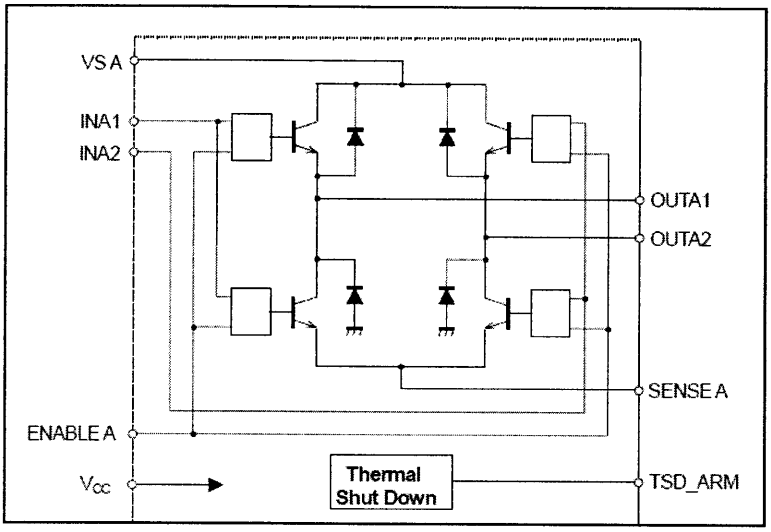


Figure 11 – NJM2670 H-Bridge connections. [4]

The connections of the H-Bridge to the motors depend on the type of motor. A DC motor only has two armature connections so two motors can be powered using one NJM2670. In contrast, the stepper motor used has six connections; as discussed later in Section 2.2.2, the two center-tapped connections are tied to ground. This eliminates the need for the H-Bridge topology since the center-tap connection to ground always allows current to flow through the coil when a positive voltage is applied to one of the other four connections. While the H-Bridge topology is unnecessary, there is a need for four controllable nodes for the four remaining motor connections; the H-Bridge integrated circuit is still used by connecting each remaining stepper motor wire to the four *OUT* pins. Thus, two ICs are required to control two stepper motors. These differences in physical interface make it obvious that different types of control signals are required for each motor.

The pulse width modulation (PWM) signals for the DC motors are digitally generated by the 82C54, a programmable 16-bit timer integrated circuit. The clock input for the 82C54 is connected to the crystal oscillator on the board, X2. Timer zero is set in rate generator mode; in this mode, the counter decrements from a given 16-bit number *N* and produces a pulse when it reaches one, and then reloads *N* to start again as shown in Figure 12. The frequency of the rate generation output is defined in Equation 1.

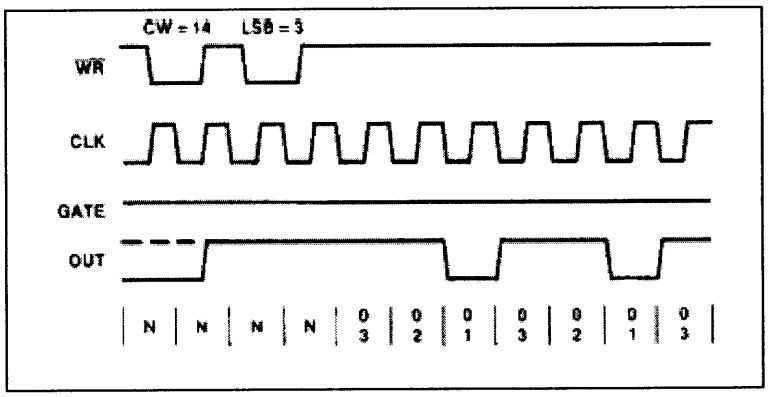


Figure 12 – Rate generator mode for 82C54. [5]

Equation 1

$$f_{sw} = \frac{f_{clk}}{2N}$$

Once the switching frequency f_{sw} has been established, the duty cycle for the two desired PWM signals must be set. The remaining two counters should be set to one-shot mode with the rate generator discussed above connected to their Gate input pins. When the rate generator pulses, the normally high output of the one-shot timer will be pulled low while a given 16-bit number *M* is decremented to one, after which the output will be high once more as shown in Figure 13. The duty cycle for a given *N* and *M* is defined in Equation 2. The duty cycles of the two PWM signals are completely independent from each other. Figure 14 shows two PWM signals with different duty cycles, a situation that crops up with M_1 does not equal M_2. This behavior can be used to control turning radius.

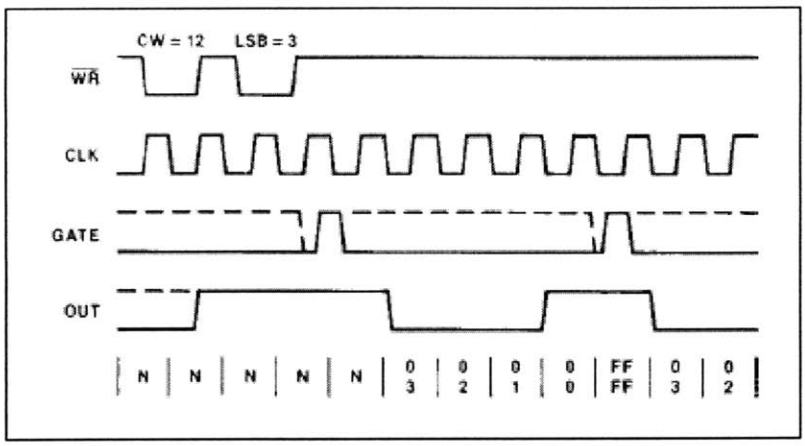


Figure 13 – One shot mode for 82C54. [5]

Equation 2

$$\frac{1-D}{f_{sw}} = \frac{2M}{f_{clk}} \rightarrow D = 1 - \frac{M}{N}$$

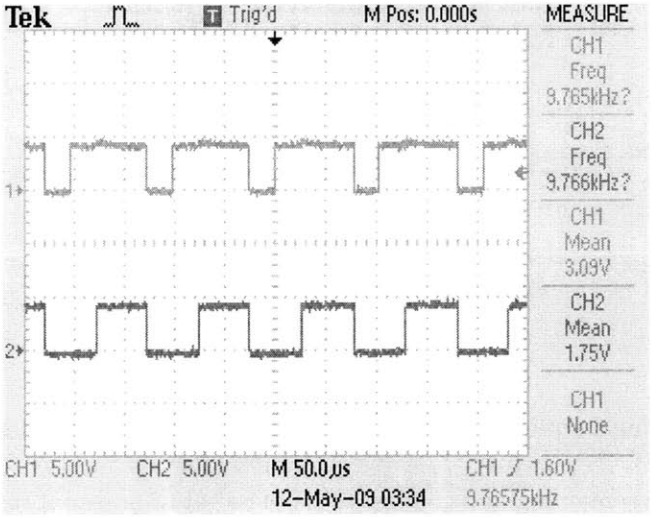


Figure 14 – Oscilloscope traces showing two different PWM TTL signals sent to H-Bridges

The resolution of the duty cycle depends on *N*. An expression for resolution is found in Equation 3. A summary of possible switching frequencies and their associated resolutions is shown in Table 2. Even the slowest possible switching frequency (305.18Hz) is acceptable for a PWM signal since the inertia of most DC motors will not allow them to respond to any signal at that high a frequency. The motor will respond only to the average value of the signal.

Equation 3

$$R = \frac{100\%}{N - 1}$$

Table 2 – Example switching frequencies and their associated resolutions.

N (hex)	F_{sw} (Hz)	Duty Cycle Resolution (%/bit)
FFFF	305.18	0.0015
7FFF	610.37	0.0031
00FF	78.431k	0.39
007F	157.48k	0.79
0003	6.667M	50.0

The main reason not to run at the slowest possible switching frequency is simplicity of computation. Since a 16-bit number must be given to the 82C54 to change the speed of the motor, the eight-bit microcontroller needs to perform some 16-bit arithmetic. To avoid implementing clunky arithmetic structures, it can be worthwhile to only calculate eight-bit input numbers. This eight-bit number can be the least significant byte of the input number *M* or *N*, while leaving the most significant byte 00h. The simplification of computation is worth the tradeoff in resolution unless exacting resolution is required for the desired robot application.

The stepper motor is simpler to control. As discussed later in Section 2.2.2, the speed and direction of a stepper motor depends on the speed and direction of a finite state machine shown in Figure 22. The simplest way to implement a repeatable sequence of states is to code a programmable logic device. The logic device used in this circuit is the GAL22V10. This chip has input and output pins that can be programmed to provide the desired combinations of static and sequential logic. Eight outputs of the GAL22V10 are physically connected to the eight inputs of the two H-Bride integrated circuits. This allows for command of each of the stepper motor coils individually as mentioned in 2.2.2 Stepper Motor Driver. The speed of the stepper motors is determined by the clock input for the GAL; because this clock must have a variable frequency, it is connected to timer zero of the 82C54.

Since the robot driver printed circuit board is used for both stepper and DC motors, there should not be extensive hardware alterations to switch from one to the other. The DC control system requires the outputs of timers one and two, as well as direction signals, to be connected to the H-Bridge inputs. The stepper control system requires eight outputs of the GAL to be connected to the H-Bridge inputs. In order to simplify requiring for switching motor types, the outputs of timers one and two of the 82C54 are routed through the GAL to the H-Bridges. Only four hardware jumpers (JP2, JP3, JP4, and JP5 in Appendix A) are required to change from DC to stepper mode. One additional jumper, JP1, is the motor-choice input to the GAL. This jumper ensures that the GAL is performing the correct function for the given motors. The motor-choice input is hardwired, rather than coded in software, ensuring that it cannot be wrongfully flipped with careless code since an unintended alteration in motor-choice will be eliminate the possibility for useful motor control.

2.2 Software Explanation

The robot driver interfaces with a pair of DC motors or a pair of stepper motors to allow for control of speed and direction of movement. This means that there are two different modes of operation that need to be incorporated into the locomotion module. Both of these modes can be distilled to sequential logic so they can be implemented using a programmable logic chip. The use of the GAL22V10 reduces the number of parts and the printed circuit board surface area required for implementing the DC and stepper motor control circuits. The following sections discuss the code that implements the control logic.

2.2.1 DC Motor Driver

Simple DC motors have two terminals, known as armature terminals, across which a voltage can be applied, energizing coils of wire inside the rotor and causing the motor to turn. It is necessary to apply a

known voltage across the armature terminals of the DC motor in order to control its speed. One problem with this method of control is that it requires a controllable DC voltage. This is difficult to directly implement in a microcontroller system since most digital-to-analog converters that might perform this action cannot source enough current to turn a motor, nor can they produce negative voltages to turn the motor in the reverse direction. Luckily, the inertia of the rotor causes the motor works as a low pass filter. The motor does not react to high frequency changes in armature voltage, but does react to changes in the time-average voltage across the armature terminals.

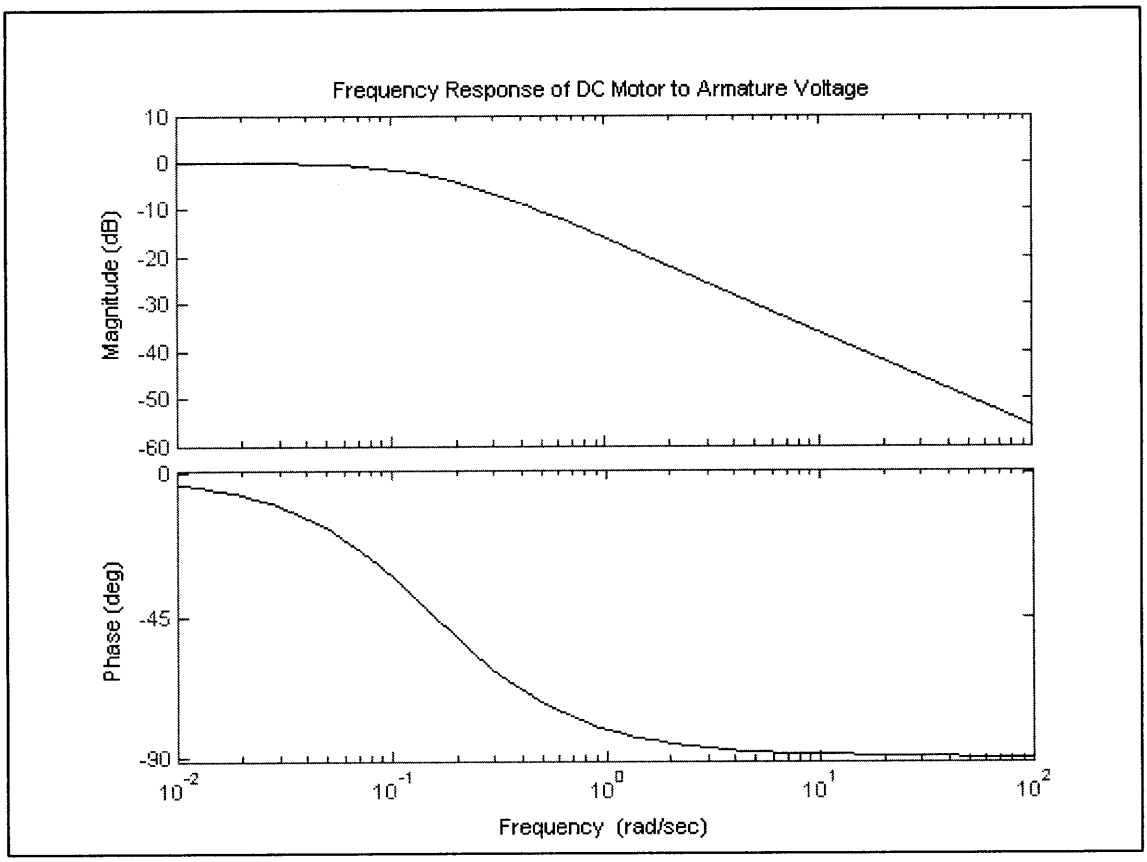


Figure 15 – Frequency response of a DC motor to armature voltage.

Equation 4

$$\frac{\omega}{V_a}(s) = \frac{k_t}{R_a J s + k_t^2}$$

The low pass nature of the motor suggests the use of pulse width modulation to digitally control the DC voltage across the armature of the motor. To effectively utilize PWM to control the speed of the motor, the switching frequency must be much greater than the crossover frequency of the open-loop motor. Figure 15 shows the normalized bode plot for a typical DC motor. The transfer function is shown in Equation 4. Choosing a switching frequency at least ten times faster than the crossover frequency ensures that the motor does not react to each individual pulse, but only sees the average voltage (corresponding to the duty cycle) of the signal.

Direction is defined with two dedicated pins, *P3.3* and *P3.5* for motors one and two respectively. *PWM* and *DIR* signals are applied at each H-Bridge input, thus the terminal voltage of the motor is the difference between *PWM* and *DIR* as shown in Figure 16. The speed-direction scheme has one subtle point. When the direction signal changes, the duty cycle also changes. In Figure 16, a *PWM* signal with duty cycle approximately 30% is applied to a motor. Note that the *DIR* signal applied changes. At first, the motor turns forward at 30% speed, but once the direction changed, the motor turns in reverse at 70% speed. This illustrates that it is important to change the PWM and direction signals in quick succession; otherwise the motor might exhibit unintended behavior.

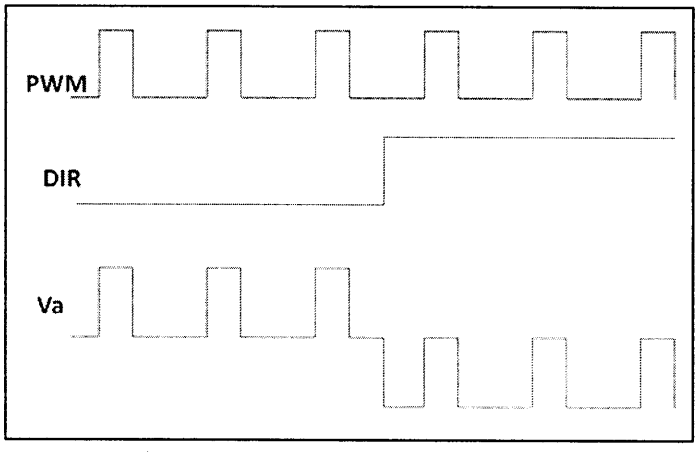


Figure 16 – Changing direction changes duty cycle.

The GAL22V10 code is shown in Appendix B. The GAL really just pushes the *PWM* and *DIR* signals through to the H-Bridges. To show that the GAL code performs correctly when it is acting as a PWM pusher as shown in Figure 17 simulated input and output waveforms are shown in Figure 18. Note that there are two different switching frequencies shown on *PWM1* and *PWM2*. This is not a situation that could occur in the physical system since both PWM signals use the same rate generator from the 82C54. This simulation just serves to illustrate that as a hardware, rather than software restriction.

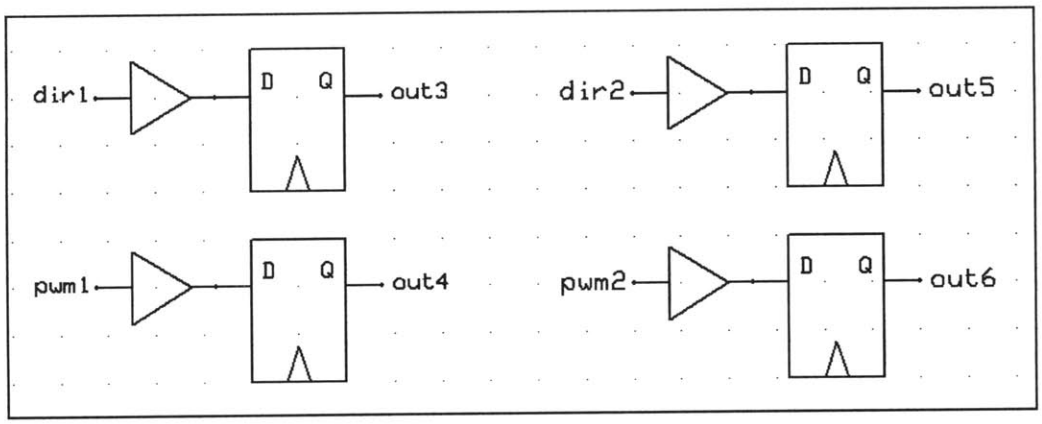


Figure 17 – Logic implemented with GAL code to run DC motors.

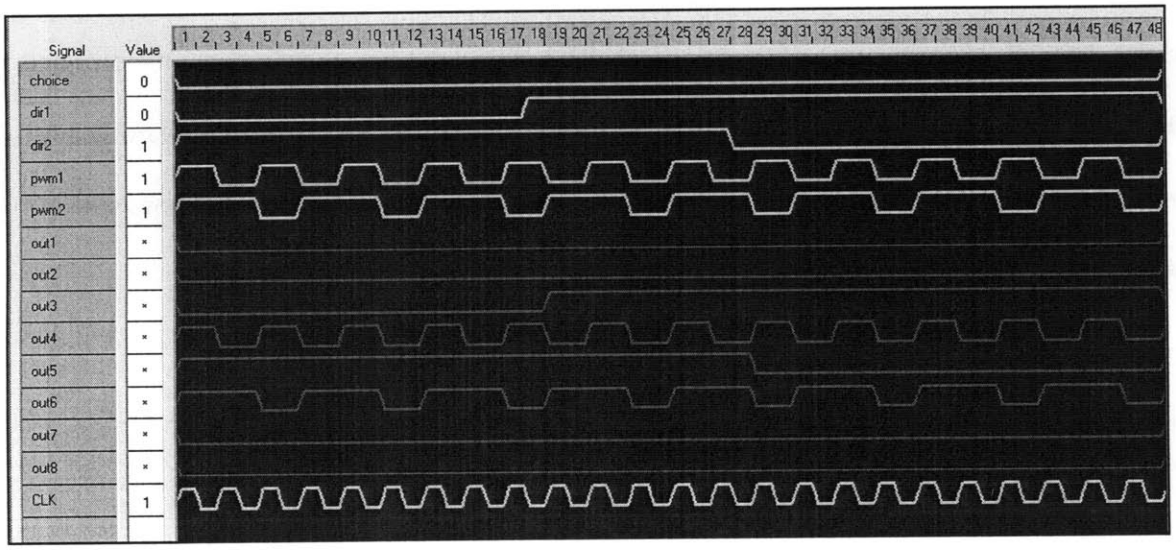


Figure 18 – Simulation of DC motor control code.

After circuit construction, the physical circuit was tested for correct DC motor control. Figure 19 shows the voltage at both H-Bridge outputs. Channel 1 is the *DIR* signal, which changes with a

frequency of 0.5Hz. Channel 2 is the *PWM* signal, which switches much faster than can be seen with the slow oscilloscope setting shown in the figure. The Math Channel shows the voltage across the motor terminals. The polarity of the terminal voltage, and so the rotor direction, change with the *DIR* input.

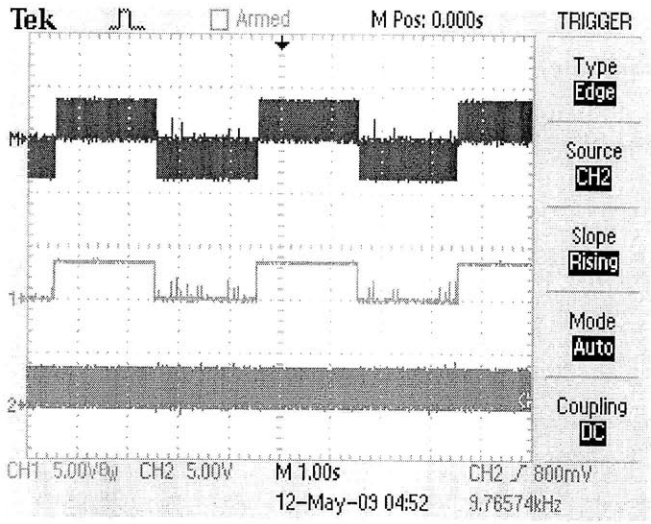


Figure 19 – Oscilloscope traces showing periodic direction changes for one DC motor

The code used to program the periodic direction change behavior in Figure 19 is shown in Figure 20. The motors were programmed with a switching frequency of about 4 kHz. The duty cycle of each motor is unchanged over the course of the test. Only *P3.3*, the bit defining the direction of motor one, is periodically changed. The physical manifestation of this code is that the robot moves forward slowly wiggling from left to right. This behavior is probably not useful in robot locomotion, but does serve to illustrate the principles for controlling the motion of the two DC motors.

Programmers Notepad - 1.asm

```
1   org 00h
2   ljmp start
3
4   org 000Bh
5       ljmp change_dir
6
7   org 100h
8   start:
9       lcall init
10      loop:
11          clr P3.5        ; motor 2 will not change direction
12          sjmp loop
13
14  change_dir:
15          inc R0
16          mov TH0, #0h
17          mov TL0, #0h
18          cjne R0, #1Ch, not_yet
19          cpl P3.3        ; motor 1 changes direction
20          mov R0, #0h     ; with freq ~0.5Hz
21          not_yet:
22              reti
23  init:
24      mov R0, #0h
25
26      mov dptr, #0FE33h
27      mov a, #34h             ; choose switching frequency
28      movx @dptr, a
29      mov dptr, #0FE30h       ; timer 0 in rate generator mode
30      mov a, #00h
31      movx @dptr, a           ; lsb of N
32      mov a, #04h
33      movx @dptr, a           ; msb of N
34
35      mov dptr, #0FE33h
36      mov a, #72h             ; choose duty cycle for motor 1
37      movx @dptr, a
38      mov dptr, #0FE31h       ; timer 1 in one-shot mode
39      mov a, #00h
40      movx @dptr, a           ; lsb of M
41      mov a, #01h
42      movx @dptr, a           ; msb of M
43
44
45      mov dptr, #0FE33h
46      mov a, #0B2h            ; choose duty cycle for motor 2
47      movx @dptr, a
48      mov dptr, #0FE32h       ; timer 2 in one-shot mode
49      mov a, #00h
50      movx @dptr, a           ; lsb of M
51      mov a, #02h
52      movx @dptr, a           ; msb of M
53
54      mov TMOD, #01h          ; initialize 16 bit timer for interrupt
55      mov TH0, #0h            ; the interrupt has frequency ~14Hz
56      mov TL0, #0h
57
58      mov IE, #82h            ; turn on interrupt
59      setb TR0
60
61      ret
62
```

Figure 20 – Code for periodically changing direction of DC motor.

In order to control the movement of the DC motors for robot locomotion, there are a variety of different sensors that could be used for feedback. For instance, optical wheel encoders could be used to calculate the exact frequency of wheel rotations to estimate the speed or turning angle of the robot. The main drawback of sensor is that it is not sensitive to wheel slippage. There are two main drawbacks

of DC motors in general: the DC voltage required across the armature to turn the rotor and the difficulty of measuring speed without additional sensors. Both of these drawbacks are rectified by using stepper motors as discussed in the next section.

2.2.2 Stepper Motor Driver

Stepper motors are similar to DC motors except the locations of the permanent and electromagnets are switched. In a DC motor, there are permanent magnets on the stator, and electromagnets on the rotor; as the rotor turns, a mechanical commutator changes the polarity of the electromagnet by energizing a different coils. Analogously, in a stepper motor, there are permanent magnets on the rotor and electromagnets on the stator; the polarity of the electromagnets is switched via electrical rather than mechanical commutation. The number of magnet pairs on the rotor determines the number of discrete steps required for a full rotation. When coils are energized, it pulls the rotor into a discrete position; the rotor will remain in this position as long as those coils are energized. When the next set of electromagnets is energized, the rotor will turn a discrete angle. The motor tested for use with the robot is a 30^0 per step six wire unipolar motor as shown in Figure 21. The stepper driver circuit performs the same function as the mechanical commutator, energizing subsequent coils of wire to turn the rotor.

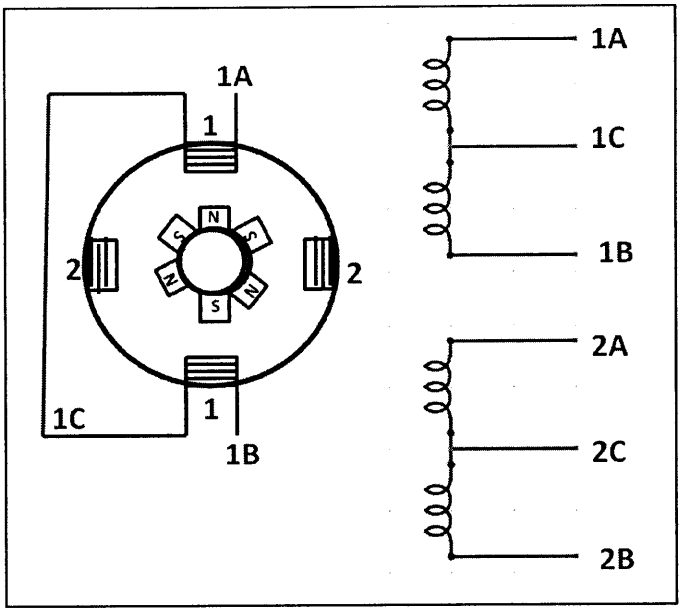


Figure 21 – Typical stepper motor connections.

To turn the rotor, the coils need to be energized in a sequence. Each electromagnetic pole pair must be energized in the opposite polarity of the previous pair in order to make the rotor turn. There are several different sequences that would turn the rotor. For simplicity's sake, the center-taps, 1C and 2C, are hardwired to ground. This allows each half coil to be energized with the BJT stacks in the NJM2670 as discussed in Section 2.1.4. The sequence used for this driver is shown in the finite state machine in Figure 22. The speed of rotation depends directly on the electrical frequency of the finite state machine as described in Equation 5.

Equation 5

$$\frac{f_{CLK}}{72} = \frac{f_{fsm}}{12} = f_{motor}$$

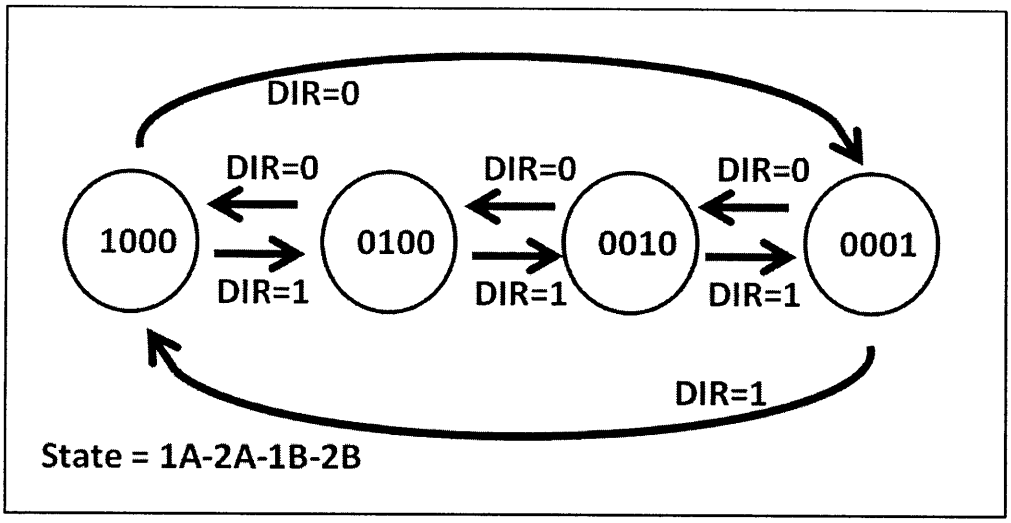


Figure 22 – Finite state machine for stepper motor control.

The finite state machine can be implemented using sequential logic; Table 3 shows a truth table. The GAL22V10 implementation of this finite state machine requires combinational and sequential logic. The combinational logic is coded into the GAL using standard Boolean operators. The sequential logic is coded using the D-flip-flop capability of the GAL. The D-flip-flop has two inputs, one for data and one for a clock signal. The input is isolated from the output except on the positive clock edge, where the input is pushed to the output. The combinational logic tests the current state of the machine and the current input; on the rising clock edge, the next state is pushed to the outputs.

Table 3 – Truth table for stepper motor control.

CURRENT STATE				INPUT	NEW STATE			
out1	out2	out3	out4	dir1	out1	out2	out3	out4
1	0	0	0	0	0	0	0	1
0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0
1	0	0	0	1	0	1	0	0
0	1	0	0	1	0	0	1	0
0	0	1	0	1	0	0	0	1
0	0	0	1	1	1	0	0	0
0	0	0	0	X	0	0	0	1

The discrete logic circuit schematic for the code listed in Appendix A is shown in Figure 23. The use of AND gates to test the current state and direction input ensures that the output state is always either one of the proscribed states or in the case of some malfunction, all zeros. The inclusion of the zero-state possibility in the logic circuit allows for recovery from an error; state 0001 will follow a zero-state, which will follow any erroneous state.

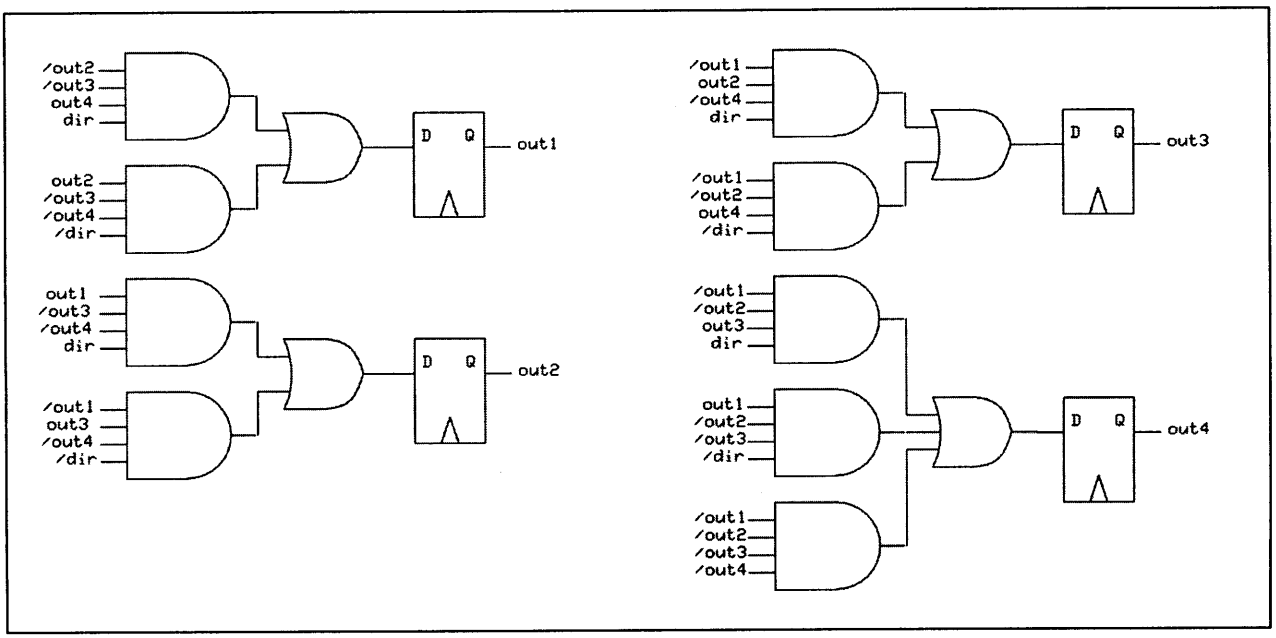


Figure 23 – Logic implemented with GAL22V10 code to turn stepper motors.

There are three inputs that control the behavior of the steppers; the clock input to the GAL, which controls the speed of the motors, and the *P3.3* and *P3.5* direction inputs that control motors one and two, respectively. The speed of the motor relates to the speed of the clock input as shown in Equation 5. As before, the output of timer zero of the 82C54 is connected to the clock input of the GAL. This means that the frequency of the clock signal to the finite state machine is defined by the 16-bit number *N* given to the 82C54 as described in Equation 6. One drawback of the code implemented is that the speed of the two motors is always the same, thus the operation of the two motors is not completely independent. While this does not allow the robot to perform behaviors like wide or narrow radius like the DC scheme does, these types of behaviors can be approximated by alternating the

direction of one of the motors with some duty cycle proportional to the desired turning radius. This drawback comes with some benefit; the R31-JP assembly code used to command these motors is simpler than that used to command the DC motors.

Equation 6

$$f_{CLK} = \frac{f_{system\_clk}}{2N}$$

The code implemented is shown in Appendix B. A simulation for this code is shown in Figure 24; in reference to Figure 21, *1A, 1B, 2A*, and *2B* are connected to *OUT1, OUT2, OUT3*, and *OUT4* respectively. The other stepper motor is connected to the other four outputs in the same way. These simulation results show that the speed of the finite state machine depends on the clock input. The *PWM1* and *PWM2* inputs shown on the simulation have no impact on the output. The direction of each of the motors changes when the *DIR1* and *DIR2* inputs change.

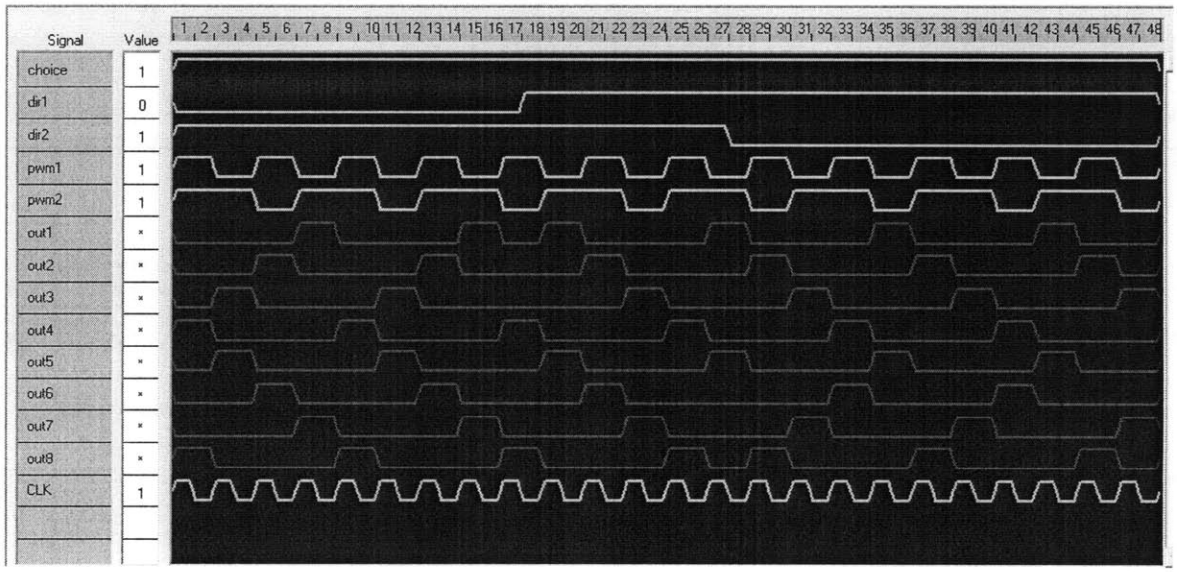


Figure 24 – Simulation of stepper motor control code

After physical construction, the circuit was tested for correct stepper motor control. Oscilloscope traces proving the functionality of the stepper driver are shown in Figure 25. In this figure Channel 1, the upper trace on each image, is *OUT1*, while Channel 2, the lower trace, changes from

OUT2 to *OUT3*, to *OUT4*, showing the progression of the finite state machine. The R31-JP assembly code used to produce these signals is shown in Figure 25.

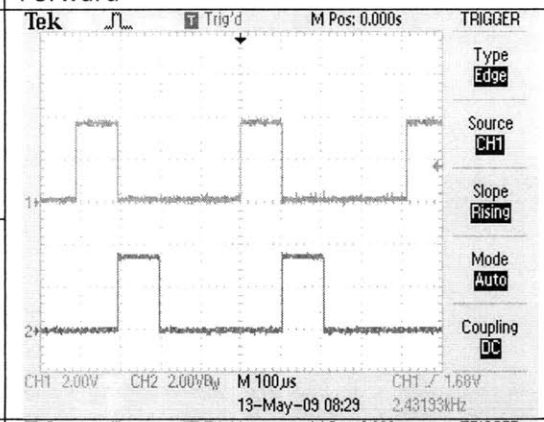
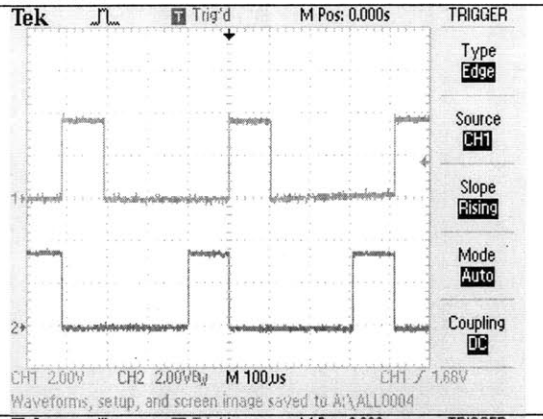
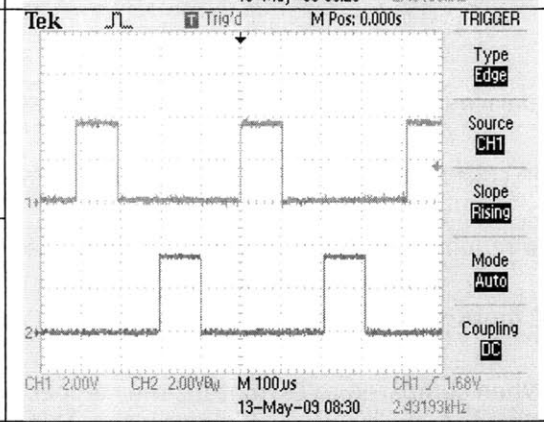
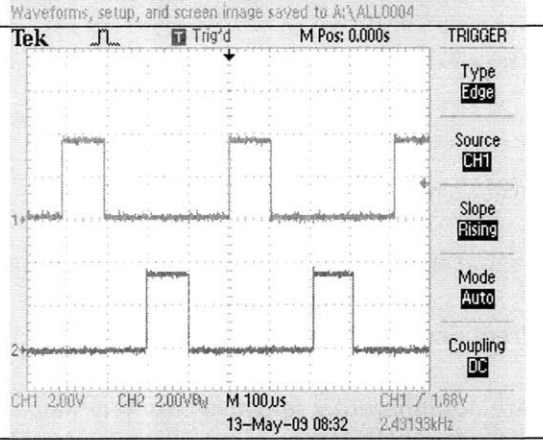
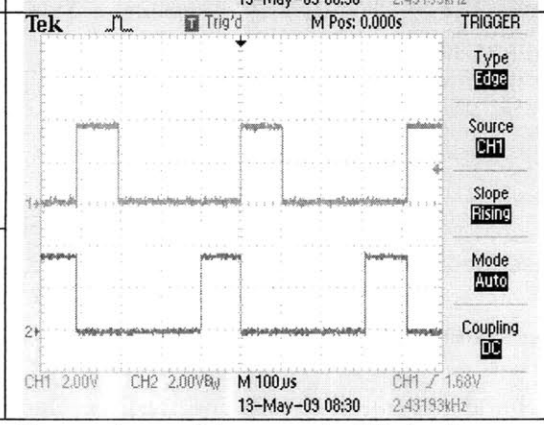
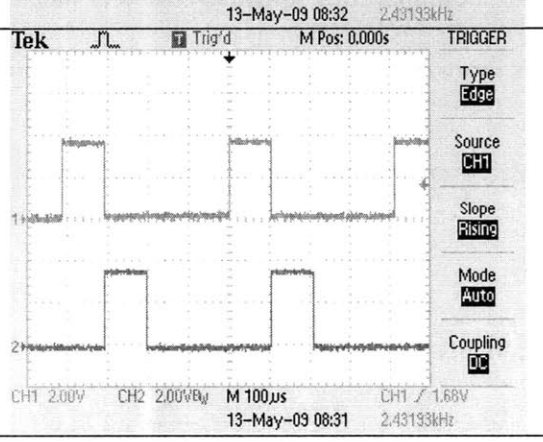
Signals	Forward	Reverse
Out1 Out2		
Out1 Out3		
Out1 Out4		

Figure 25 – Oscilloscope traces of stepper motor TTL signals.

Programmers Notepad - 2.asm

```
1   org 00h
2   ljmp start
3
4   org 100h
5   start:
6       lcall init
7       loop:
8           clr P3.5            ; motor2 goes reverse
9           setb P3.3           ; motor1 goes forward
10          sjmp loop
11
12  init:
13      mov R0, #0h
14
15      mov dptr, #0FE33h
16      mov a, #36h             ; choose frequency for
17      movx @dptr, a           ; clk input on GAL
18      mov dptr, #0FE30h
19      mov a, #00h
20      movx @dptr, a
21      mov a, #40h
22      movx @dptr, a
23      ret
24
```

Figure 26 – Code for turning stepper motors.

Stepper motors can be run very slowly so each discrete step is seen or more quickly so the rotor's movement seems more continuous. If the coded frequency is too great, the rotor will vibrate rather than rotate. Stepper motors eliminate the requirement for outside encoder devices since the frequency of the rotor is already known. Like DC motors with wheel encoders however, stepper motors are not inherently sensitive to wheel slippage. When compared to DC motors, the stepper motor code is more complex, but the electrical hardware and R31-JP assembly code is simpler.

2.3 Robot Use

The electrical hardware and software described above allow for a robot driver circuit board that can interface between the R31-JP microcontroller system and any two-motor robot. The robot driver also allows for the connection of additional peripheral integrated circuits and sensors. The final version of the robot driver printed circuit board is shown in Figure 27.

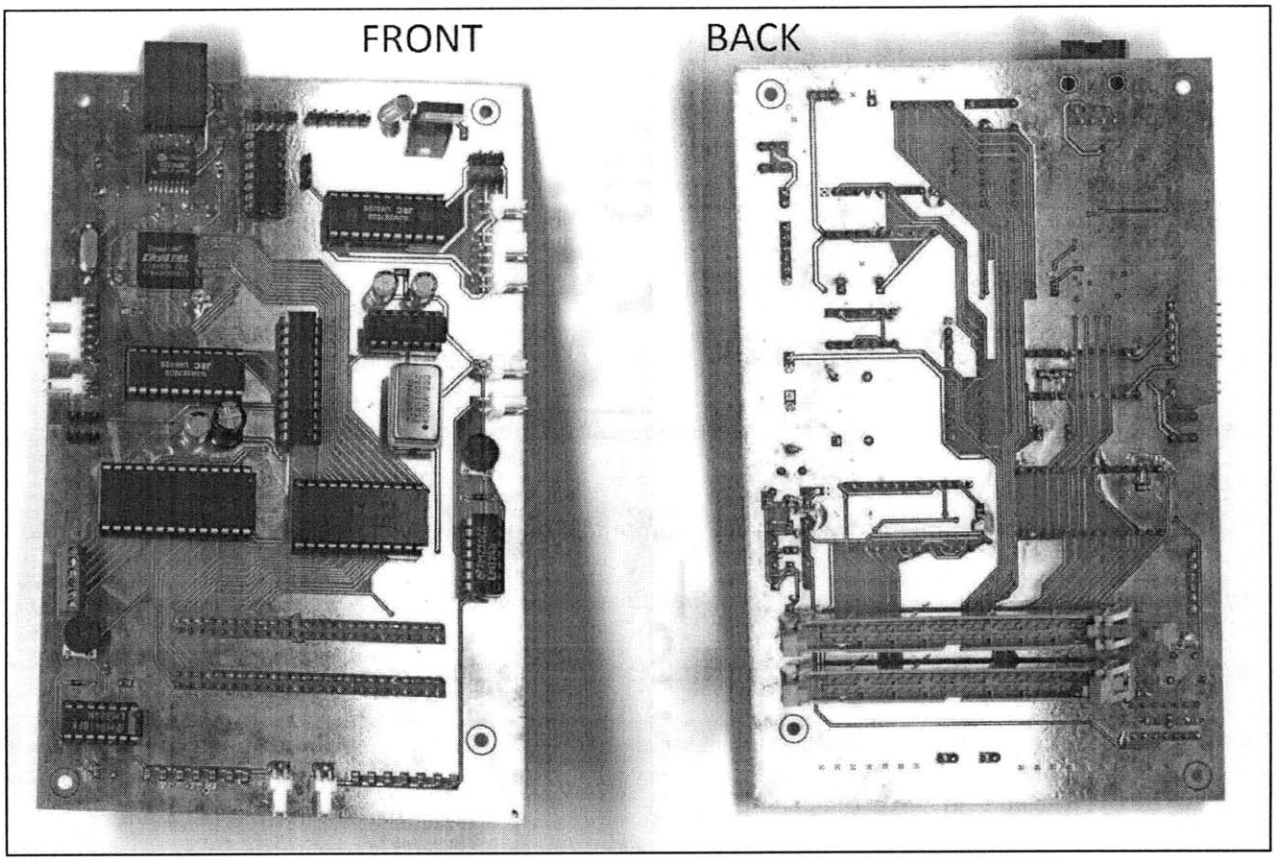


Figure 27 – Robot driver printed circuit board.

3 Shipboard Generator Emulator Sensor Board

The shipboard generator emulator is a long term project that seeks to integrate Non-Intrusive Load Monitoring (NILM) and Zonal Electrical Distribution [**1**] with future shipboard power distribution systems. The scaled hardware model allows for the construction of different power distribution systems and the performance of control experiments and control techniques that incorporate NILM. The generator emulator will be used to test different power distribution systems for possible shipboard use.

The generator emulator was originally comprised of one five-kilowatt three-phase salient-pole synchronous generator. The generator shaft was turned by two series-connected DC motors; these motors were powered by two parallel-connected Xantrex XHR 150-7S 1000W remote-controlled voltage- mode power supplies. The armature voltage across the motors, thus the speed of the generator shaft, was controlled using a Proportional-Integral controller implemented in LabView. The hardware was interfaced with the software through the K-8061 Velleman Hobbyist USB Board shown in Figure 28, which has numerous analog and digital inputs and outputs that allow for commanding the power supply as well as reading sensor inputs [**6**]. To inform this feedback compensator, analog frequency and RMS voltage sensors were designed by Jacob Osterberg [**7**]. The sensors' analog outputs reduced the number of wires required to connect to the K-8061. Unfortunately, this priority reduced the range, resolution, and linearity of the sensors as discussed later in this chapter.

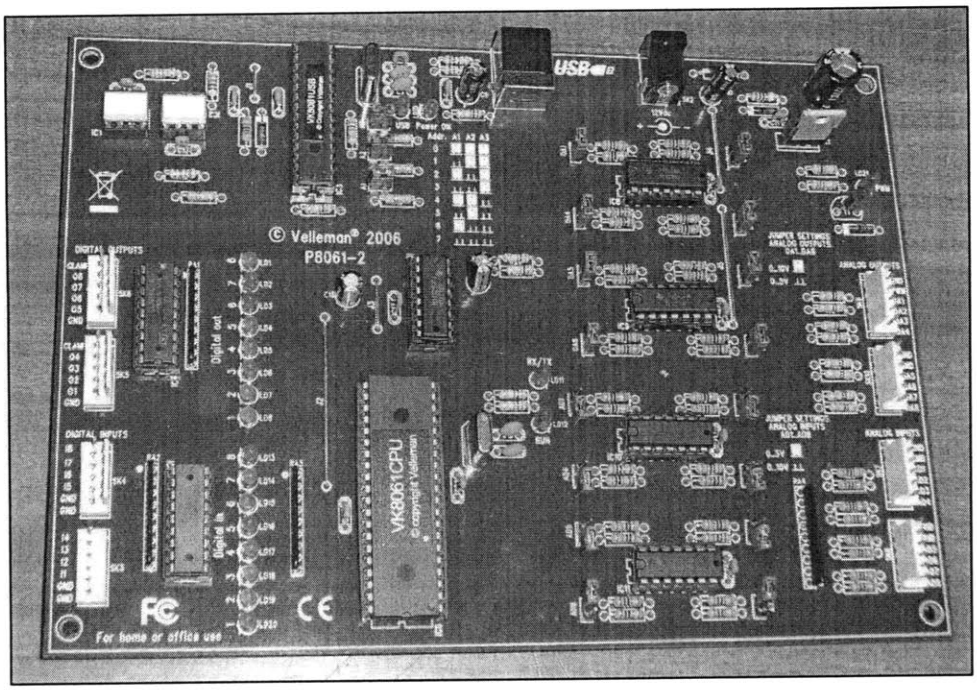


Figure 28 – The K-8061 used to interface emulator hardware and software.

The described generator system could not emulate many of the behaviors of shipboard generators. Specifically, the amount of power generated was not sufficient; a step in the size of the load would cause the prime mover power supplies to current limit. Additionally, there was no way to emulate split-plant alternating current power distribution systems. The following sections explain the improvements to the generator emulator system to fulfill these needs.

3.1 Power Upgrades

The prime mover power supplies cannot produce enough power to keep the generator behaving linearly with its Proportional-Integral controller. This becomes apparent when the size of the load driven by the output of the generator changed in an instant; the power supplies would current-limit. In order to reduce this saturation behavior, an upgrade in prime mover power supply was performed. A remotely controlled power supply with greater ratings would allow more power to be applied to the prime mover. The new power supply, the Xantrex XFR150-18 is capable of supplying 2800W, an

increase of 40% from the previous power supplies. This was shown to be more than enough to keep from current limiting given the conditions mentioned above.

The XHR150-7S was moved to the field windings to allow for remote control. This allows for software control through the K-8061 for the field windings as well as for the shaft speed. The repurposing of this power supply allows for the extension of the LabView Proportional-Integral controller to include the root-mean-squared voltage level of the output waveforms as well as their frequency.

The acquisition of another similar synchronous generator allows for the possibility of emulating split-plant power distribution systems, control systems for multiple synchronized generators, and power distribution systems requiring much greater generation capabilities. The remaining XHR150-7S was connected across the field windings and another XFR150-18 was connected to the prime mover. The new emulator set up, complete with two generators is shown in Figure 29.

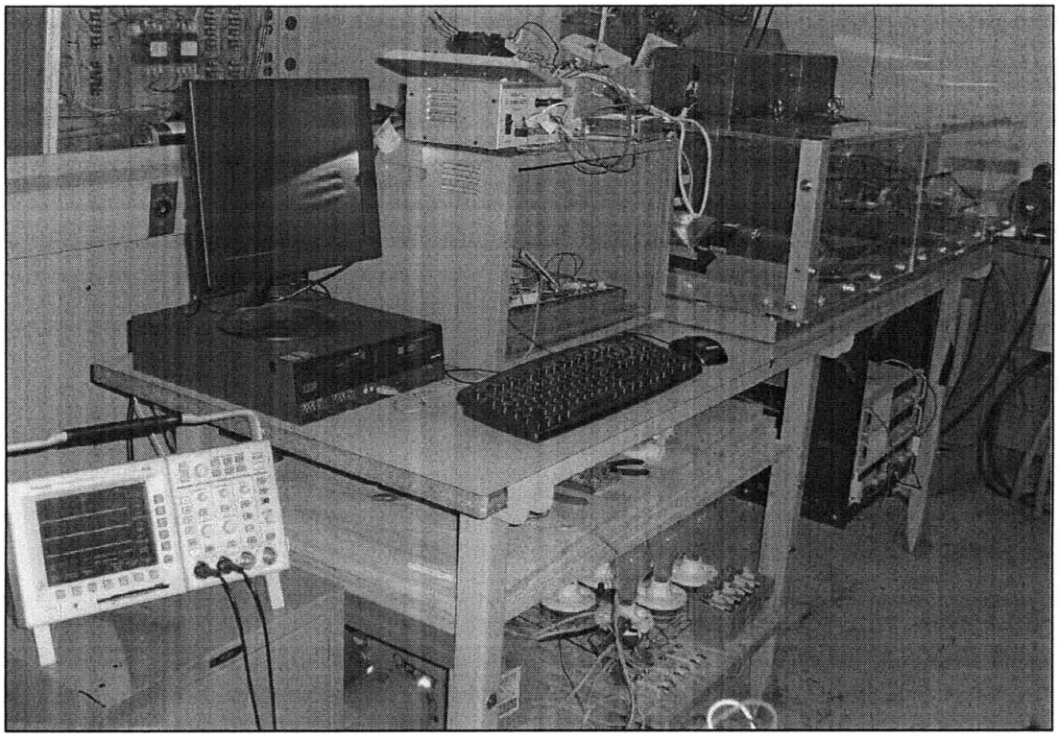


Figure 29 – Two-generator tabletop apparatus.

Split-plant power distribution systems allow each of the two generators to power separate loads. The only additional control required to emulate split-plant systems is another Proportional-Integral compensator to control the new generator apparatus. Synchronizing the generators, however, poses a much greater control problem. The next sections discuss how to control the two generators to safely synchronize and share loads.

3.2 Generator Synchronization Algorithm

Synchronizing the two generators allows for more power to be applied to the experimental loads. Additionally, synchronized generators make the hardware emulator a more exact scaled model for the DDG-51 power supply system [**8**]. This section discusses one method for synchronizing and load-balancing two generators of approximately equal strength. One generator already evenly loaded and stably controlled as shown by *GEN1* in Figure 30. This generator may be controlled with the Proportional-Integral LabView code used before. The remaining generator is controlled according to the

algorithm described below to allow for the safe connection of switches SA, SB, and SC, implemented using mechanical relays. It is important not to short nodes with different potentials together as this would cause inordinate current to flow between the generators. These circulating currents will certainly reduce the instantaneous power available from the generators and can damage the generators permanently. To ensure these nodes can be safely connected, there are four specific conditions that must be met. [**9**]

1. The root-mean-squared voltages of the two generators must be equal.
2. The generators must have the same phase sequence.
3. The generators must have the same phase angle.
4. The frequency of oncoming generator must be slightly higher than that of the already loaded generator.

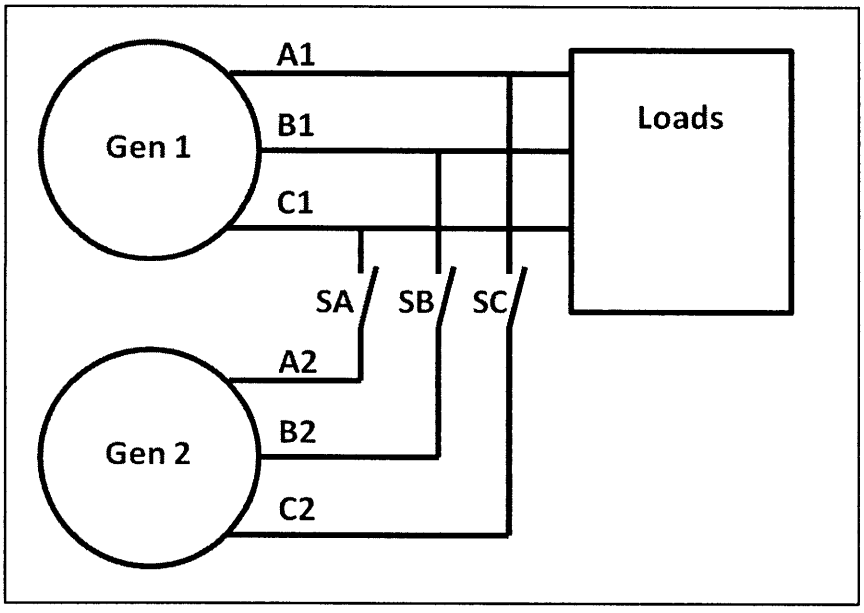


Figure 30 – Connections for synchronizing two generators [9].

Condition one ensures that the outputs of a highly energized generator do not connect to the outputs of a less energized generator. If this condition were not met, current would flow into the lesser

energized generator, backdriving the apparatus and probably destroying the power supplies that control the generators. This condition can be monitored by using root mean squared sensors or voltage transducers. The former would allow direct root mean squared comparison while the latter would require computation to compare the complete waveforms.

Condition two ensures that the sequence of peaking among the three phases is the same for both generators; if this condition is not met, excessive short-circuit currents would circulate between the two mismatched phases even if the last phase matched perfectly. This condition can be monitored by using a sinusoidal phase sensors or voltage transducers on the *A* and *B* outputs of each generator. Assuming there is no phase difference between *A1* and *A2*, the difference between *B1* and *B2* will be either 0^{o} in the case that both generators are ordered *ABC* or 180^{o} in the case that the generators differ *ABC* and *ACB*.

Condition three ensures minimal voltage difference between the nodes that will be connected when synchronizing is complete. This condition can be monitored by using sinusoidal phase sensors or voltage transducers on the outputs of one phase of each generator.

Condition four seems counterintuitive since the voltage waveforms at the nodes are supposed to be identical at the time of connection, which seems to suggest equal amplitude and frequency. A slight difference in frequency between the two machines can be approximated as an identical frequency with a slowly varying difference in phase as shown in Figure 31. This allows the phase between the outputs of the two generators to drift until it reaches an ideal point, at which point, switches *SA*, *SB*, and *SC* can be closed. As shown in the image, the variation of phase difference is slower when the frequencies are most similar thus the difference in frequency can define the speed of synchronization. The phase variation needs to be slow enough for the control system to recognize the state before it changes dramatically, but quick enough for the synchronization to happen in a timely manner. Since it is

unlikely that the two generators will ever be controlled to exactly the same frequency, it is probably unnecessary to command the prime movers to turn at different speed. The phase difference can be monitored as mentioned above. If the frequency difference is slight, the generators stabilize to a common frequency. [**9**]

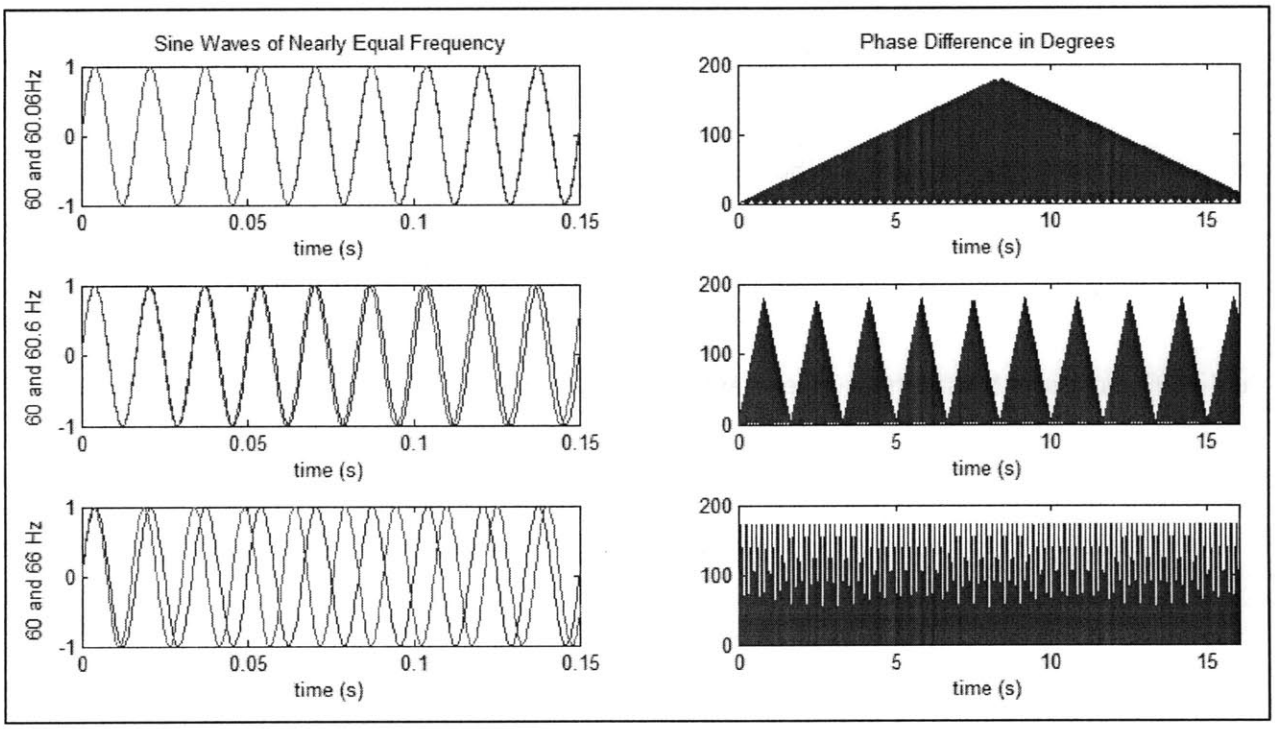


Figure 31 – Slight differences in frequency approximated as variable phase difference.

Once synchronized, if the generators do not balance the loads properly, at best the generators will not run at maximum efficiency and at worst they are in danger of coming out of synchronization. The generators should share the loads equally; the real and reactive power supplied by one generator must be the same as the real and reactive power supplied by the other. This requires an active control scheme.

To balance the generators, it is necessary to monitor the real and reactive power supplied by each of them. As long as the three phases are equally loaded, this can be done by monitoring the voltage and current waveforms of just one phase of each generator. The quantities that must be sensed

are real and reactive power. While no real and reactive power sensors are readily available, the fact that the waveforms are all sinusoidal suggests the use of arithmetic to derive the desired quantities from more easily measureable ones. In particular, Equation 7 and Equation 8 show the relationship between total power components and easily measureable quantities like phase current and voltage magnitude and phase angle. These calculations assume that the three phases of the generator are equally loaded; if this assumption is not true, then no easy linear relationship can be found between the power delivered by phase A and the power delivered by the entire generator [**10**].

Equation 7

$$P_{out} = 3V_A I_A \cos(\theta), \qquad \theta = phase\ difference\ betwen\ V_A\ and\ I_A$$

Equation 8

$$Q_{out} = 3V_A I_A \sin(\theta), \qquad \theta = phase\ difference\ betwen\ V_A\ and\ I_A$$

If Generator 1 is supplying less real power, the speed of its prime mover should be increased, since applying more mechanical power allows the generator to produce more electrical power. If *GEN1* is supplying less reactive power, the current through the field windings should be increased. The greater amplitude of the voltage waveforms increases reactive and real power without changing their ratio since the phase angle is unchanged. In order to adjust the system frequency or the system amplitude it is necessary to simultaneously adjust each generator's prime mover speed or field current.

The only quantities that need to be sensed for synchronizing the generators and load-balancing are frequency, phase order, phase angle, root mean squared voltage, and root mean squared current. All these quantities can be computed using arithmetic from the voltage and current waveforms ar two outputs of each generator. The already-implemented frequency and voltage root mean squared sensors are not sufficient for these measurements. The use of voltage and current transducers that recreate

scaled versions of the voltage and current waveforms allow computational devices to calculate the order, magnitude, and phase of the signals as discussed below.

3.3 Frequency Sensors

The speed of rotation of the generator shaft is directly proportional to the electrical frequency of the output waveforms. This means that a simple mechanical sensor that measures the speed of the shaft can be used to measure the frequency of the output waveforms. A Hall-Effect speed sensor with a 14 tooth sprocket is used to produce an open-collector signal that oscillates at fourteen times the frequency of the shaft. An additional pull-up resistor to five volts creates a 420Hz TTL signal for a 60Hz electrical frequency as shown in Equation 9 and Equation 10 where all frequencies are measured in hertz and *P* is the number of poles in one generator. In the generators used in this project, there are four poles.

Equation 9

$$f_{shaft} * 60\ s/min = \frac{120^0 * f_{electrical}}{P}$$

Equation 10

$$f_{hall} = 14 * f_{shaft}$$

The most obvious way to sense the frequency of this signal might be to connect the output of the Hall-Effect sensor to one of the digital bits on the K-8061 data bus. This would allow for constant computer supervision of the electrical frequency. Unfortunately, this scheme would require that the computer sample that bit at a frequency greater than $2f_{hall}$, the Nyquist frequency for this signal. This would load the computer down doing arithmetic to keep a constant frequency measurement, keeping it from performing the more complex calculations required to implement the feedback compensators used for control.

The previous signal conditioning circuit utilized a PIC microcontroller to eliminate this load on the primary computation device by calculating the TTL frequency of the signal generated by the Hall-Effect sensor. After calculating the frequency, the PIC circuit would use an embedded digital-to-analog converter to output a voltage proportional to the electrical frequency of the generator's output. One problem with this approach is in range and resolution. The PIC used is an eight-bit system. In order to output an analog signal with enough resolution to settle exactly on 60Hz, the code used requires minimum and maximum frequencies to sense [**7**]. This scheme reduces frequencies below the minimum to a zero volt output and frequencies above the maximum to a five volt output. While this may be acceptable for controlling steady-state systems, it almost ensures integrator wind-up if a feedback loop controls the generator from power-on or there is a large step change in load. Another problem with this sensor is the number of data conversions required; digital computation is performed, after which the signal is converted to an analog value, after which the signal is converted back to a digital value on the K-8061 board. These excess conversions reduce the resolution of the sensor even more.

In order to reduce the problems mentioned above, as well as to decrease the number of parts that need to be attached to the system, the frequency sensor was completely switched out for a new one. The new sensor is digital and thus must be wired to the K-8061 eight bit input data bus. The general idea is shown in abbreviated four bit form for in Figure 32.

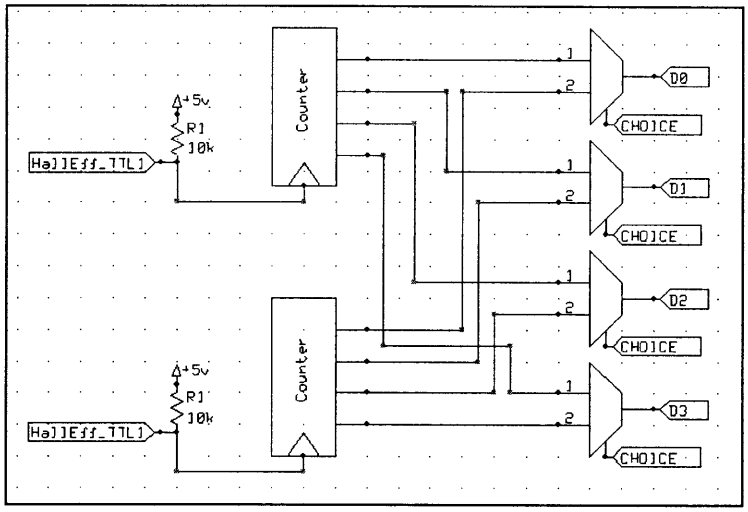


Figure 32 – Designed counter-multiplexer frequency sensor.

The output of each of the digital TTL Hall-Effect sensors is used as the clock input for a four bit counter, the most significant bit of which is used as the clock input for another four bit counter, thus creating an eight bit counter. Two such circuits are used, one for each generator. The eight bit outputs of the two counters are multiplexed such that a *CHOICE* bit will allow either counter to be connected to the input data bus. The counter reduces the required sampling frequency for calculating electrical frequency. By decreasing the frequency of the speed sensor signal from approximately 420Hz to 1.64Hz, this sensor reduces the required sampling rate for the primary computation device as shown in Figure 33. This figure shows the output of the Hall-Effect sensor on top and the eight bit output of the counter circuit on bottom. In steady state, each frequency sensor may be polled as infrequently as four times per second. During transient states, the frequency sensors can be polled more frequently without any loss in resolution since the resolution of the sensor circuit is as good as the resolution of the Hall-Effect sensor device. The current apparatus provides a resolution of about 7%. This resolution would suggest that even when both frequency sensor say they are running at 60Hz, they may be running at slightly different frequencies. In reference to Figure 31, the difference in frequency is slightly less than the second case. While this might allow for slow enough phase sensing, it may be better to further slow the

phase shift variation between the generators by increasing the resolution of the frequency sensors by using a sprocket with more than 14 teeth. Unfortunately this will also require more frequent sampling of the data bus.

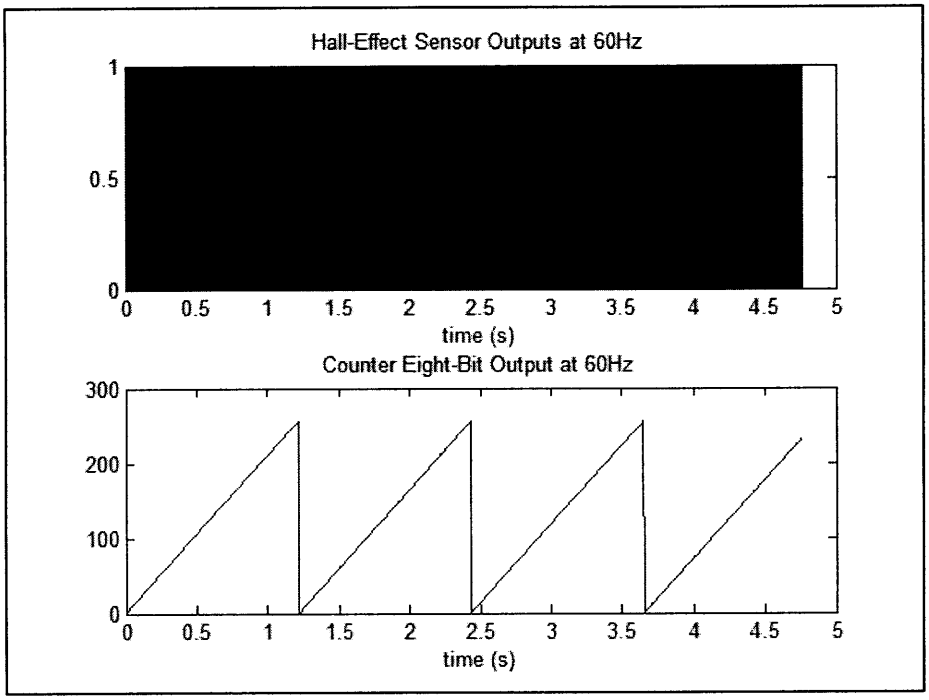


Figure 33- Decreased sampling rate with frequency sensor.

In comparison to the previous frequency sensor, the new sensor is simpler because it does not require any code. The new sensor also has increased range and resolution and does not lose accuracy by converting the data to and from analog and digital formats. This sensor was integrated with the voltage and current gain stages described in the next section in a sensor board detailed in Appendix D.

3.4 Voltage and Current Sensors

The remaining sensors required for control of synchronizing and balancing generators are those for phase difference, real power, and reactive power. In order to get these values, the actual sensors used will create scaled copies of the voltage and current waveforms. Calculation of magnitude and phase will occur in the primary computing device, rather than in peripheral analog or digital circuits.

The previous voltage sensor scaled and offset the generator output voltage waveforms with resistors and then input the signal to an analog root-mean-squared integrated circuit, the LTC1966. Finally, the output signal is filtered, amplified, and connected to an analog input channel on the K-8061 [**6**]. While this chip is very linear, correct within 1% over the range of inputs of interest, any error is compounded by the scaling and amplification stages. In addition, the output of this voltage sensor cannot be used to fulfill two of the three desired quantities: phase difference and current magnitude.

Voltage and current transducers allow for the accurate generation of scaled waveforms. The schematic showing the connections to the transducers is shown in Figure 35. The LV 25-P is a voltage transducer that connects across the generator output phase and produces an isolated scaled output signal. The voltage transducer requires a resistance across the input terminals that allows for a nominal 10mA current. This suggested the use of five 62kΩ resistors in parallel to keep from dissipating more than 0.25W per resistor at a nominal 120 VRMS. A measurement resistor is connected at the output so the nominal output voltage is 0.47 V. The LA 55-P is a current transducer that loops around the current-carrying cables for each phase. The current transducer also includes a measurement resistor. These two transducers are shown in Figure 34.

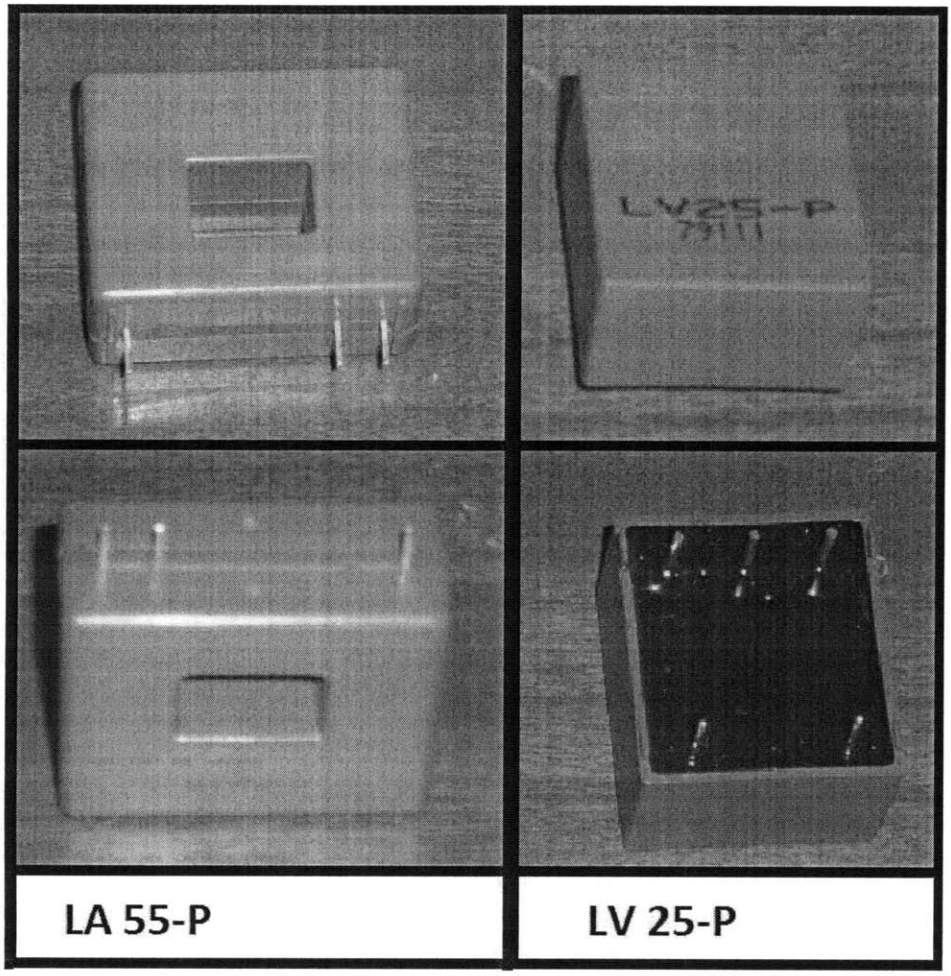


Figure 34 – Voltage and current transducers.

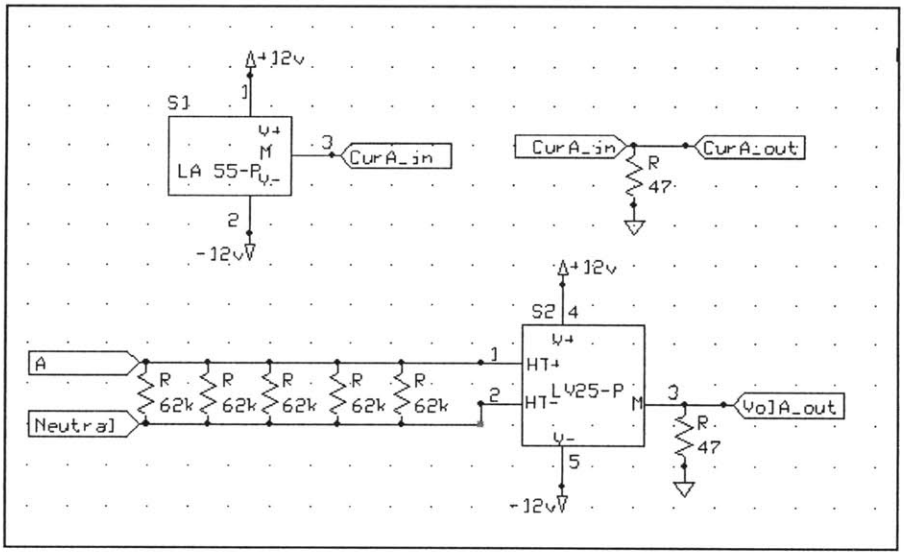


Figure 35 – Connections for voltage and current transducers.

The outputs of these two transducers are alternating current signals. In order to use these signals with a digital computation device, it is necessary to condition them to allow for the best possible analog-to-digital conversion. This necessitates that the signal remain between 0 and 3 volts. To retain the positive and negative sections of the alternating current signal, it is necessary to add a direct current offset of 1.5 volts. A schematic of the gain topology used is shown in **Error! Reference source not found.**.

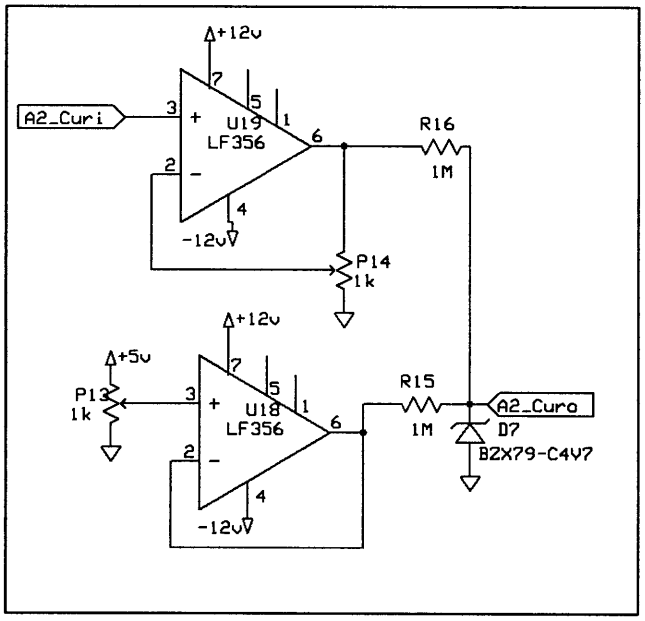


Figure 36– Gain topology for conditioning voltage and current signals.

This gain topology outputs a weighted average of the input signal and the direct current offset. A non-inverting operational amplifier gain stage increases the amplitude of each incoming signal. A non-inverting buffer stage contributes an offset voltage chosen using a potentiometer. This buffer keeps the offset voltage at its chosen value; without the buffer, the following resistors and diode might load down the output of the potentiometer, resulting in an offset different than the chosen value. The LF356 operational amplifier has a JFET input stage; this is preferable to the BJT input stages more commonly used because it pulls less current from the transducers. The outputs of the two op amps are averaged using low tolerance (0.1%) resistors; while an op amp adder could have been used, this stage would

increase error and slightly skew the addition. By averaging with resistors, the number of gain stages and the number of components are both reduced resulting in more exact signal conditioning as well as cheaper and smaller printed circuit boards.

The gain and offset are manually chosen with potentiometers in order to achieve exacting gain and offset. They are to be tweaked to near perfection when the board is build, then never touched again. Potentiometer *P1* should be set such that the voltage on the wiper is 3.0V in order to contribute a 1.5V direct current offset to the output. P2 should be set such that the nominal 120 VRMS signal fills half of the allowed space, a peak to peak voltage of 1.5V on the output.

This potentiometer scheme increases the load on the 5V linear regulator. With twelve 1kΩ potentiometers and four digital integrated circuits, the amount of current drawn from the 7805 linear regulator is approximately 0.96A as shown in Equation 11 and Equation 12. This is very near the maximum current that may be drawn from the linear regulator. This requires that the 7805 be outfitted with a heat sink and also suggests that a different linear regulator would be more apt. One possible solution is to use a linear regulator with a higher current draw, such as the L78S05CV which can produce 2A.

Equation 11

$$I = \frac{V}{R} + I_{74LS393} + I_{74LS147}$$

Equation 12

$$I_{5v} = \frac{5V * 12}{1k\Omega} + 2(0.15A) + 2(0.30A) = 0.96A$$

The zener diode on the output ensures that despite errant sensor readings or gain settings, the output remains in the acceptable range. Interestingly, a 3.0V zener diode is not used. Because of the large resistors used to average the SDC offset and amplified signal, very little current will ever be

allowed to flow though the diode. The maximum in this case is 24μA. If insufficient current flows though a zener, it does not break down at its proscribed voltage. This necessitates the use of a 4.7V zener diode which breaks down at 3.0V when the current through it is very low.

After physical construction, the gain stages were tested. Oscilloscope traces showing the results of these tests are listed below. Figure 37 shows correctly functioning signal conditioning. A 500mV signal is amplified, and then offset by 1.5V. The output signal remains between 0 and 3V for the safety of the following analog-to-digital converter. Figure 38 shows the amplified waveform saturating. This image shows the effect of the zener diode. Figure 39 shows voltage transducer outputs on the left and corresponding conditioned signals on the right.

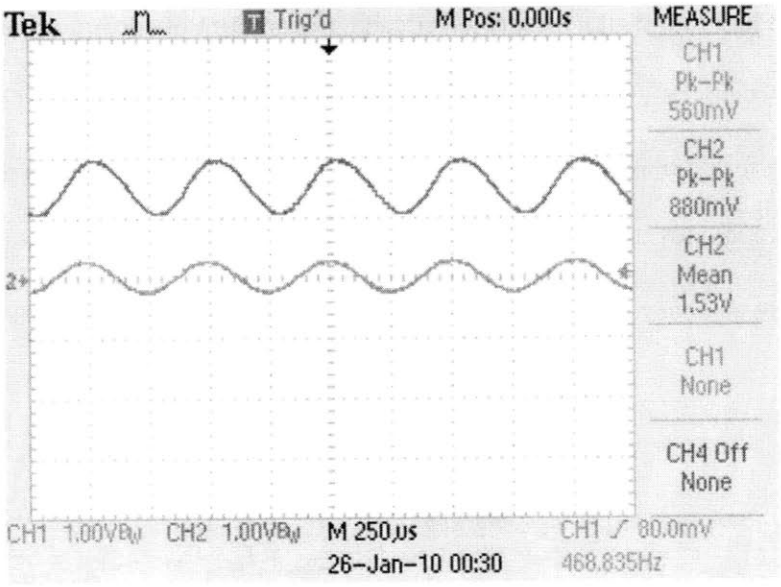


Figure 37 – Correct transducer signal conditioning.

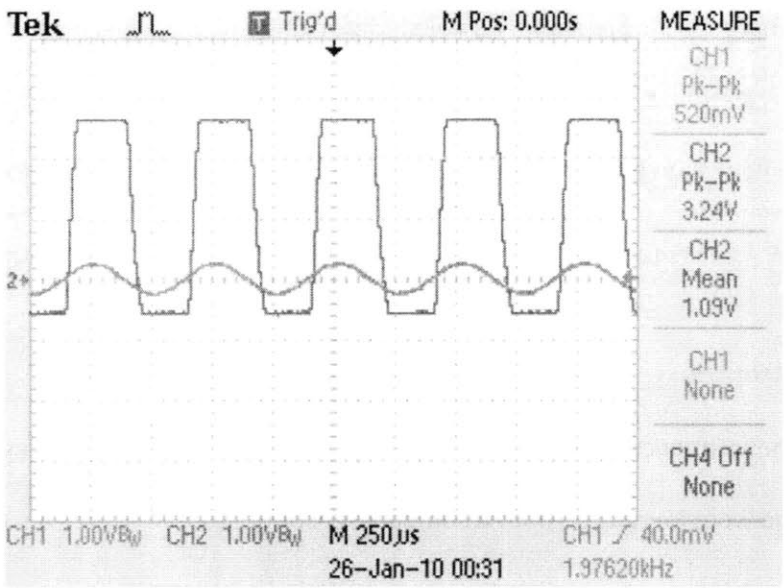


Figure 38 – Saturated transducer signal conditioning.

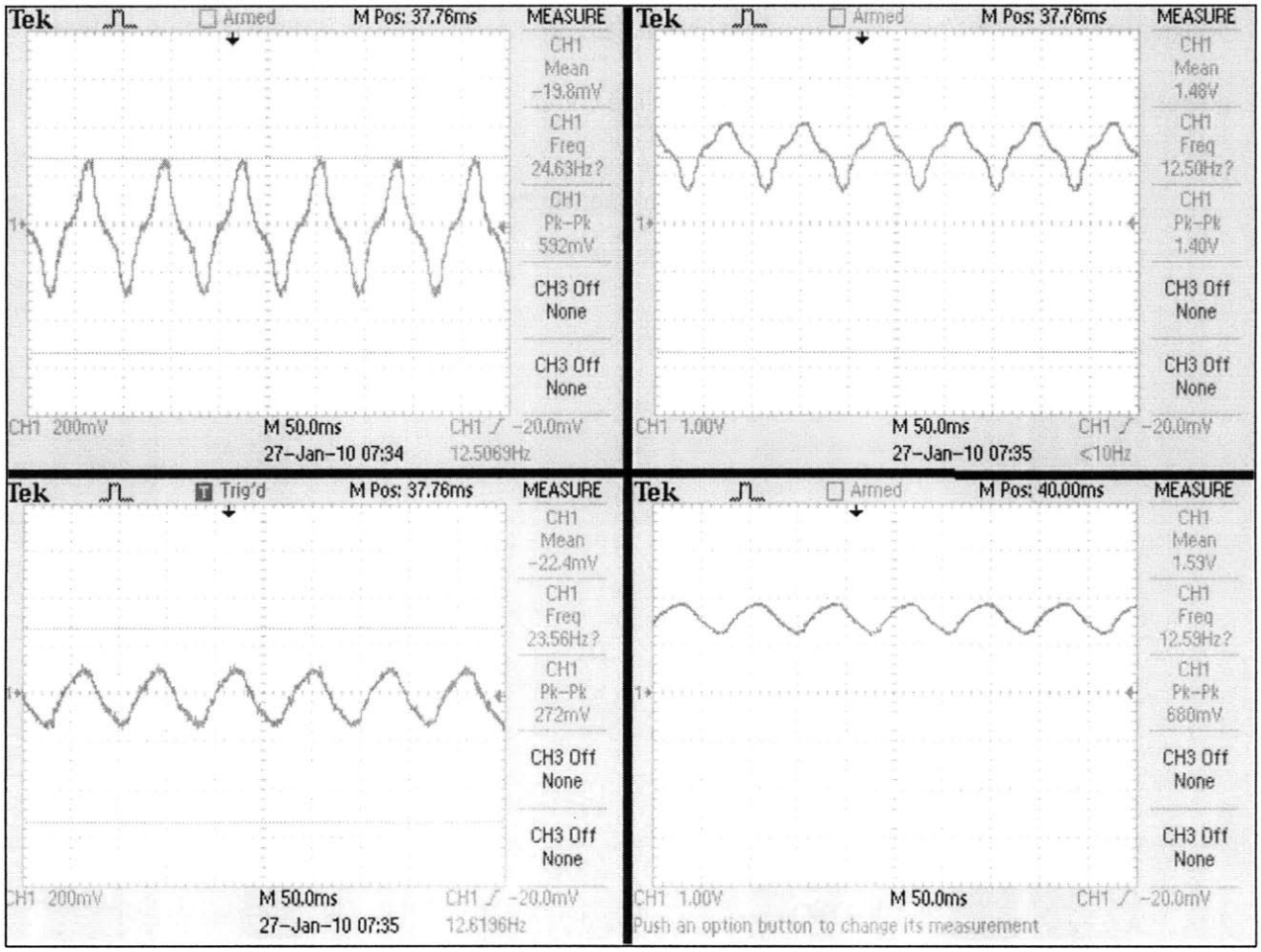


Figure 39 – Conditioning of real voltage transducer signals.

The signal saturates just under 3 volts. Unfortunately, the signal voltage is not cut off at exactly 0 volts, but this cannot be helped with a passive component like a zener. Perhaps the implementation of this gain topology with single-rail operational amplifiers would eliminate this problem.

3.5 Generator Use

Both the frequency sensors and the voltage and current transducer signal conditioners described above were integrated onto a compact sensor board that acts as an interface between the outputs of the generator emulator system and any digital computation devices. In addition to the K-8061 and the computer, another particular device with which the emulator must interface is a board designed and coded by Zack Remscrim that uses a field programmable gate array (FPGA) to calculate power factor which can easily be altered to calculate voltage and current magnitude and phase.

The final version of the generator emulator sensor printed circuit board is shown in Figure 39.

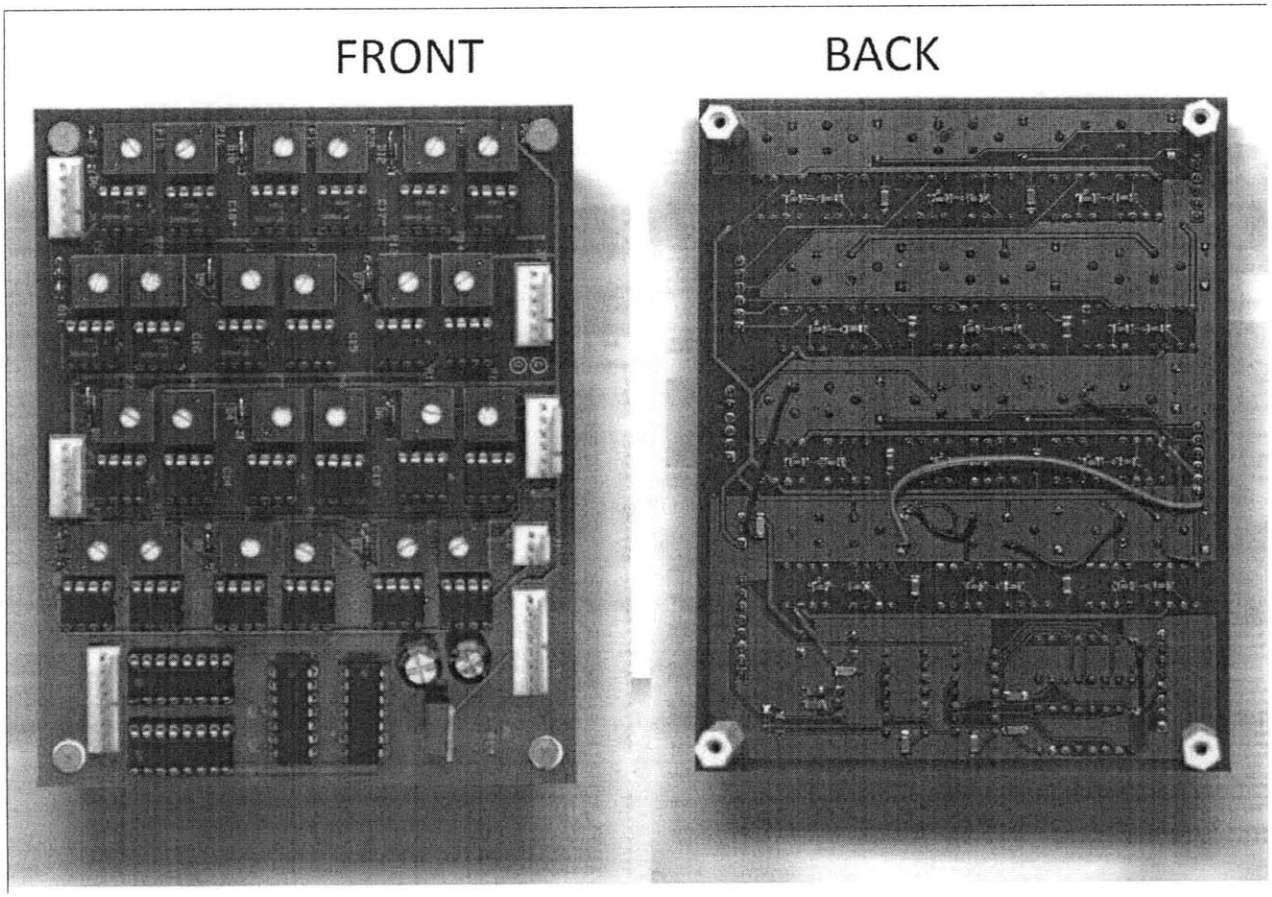


Figure 40 – Sensor conditioning printed circuit board.

4 Conclusions and Future Work

The robot driver board allows the R31-JP microcontroller system to be interfaced with two-motor mobile robots. The driver successfully drives both direct current and stepper motors. Additionally, the driver conditions signals picked up with the resonant-frequency amplifiers and connects an eight channel analog-to-digital converter to the microcontroller system. These circuit elements allow for extensibility and versatility. The Ethernet interface will allow for the translation of 8051 assembly code instructions to and from UDP packets. To meet this goal, it will be necessary to test the Ethernet code from [**2**]. Once this circuit block is tested and shown to successfully send packets of information across wired Ethernet, a worthwhile extension might be wireless Ethernet. The implementation of a wireless network that allows for the connection of embedded-Ethernet robots will be a nontrivial pursuit as it will necessitate some special work with network addressing and security.

The generator emulator sensor printed circuit board allows the generator system to interface with digital sensing and control boards. This will allow for the calculation of real-time values such as phase difference between the voltage and current as well as the magnitude of real and reactive power supplied by the generator. These capabilities make this board useful for the process of synchronizing and load balancing for the two generators. They will aid in the implementation of the algorithms described above. Once the generators can be controlled to share loads efficiently, the system can be used to power high-draw loads or to demonstrate experimental power distribution systems that will lead to increases in the safety and efficiency of shipboard power distribution systems.

References

[1] Norbert Doerry, *Next Generation Integrated Power Systems (NGIPS) for the Future Fleet.* Baltimore, USA: Naval Sea Systems Command, 2009.

[2] Sencer Yeralan and Helen Emery, *Internetworking Eight-Bit Microcontrollers*, 1st ed., Eginhard J. Muth, Ed.: Rigel Press, 2003.

[3] Sabrina Neuman, *6.UR Report: A New Ethernet Lab for 6.115.*, 2009.

[4] New Japan Radio Co., Ltd., *"Dual H Bridge Driver," NJM2670 datasheet.*

[5] Intel, *"CHMOS Programmable Interval Timer," 82C54 datasheet.*, Oct. 1994.

[6] Vanessa Esch, "Control System for a DC Motor-Generator System for Operation as a Shipboard Gas Turbine Generator Model," Massachusetts Institute of Technology, Cambridge, MA, 6.UAP 2009.

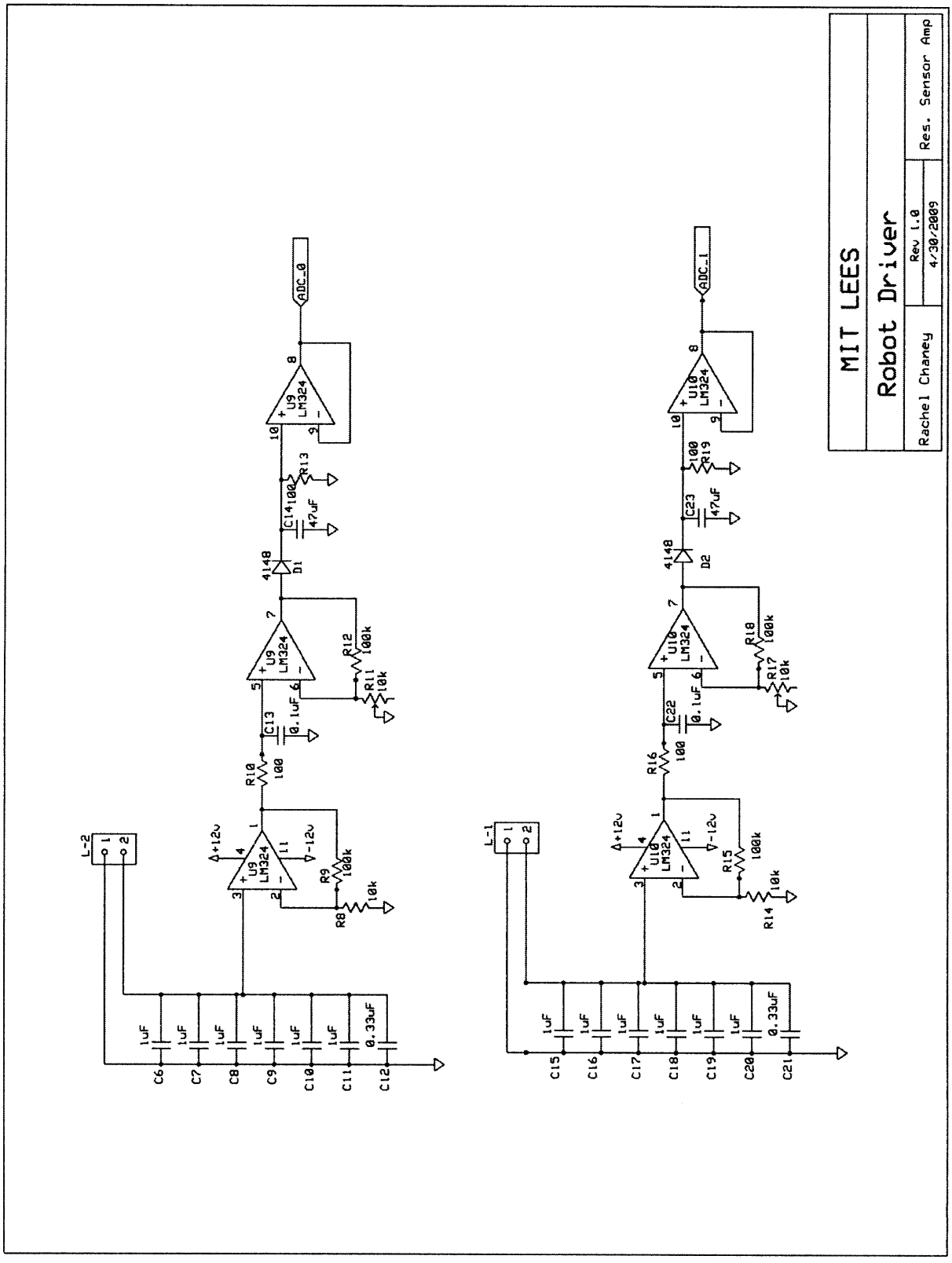
[7] Jacob Osterberg, "Generator Feedback Electronic Hardware Specifications," Massachusetts Institute of Technology, Cambridge, MA, Undergraduate Research Opportunities Program 2009.

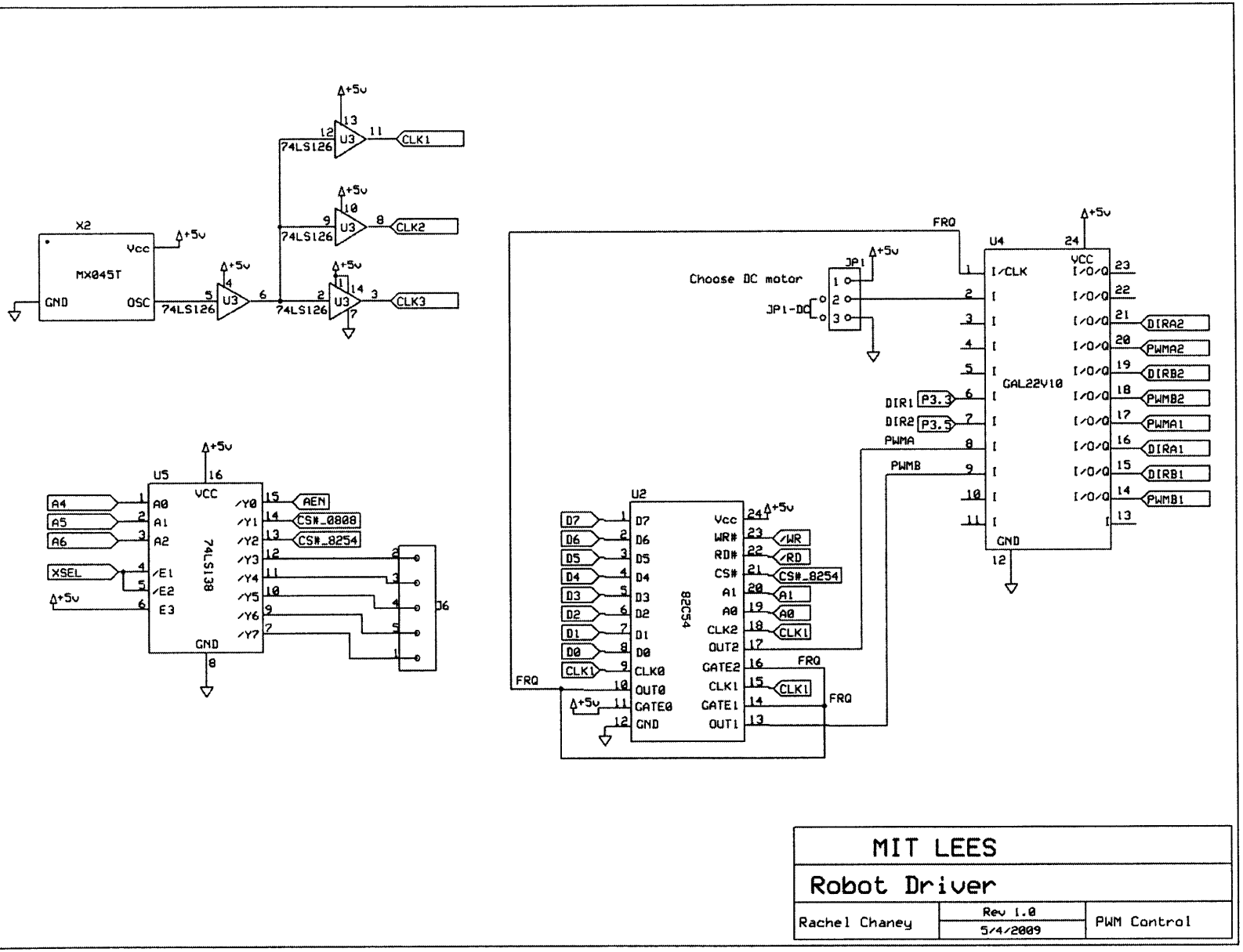
[8] Gregory L. Elkins, *Hardware Model of a Shipboard Generator.* M.S. thesis, Massachusetts Institute of Technology, 2009.

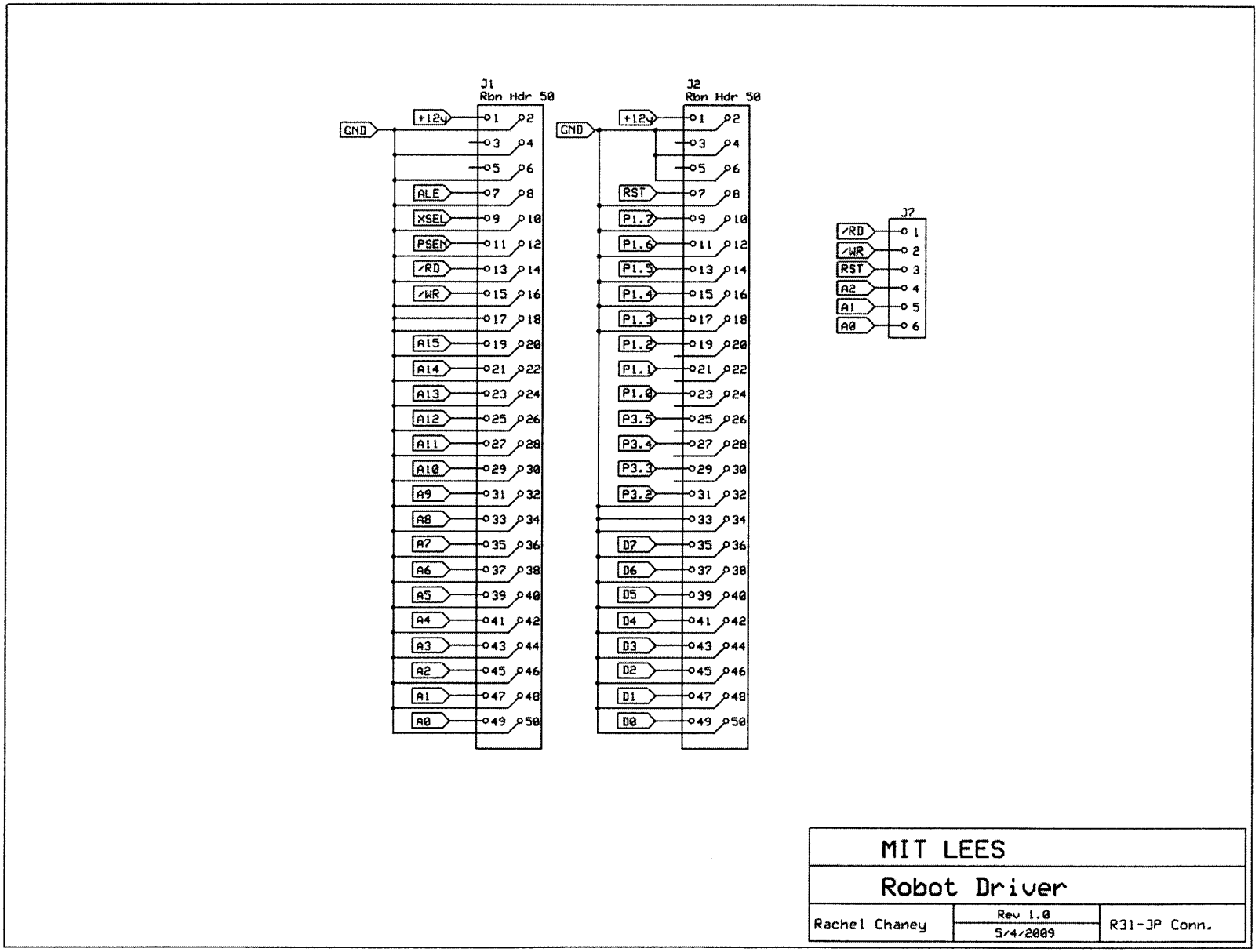
[9] Stephen J. Chapman, *Electric Machinery and Power System Fundamentals.* New York City, NY, USA: McGraw-Hill Higher Education, 2002.

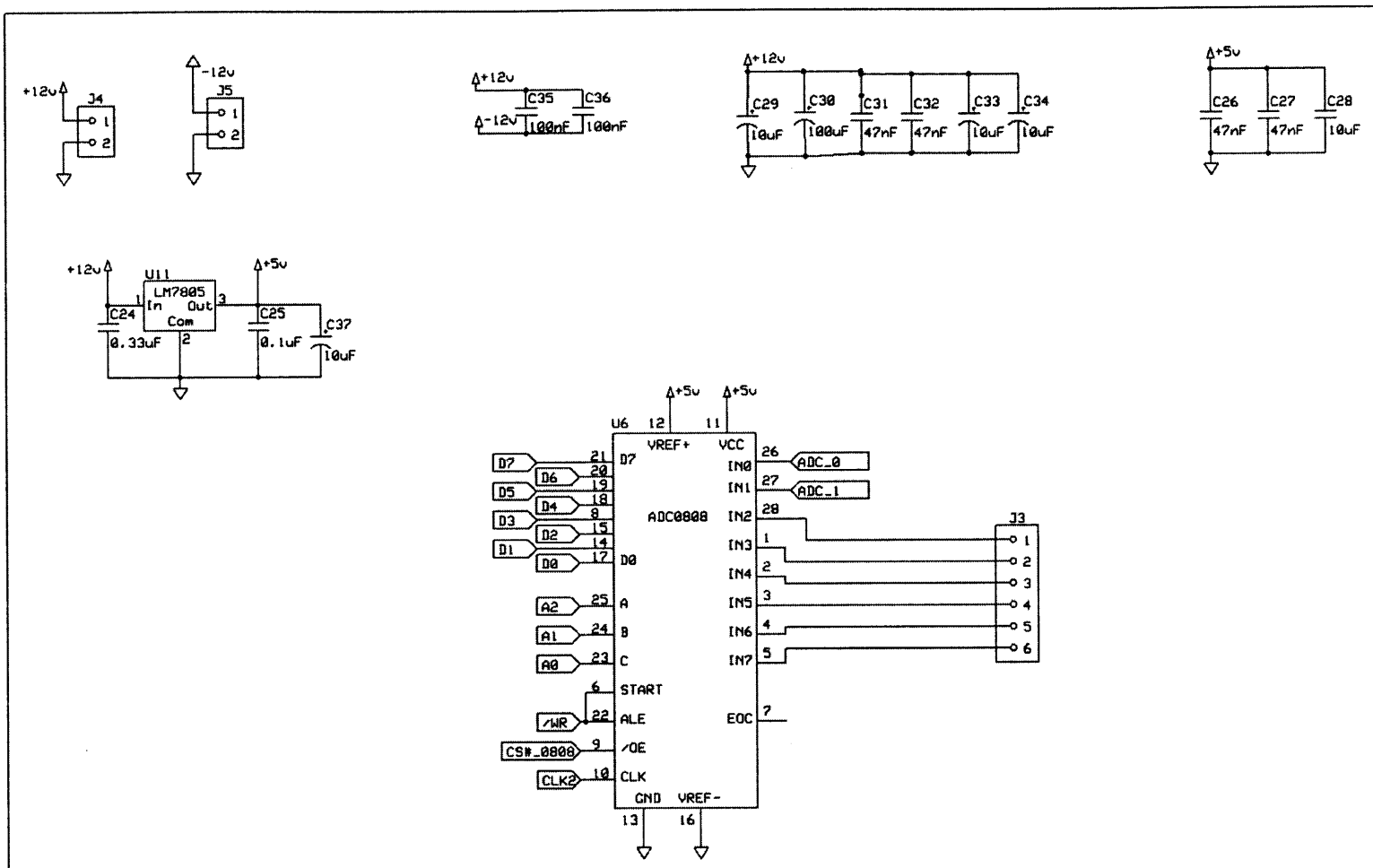
[10] James L. Kirtley, *6.685 Electric Machines.* Cambridge, MA, USA: Massachusetts Institute of Technology, 2005.

Appendix A: Robot driver schematic and bill of materials

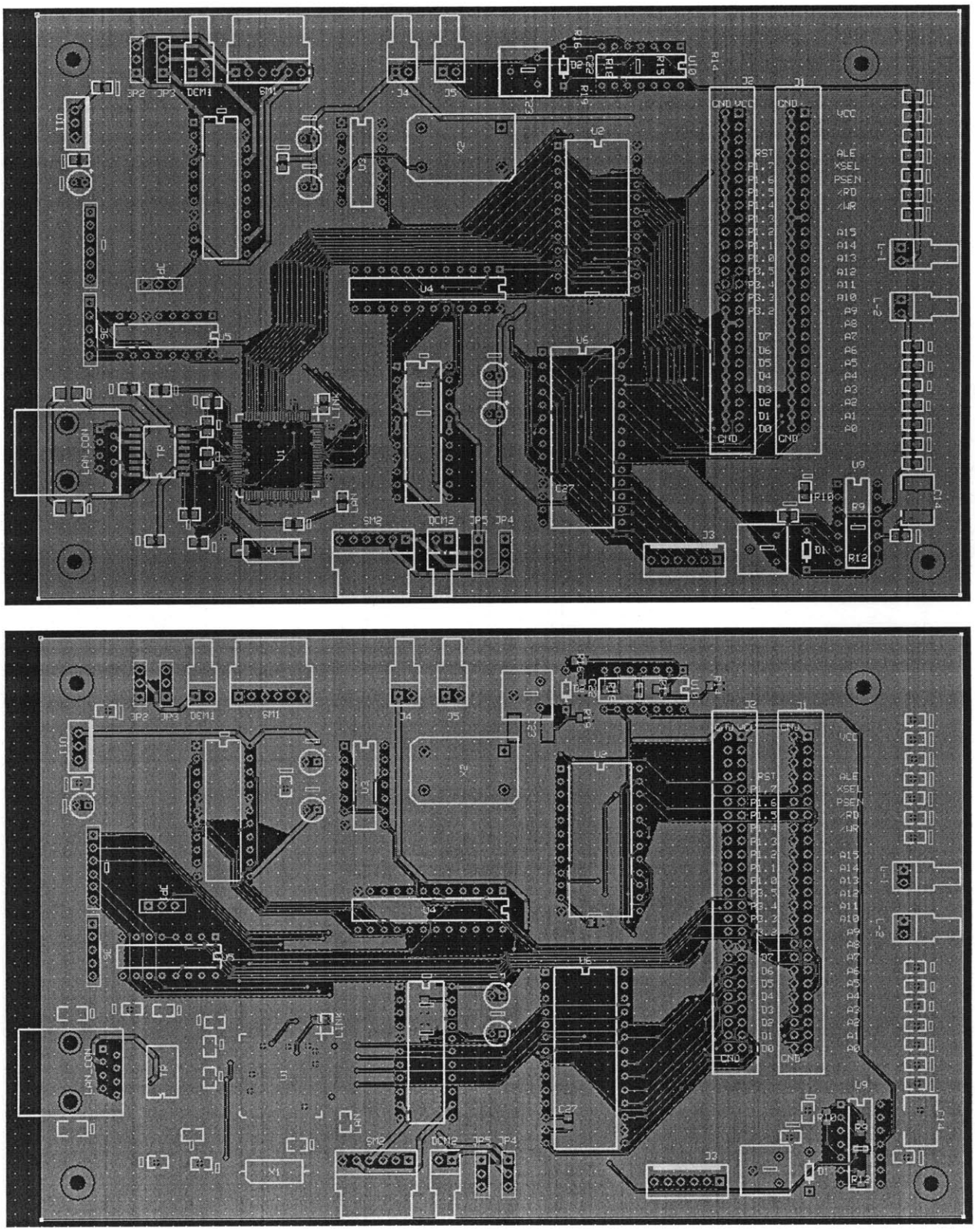



X2
MX045T
Vcc
GND
OSC
+5v
74LS126
U3
CLK1
CLK2
CLK3
U5
74LS138
A0
A1
A2
/E1
/E2
E3
/Y0
/Y1
/Y2
/Y3
/Y4
/Y5
/Y6
/Y7
VCC
GND
A4
A5
A6
XSEL
AEN
CS#_0808
CS#_8254
J6
U2
82C54
D7
D6
D5
D4
D3
D2
D1
D0
CLK0
OUT0
GATE0
Vcc
WR#
RD#
CS#
A1
A0
CLK2
OUT2
GATE2
CLK1
GATE1
OUT1
/WR
/RD
FRQ
JP1
Choose DC motor
JP1-DC
U4
GAL22V10
/CLK
I/O/Q
I
GND
DIR1
DIR2
PWMA
PWMB
P3.3
P3.5
DIRA2
PWMA2
DIRB2
PWMB2
PWMA1
DIRA1
DIRB1
PWMB1
MIT LEES
Robot Driver
Rachel Chaney
Rev 1.0
5/4/2009
PWM Control


J1
Rbn Hdr 50
+12V
GND
ALE
XSEL
PSEN
/RD
/WR
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
J2
Rbn Hdr 50
+12V
GND
RST
P1.7
P1.6
P1.5
P1.4
P1.3
P1.2
P1.1
P1.0
P3.5
P3.4
P3.3
P3.2
D7
D6
D5
D4
D3
D2
D1
D0
J7
/RD
/WR
RST
A2
A1
A0
MIT LEES
Robot Driver
Rachel Chaney
Rev 1.0
5/4/2009
R31-JP Conn.



MIT LEES		
Robot Driver		
Rachel Chaney	Rev 1.0 6/19/2009	Power, ADC


JP2
JP3
DCM1
SM1
J4
J5
U10
R16
D2
C22
R15
R14
J2
J1
U2
X2
U3
U4
U5
U6
U1
JP1
J6
LAN_CON
TP
U7
LAN
SM2
DCM2
JP5
JP4
X1
J3
U9
R10
R9
R12
D1
C27
C24
L-1
L-2
VCC
GND
RST
ALE
XSEL
PSEN
/RD
/WR
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
P1.7
P1.6
P1.5
P1.4
P1.3
P1.2
P1.1
P1.0
P3.5
P3.4
P3.3
P3.2

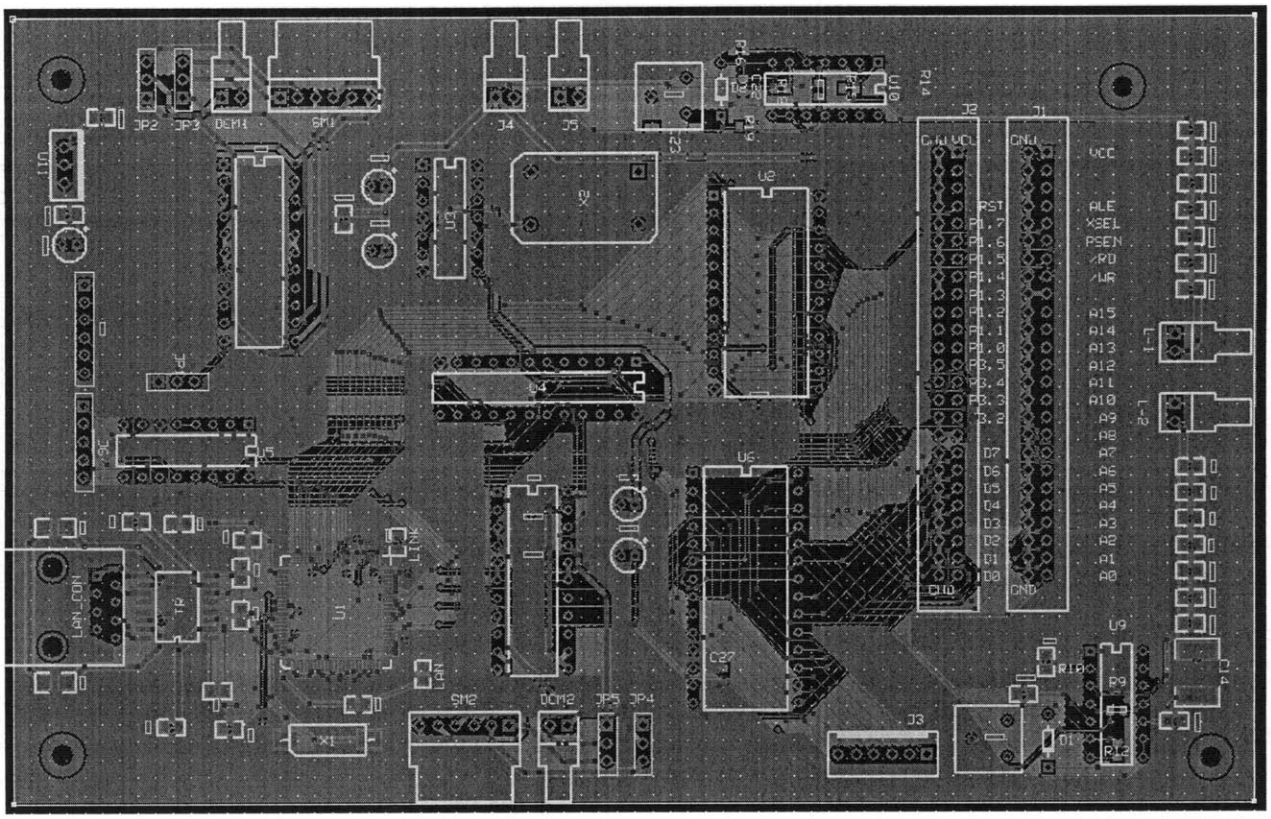


Table 4– Robot Driver Bill of Materials

Part No.	Value	Footprint	Company	Product No.
C1	68pF	SMT0805	Digikey	PCC680CGCT-ND
C2	0.1uF	SMT0805	Digikey	PCC1864CT-ND
C3	0.1uF	SMT0805	Digikey	PCC1864CT-ND
C4	4.7nF,600V	SMT1206	Digikey	PCC2294CT-ND
C5	4.7nF,600V	SMT1206	Digikey	PCC2294CT-ND
C6	1uF	SMT0805	Digikey	587-1291-1-ND
C7	1uF	SMT0805	Digikey	587-1291-1-ND
C8	1uF	SMT0805	Digikey	587-1291-1-ND
C9	1uF	SMT0805	Digikey	587-1291-1-ND
C10	1uF	SMT0805	Digikey	587-1291-1-ND
C11	1uF	SMT0805	Digikey	587-1291-1-ND
C12	0.33uF	SMT0805	Digikey	399-1165-1-ND
C13	0.1uF	SMT0805	Digikey	PCC1864CT-ND
C14	47uF	SMT2220	Digikey	445-3486-1-ND
C15	1uF	SMT0805	Digikey	587-1291-1-ND
C16	1uF	SMT0805	Digikey	587-1291-1-ND
C17	1uF	SMT0805	Digikey	587-1291-1-ND

C18	1uF	SMT0805	Digikey	587-1291-1-ND
C19	1uF	SMT0805	Digikey	587-1291-1-ND
C20	1uF	SMT0805	Digikey	587-1291-1-ND
C21	0.33uF	SMT0805	Digikey	399-1165-1-ND
C22	0.1uF	SMT0805	Digikey	PCC1864CT-ND
C23	47uF	SMT2220	Digikey	445-3486-1-ND
C24	0.33uF	SMT0805	Digikey	399-1165-1-ND
C25	0.1uF	SMT0805	Digikey	PCC1864CT-ND
C26	47nF	SMT0805	Digikey	478-3771-1-ND
C27	47nF	SMT0805	Digikey	478-3771-1-ND
C28	10uF	RAD2mm	Digikey	P980-ND
C29	10uF	RAD2mm	Digikey	P980-ND
C30	100uF	RAD2mm	Digikey	565-1059-ND
C31	47nF	SMT0805	Digikey	478-3771-1-ND
C32	47nF	SMT0805	Digikey	478-3771-1-ND
C33	10uF	RAD2mm	Digikey	P980-ND
C34	10uF	RAD2mm	Digikey	P980-ND
C35	100nF	SMT0805	Digikey	PCC1864CT-ND
C36	100nF	SMT0805	Digikey	PCC1864CT-ND
C37	10uF	RAD2mm	Digikey	P980-ND
DCM1	2 pin	.1" KK, rt ang	Digikey	WM4300-ND
DCM2	2 pin	.1" KK, rt ang	Digikey	WM4300-ND
J1	Rbn Hdr 50		Digikey	MHB50K-ND
J2	Rbn Hdr 50		Digikey	MHB50K-ND
J3	6 pin	screw term	Digikey	A98337-ND
J4	2 pin	.1" KK, rt ang	Digikey	WM4300-ND
J5	2 pin	.1" KK, rt ang	Digikey	WM4300-ND
J6	5 pin	.1"KK, rt ang	Digikey	SAM1050-50-ND
J7	6 pin	header (m)	Digikey	SAM1050-50-ND
JP1	3 pin	header (m)	Digikey	SAM1050-50-ND
JP2	3 pin	header (m)	Digikey	SAM1050-50-ND
JP3	3 pin	header (m)	Digikey	SAM1050-50-ND
JP4	3 pin	header (m)	Digikey	SAM1050-50-ND
JP5	3 pin	header (m)	Digikey	SAM1050-50-ND
JP1-DC	2 pin	jumper	Digikey	S9001-ND
JP2-DC	2 pin	jumper	Digikey	S9001-ND
JP3-DC	2 pin	jumper	Digikey	S9001-ND
JP4-DC	2 pin	jumper	Digikey	S9001-ND
JP5-DC	2 pin	jumper	Digikey	S9001-ND
L-1	2 pin	.1" KK, rt ang	Digikey	WM4300-ND
L-2	2 pin	.1" KK, rt ang	Digikey	WM4300-ND
LAN	Orange LED	SMT0805	Digikey	160-1413-1-ND
LAN_CON	RJ45	Ethernet	Mouser	538-43249-8104
LINK	Green LED	SMT0805	Digikey	160-1414-1-ND

R1	100,1%	SMT0805	Digikey	RHM100CRCT-ND
R2	24.9,1%	SMT0805	Digikey	RHM24.9CRCT-ND
R3	24.9,1%	SMT0805	Digikey	RHM24.9CRCT-ND
R4	4.7k	SMT0805	Digikey	RHM4.70KCRCT-ND
R5	4.99k,1%	SMT0805	Digikey	RHM4.99KCRCT-ND
R6	680	SMT0805	Digikey	RHM680CRCT-ND
R7	680	SMT0805	Digikey	RHM680CRCT-ND
R8	10k	SMT0805	Digikey	RR12P10.0KDCT-ND
R9	100k	SMT0805	Digikey	RR12P100KDCT-ND
R10	100	SMT0805	Digikey	RHM100CRCT-ND
R11	10k	3386F	Digikey	
R12	100k	SMT0805	Digikey	RR12P100KDCT-ND
R13	100	SMT0805	Digikey	RHM100CRCT-ND
R14	10k	SMT0805	Digikey	RR12P10.0KDCT-ND
R15	100k	SMT0805	Digikey	RR12P100KDCT-ND
R16	100	SMT0805	Digikey	RHM100CRCT-ND
R17	10k	3386F	Digikey	
R18	100k	SMT0805	Digikey	RR12P100KDCT-ND
R19	100	SMT0805	Digikey	RHM100CRCT-ND
SM1	6 pin	.1" KK, rt ang	Digikey	WM4304-ND
SM2	6 pin	.1" KK, rt ang	Digikey	WM4304-ND
TR	E2023NL	SOIC-16	Mouser	673-E2023NL
U1	CS8900A-CQZ	QFP-100	Digikey	598-1128-ND
U2	82C54	DIP-24	Digikey	CP82C54-ND
U3	74LS126	DIP-14	Digikey	296-3643-5-ND
U4	GAL22V10	DIP-24	Digikey	ATF-22V10CQ-15PC-ND
U5	74LS138	DIP-16	Mouser	595-SN74LS138N
U6	ADC0808	Dip-28	Digikey	ADC0808CNN/NOPB
U7	NJM2670	Dip-22 (0.4")	Digikey	NJM2670D2#-ND
U8	NJM2670	Dip-22 (0.4")	Digikey	NJM2670D2#-ND
U9	LM324	DIP-14	Digikey	LM324NFS-ND
U10	LM324	DIP-14	Digikey	LM324NFS-ND
U11	LM7805	TO-220	Mouser	512-LM7805CT
X1	20MHz	HC49SD SMD	Mouser	559-FOXSD200-20-LF
X2	MX045T	Full Size	Digikey	X104-ND

Appendix B: GAL22V10 code for Robot Driver

```
Name          robot0;
Partno        none;
Date          07/15/2009;
Revision      01;
Designer      Rachel Chaney;
Company       MIT LEES;
Assembly      none;
Location      none;
Device        G22V10;

/* This program uses the gal22v10 to drive a stepper motor or a dc motor */
/* choose which kind of motor in hardware with input CHOICE (1=stepper, 0=dc) */
/* control direction by setting/changing dir1 and dir2 in software */

/** Inputs **/
pin 1 = CLK;
pin 2 = choice;
/* pins 3-5 are unused input pins */
pin 6 = dir1;
pin 7 = dir2;
pin 8 = pwm1;
pin 9 = pwm2;

/** Outputs **/
pin 14 = out1;
pin 15 = out2;
pin 16 = out3;
pin 17 = out4;

pin 21 = out5;
pin 20 = out6;
pin 19 = out7;
pin 18 = out8;

/* pins 22-3 are unused output pins */

/** Logic **/

/* stepper motor logic */
out1.d = choice & (dir1 & (!out2 & !out3 & out4) # !dir1 & (out2 & !out3 & !out4));
out2.d = choice & (dir1 & (out1 & !out3 & !out4) # !dir1 & (!out1 & out3 & !out4));
out3.d = choice & (dir1 & (!out1 & out2 & !out4) # !dir1 & (!out1 & !out2 & out4));
out4.d = choice & (dir1 & (!out1 & !out2 & out3) # !dir1 & (out1 & !out2 & !out3) # (!out1
& !out2 & !out3 & !out4));

out5.d = choice & (dir2 & (!out6 & !out7 & out8) # !dir2 & (out6 & !out7 & !out8));
out6.d = choice & (dir2 & (out5 & !out7 & !out8) # !dir2 & (!out5 & out7 & !out8));
out7.d = choice & (dir2 & (!out5 & out6 & !out8) # !dir2 & (!out5 & !out6 & out8));
out8.d = choice & (dir2 & (!out5 & !out6 & out7) # !dir2 & (out5 & !out6 & !out7) # (!out5
& !out6 & !out7 & !out8));

/* dc motor logic */

APPEND out4.d = !choice & pwm1;
APPEND out3.d = !choice & dir1;
APPEND out6.d = !choice & pwm2;
APPEND out5.d = !choice & dir2;
```

Appendix C: Ethernet Functional Block Details

Sabrina Neuman
sneuman@mit.edu

IAP 2009
Professor Steven B. Leeb

Drafted: 1/26/2009

6.UR Report:

A New Ethernet Lab for 6.115

INTRODUCTION

The Microcomputer Project Laboratory (6.115) is an Institute and Department Lab that teaches students how to design intelligent and effective systems involving microcontrollers. The class has lectures that are closely related to five large laboratory activities, and a final project to conclude the semester. These lectures and lab activities explore different applications for microcontrollers, and are selected to be highly relevant to useful real-world systems.

Currently, the labs cover topics such as motor control, lamp ballasts, speaker systems, and touch-tone telephone dialing.

A new lab activity for 6.115 related to Ethernet and Internet Protocol would greatly enrich the experience of students taking the class. They would learn to implement a complicated set of procedures on microcontrollers, and enjoy the satisfaction of having their lab kit communicate on a Local Area Network (LAN), or even the Internet!

PROGRESS MADE DURING IAP 2009

During the IAP term 2009, I designed and began building/testing a new piece of hardware for the proposed 6.115 Ethernet lab activity. The board I created should interface smoothly with the existing 6.115 Lab Kit hardware, and allow students to connect their 8051 microcontroller to a LAN or the Internet, via an Ethernet cable.

I assessed off-the-shelf options for microcontroller Ethernet hookup, but found that no existing products had the Ethernet controller chip (Cirrus Logic's CS8900A) configured for the communication protocol of the 8051 microcontroller setup used in 6.115. The closest products I could find seemed to only work with a dedicated bus, because they provided no "Chip Select" signal to turn on and off bus communication to the chip. This is unacceptable for the 8051 setup in 6.115 because the same bus that would be used to communicate with the Ethernet chip board is also used for the 8051's external ROM and RAM. Thus, with no ability to disable bus communication from the Ethernet chip board at times when the 8051 had to access its external memory, there would be permanent "bus contention" where the Ethernet chip would attempt to communicate via the bus simultaneously with the ROM and RAM--causing nonsense signals to be sent back and forth.

The most practical alternative was to lay out my own CS8900A board, which I did in ExpressPCB, an simple printed circuit board (PCB) layout software. For my design, I made a breadbord-mating PCB in the style of the Olimex CS8900A-H (a board with the aforementioned "bus contention" issues), but with an available "Chip Select" signal. The result was more akin to the example schematic of a CS8900A chip configured for 8-bit mode as shown in the CS8900A Application Note AN181.

The resulting schematic and PCB layout of my board-- which I have tentatively named "EtherNed"--

are pictured below in Figures 1 and 2, respectively.

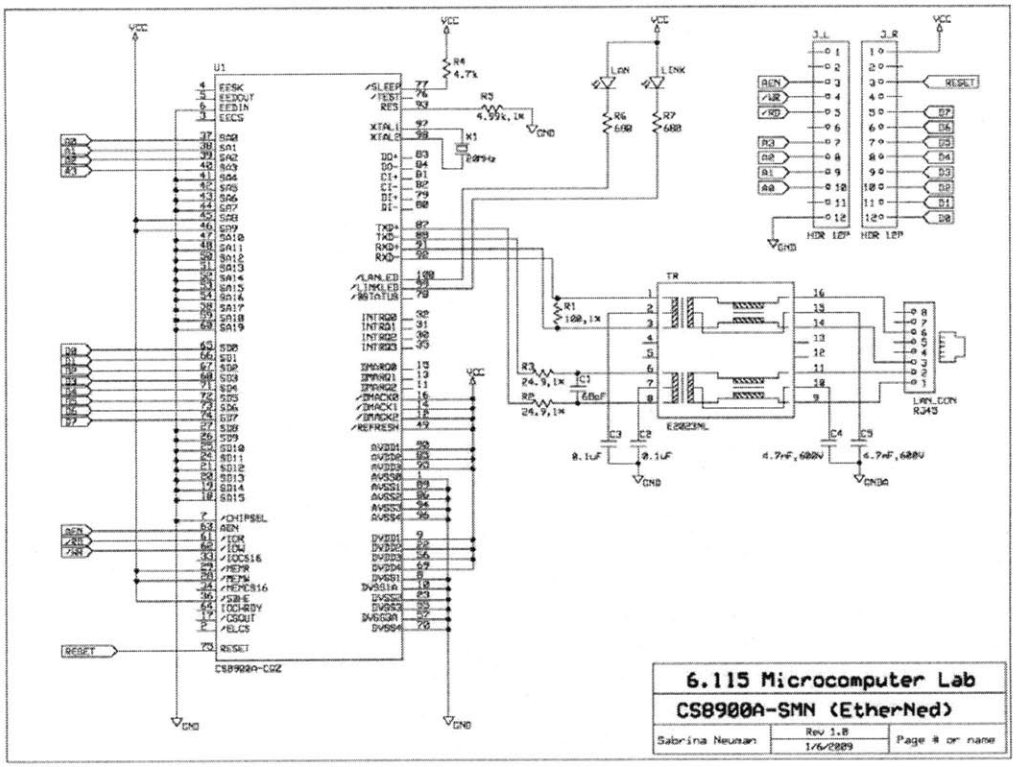


Figure 1: Ethernet Board Schematic

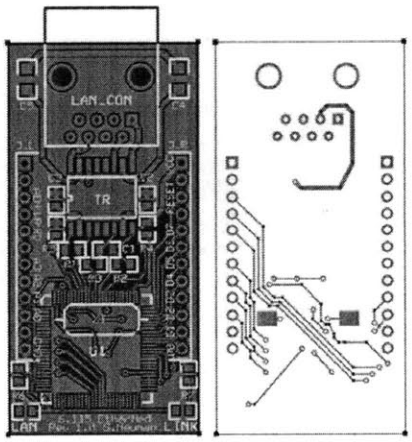


Figure 2: Ethernet Board PCB Layout. Top Layer (left). Bottom Layer (right).

After I designed my board, I ordered the PCB and parts. Once those arrived, I began painstakingly

assembling a board. I found the 100-pin CS8900A chip to be challenging to solder. It took many attempts to successfully place the chip, solder it, and then remove excess solder material that shorted various pins together.

Eventually, with the advice and assistance of some of my colleagues, I assembled a complete board, shown below in Figure 3.

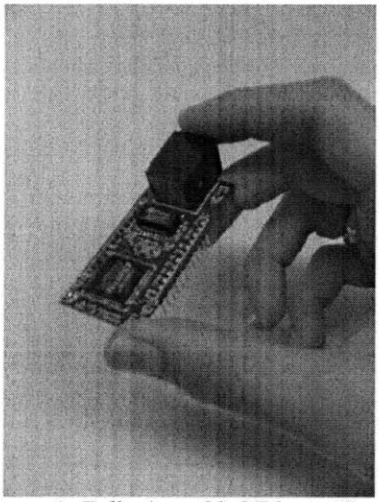


Figure 3: Fully-Assembled Ethernet Board

The board has been thoroughly tested with a multimeter, and all of the electrical and mechanical connections appear correct.

NEXT STEPS

Now that the board has been assembled and the connections have been checked, the next step is to begin testing whether the board works as intended. This will involve wiring up the board to a standard 6.115 Lab Kit and testing with a very simple piece of code if the 8051 microcontroller board can successfully communicate with the CS8900A chip on board.

Once it can be established that the Lab Kit is communicating with the CS8900A chip, it will be critical to test that the new "Chip Select" functionality is working, and see if the 8051 can access both the CS8900A chip and its own ROM and RAM in a single test program.

Should the Ethernet board pass these tests, then software development for the lab activity can continue, with the assurance that the code can be reliably tested on a cooperative piece of new hardware.

Should the Ethernet board fail these tests, then it will have to be determined exactly where things are going wrong, and with that information the board layout can be altered and a new revision of the board can be designed and ordered.

PERSONAL

Working on this project during IAP helped further develop my skills with schematic design and PCB layout. I also learned a lot of useful techniques for board assembly, especially concerning surface mount chips with many tiny pins (soldering these is much more difficult than I ever imagined).

Most importantly, though, I learned a lot about managing decisions for a project for which I was solely responsible. I had to decide exactly what chip configuration and part values I thought were best, based on competing sources of information. I had to pick out the parts that would be effective, cheap, and available. I had to make functionality and layout decisions tailored to the particular focus of the final product, which is to serve as an educational tool.

I am excited by how my board came out, and I look forward to testing it and modifying the design if necessary.

My thanks to the UROP program, Professor Leeb and the Laboratory for Electromagnetic and Electronic Systems for this experience.

Sabrina Neuman
Course VI-1, Class of 2009

Programmers Notepad - lowlev.asm

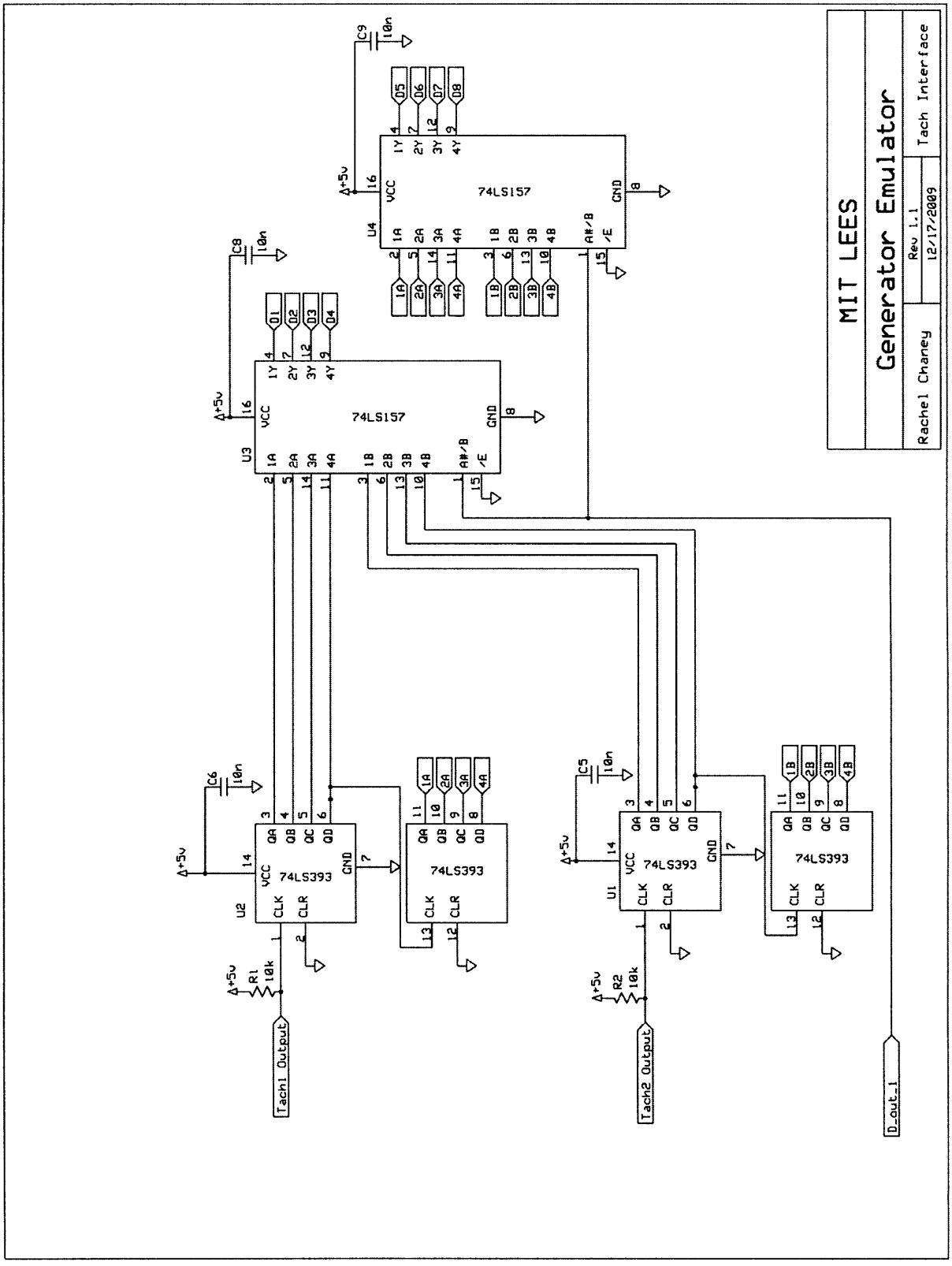
```
;*********************************************************************
; CS8900A Low Level Subroutines
;*********************************************************************

; The I/O Mode of the CS8900A gives 16 byte-long Ports
; These ports are really grouped into 8 two-byte Registers
; These registers are:
;   Port Address        Description
;   [high:low]
;   ---------------------------------------------------
;   [1:0]           Receive Transmit Data (Port 0)
;   [3:2]           Receive Transmit Data (Port 1)
;   [5:4]           TxCMD (Transmit Command)
;   [7:6]           TxLength (Transmit Length)
;   [9:8]           Interrupt Status Queue
;   [B:A]           PacketPage Pointer (PPPTR)
;   [D:C]           PacketPage Data (Port 0)
;   [F:E]           PacketPage Data (Port 1)

;=====================================================================

;=====================================================================
; low level subroutine addresswithPPPTR
; given:
;   intaddh     = high byte of 16-bit address within CS8900A
;   intaddl     = low byte of 16-bit address within CS8900A
; this subroutine will then send that address to the PPPTR
;   [B:A]           PacketPage Pointer (PPPTR)
;=====================================================================
addresswithPPPTR:
    mov dph, #0FEh
    mov dpl, #0Ah   ; low byte of PPPTR
    mov a, intaddl
    movx @dptr, a   ; write the low byte first...
    mov dph, #0FEh
    mov dpl, #0Bh   ; high byte of PPPTR
    mov a, intaddh
    movx @dptr, a   ; ... then write the high byte
    ret

;=====================================================================
; low level subroutine readinternal
; given:
;   intaddh     = high byte of 16-bit address within CS8900A
;   intaddl     = low byte of 16-bit address within CS8900A
; this subroutine will address the internal register with PPPTR,
; read data from PPD0,
;   [D:C]           PacketPage Data (Port 0)
; and return the data read there in:
;   rdinth      = the high byte read from within the CS8900A
;   rdintl      = the low byte read from within the CS8900A
;=====================================================================
readinternal:
    lcall addresswithPPPTR  ; move PPPTR to the address
    mov dph, #0FEh
    mov dpl, #0Dh   ; high byte of PPD0
    movx a, @dptr   ; read the high byte first...
    mov rdinth, a   ; and store it in rdinth
    mov dph, #0FEh
    mov dpl, #0Ch   ; low byte of PPD0
    movx a, @dptr   ; ... then read the low byte
    mov rdintl, a   ; and store it in rdintl
```

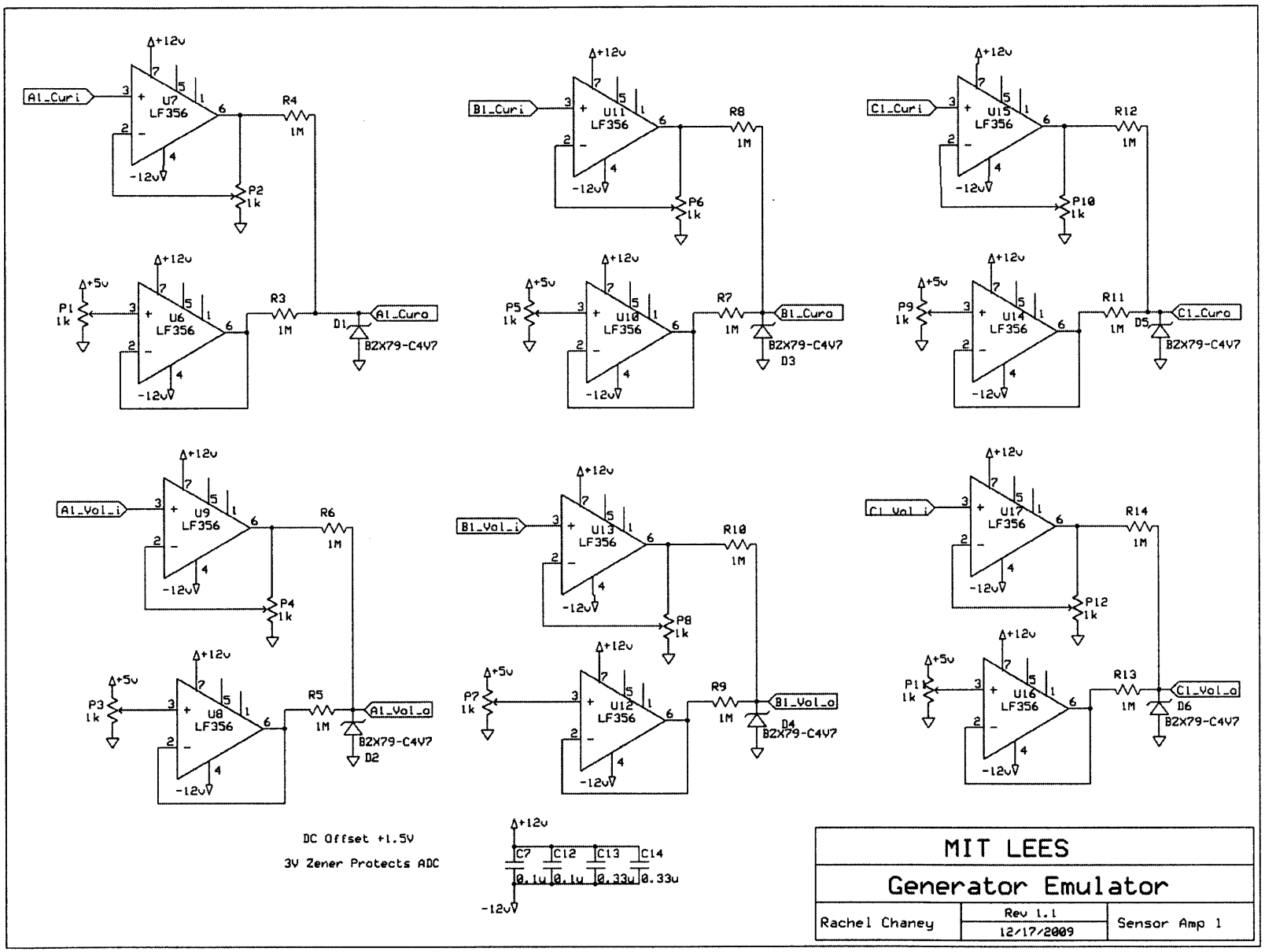
Page 1, 1/19/2010 - 11:27:27 PM

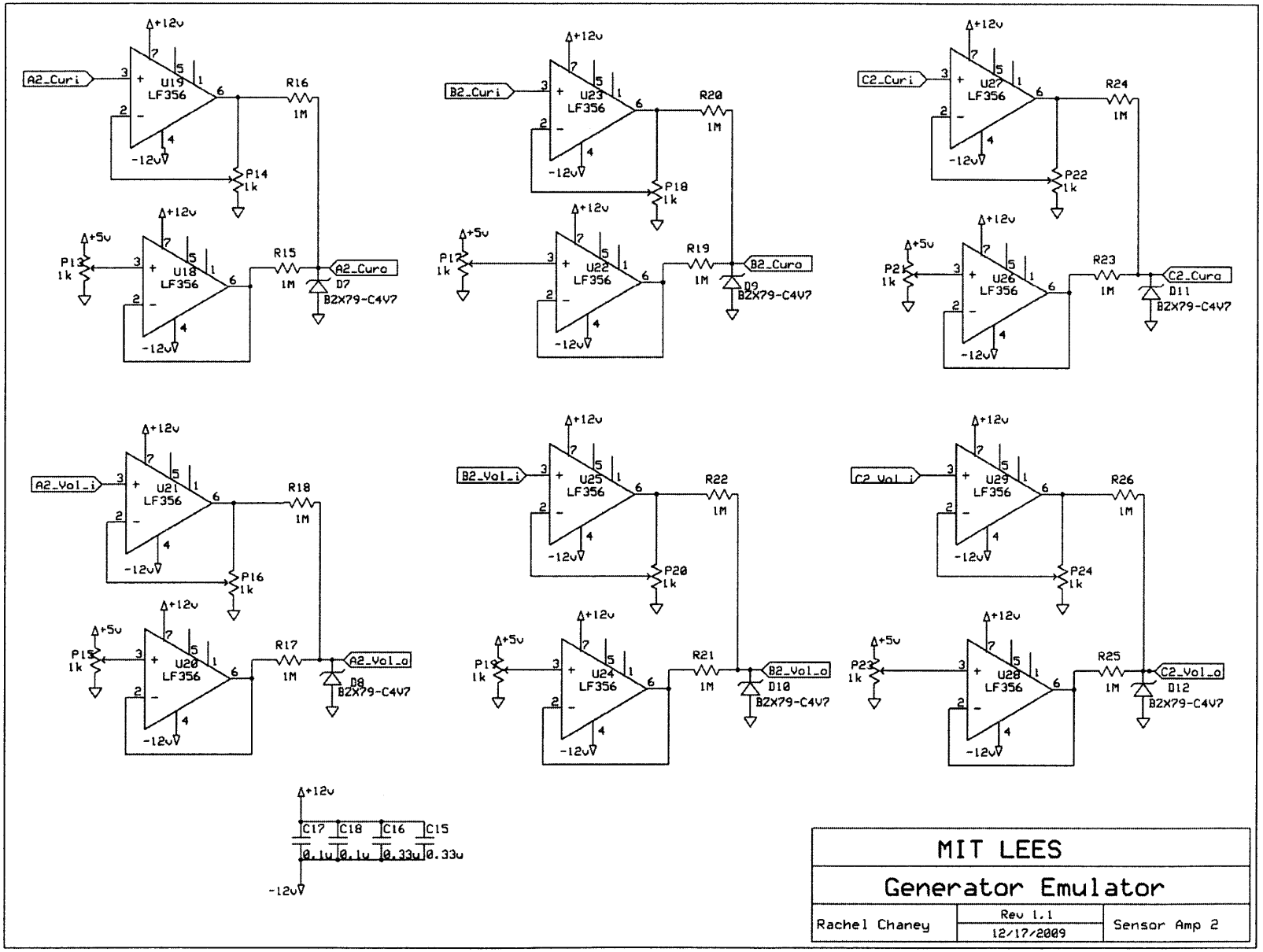
```
    ret

;=============================================================
; low level subroutine writeinternal
; given:
;   intaddh     = high byte of 16-bit address within CS8900A
;   intaddl     = low byte of 16-bit address within CS8900A
;   wrinth      = the high byte to write to within the CS8900A
;   wrintl      = the low byte to write to within the CS8900A
; this subroutine writes the data to the register in the chip,
; using the PPPTR to address,
; and the PPD0 to write data to.
;   [D:C]           PacketPage Data (Port 0)
;=============================================================
writeinternal:
    lcall addresswithPPPTR  ; move PPPTR to the address
    mov dph, #0FEh
    mov dpl, #0Ch   ; low byte of PPD0
    mov a, wrintl
    movx @dptr, a   ; write the low byte first...
    mov dph, #0FEh
    mov dpl, #0Dh   ; high byte of PPD0
    mov a, wrinth
    movx @dptr, a   ; ... then write the high byte
    ret

;=============================================================
; low level subroutine readport0
; this subroutine will address RTD0,
;   [1:0]           Receive Transmit Data (Port 0)
; and return the data read there in:
;   rdinth      = the high byte read from port 0
;   rdintl      = the low byte read from port 0
;=============================================================
readport0:
    mov dph, #0FEh
    mov dpl, #01h   ; high byte of RTD0
    movx a, @dptr   ; read the high byte first...
    mov rdinth, a   ; and store it in rdinth
    mov dph, #0FEh
    mov dpl, #00h   ; low byte of RTD0
    movx a, @dptr   ; ... then read the low byte
    mov rdintl, a   ; and store it in rdintl
    ret

;=============================================================
; low level subroutine writeport0
; given:
;   wrinth      = the high byte to write to port 0
;   wrintl      = the low byte to write to port 0
; this subroutine writes the data to port 0 (RTD0)
;   [1:0]           Receive Transmit Data (Port 0)
;=============================================================
writeport0:
    mov dph, #0FEh
    mov dpl, #00h   ; low byte of RTD0
    mov a, wrintl
    movx @dptr, a   ; write the low byte first...
    mov dph, #0FEh
    mov dpl, #01h   ; high byte of RTD0
    mov a, wrinth
    movx @dptr, a   ; ... then write the high byte
```

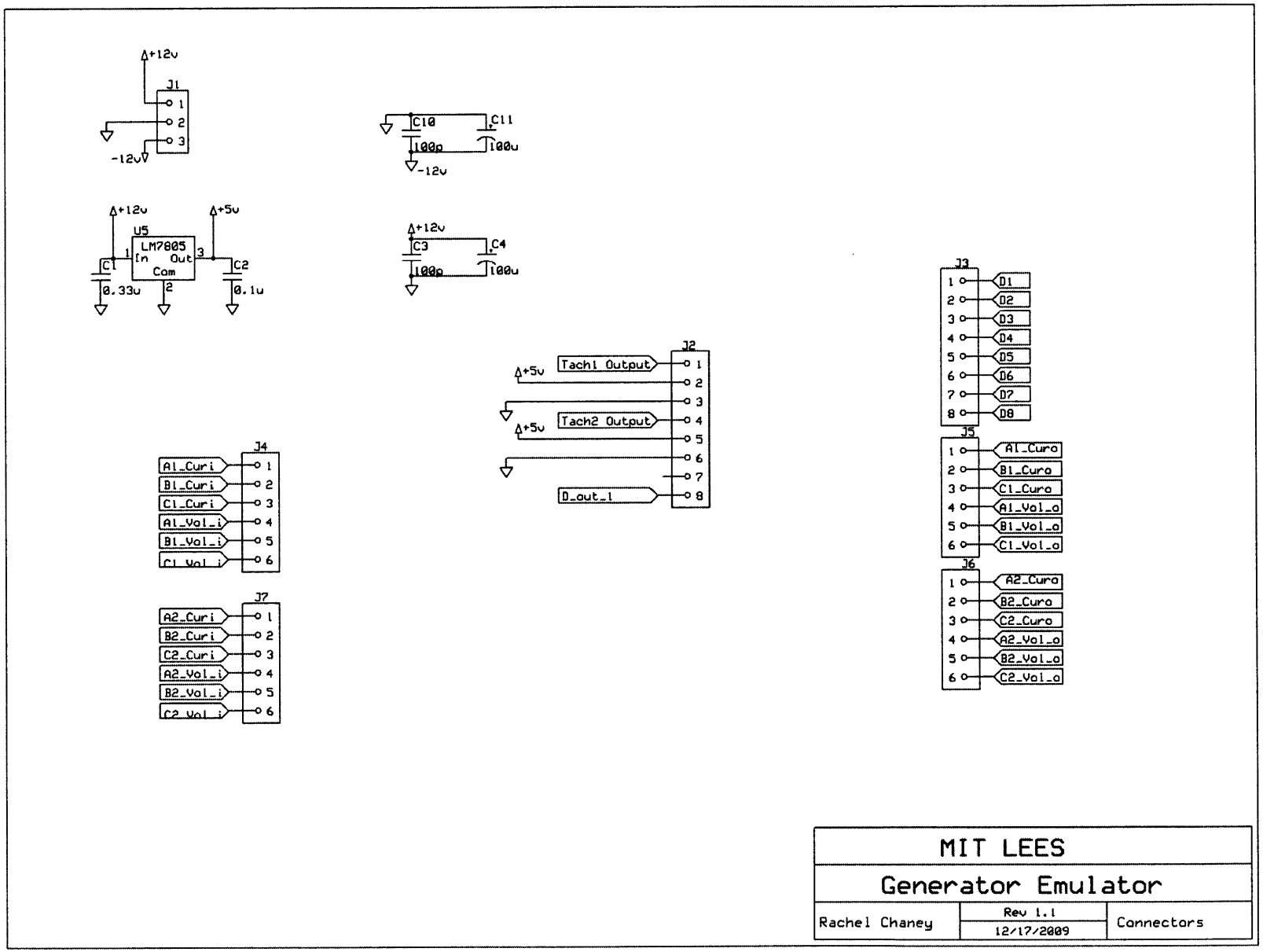
```
    ret

;=============================================================================
; low level subroutine writeTxCMD
; given:
;   wrinth  = the high byte to write
;   wrintl  = the low byte to write
; this subroutine writes the data to TxCMD
;   [5:4]           TxCMD (Transmit Command)
;=============================================================================
writeTxCMD:
    mov dph, #0FEh
    mov dpl, #04h   ; low byte of TxCMD
    mov a, wrintl
    movx @dptr, a   ; write the low byte first...
    mov dph, #0FEh
    mov dpl, #05h   ; high byte of TxCMD
    mov a, wrinth
    movx @dptr, a   ; ... then write the high byte
    ret

;=============================================================================
; low level subroutine writeTxLEN
; given:
;   wrinth  = the high byte to write
;   wrintl  = the low byte to write
; this subroutine writes the data to TxLEN
;   [7:6]           TxLength (Transmit Length)
;=============================================================================
writeTxLEN:
    mov dph, #0FEh
    mov dpl, #06h   ; low byte of TxLEN
    mov a, wrintl
    movx @dptr, a   ; write the low byte first...
    mov dph, #0FEh
    mov dpl, #07h   ; high byte of TxLEN
    mov a, wrinth
    movx @dptr, a   ; ... then write the high byte
    ret
```

Appendix D: Generator Emulator Sensor Board Schematics

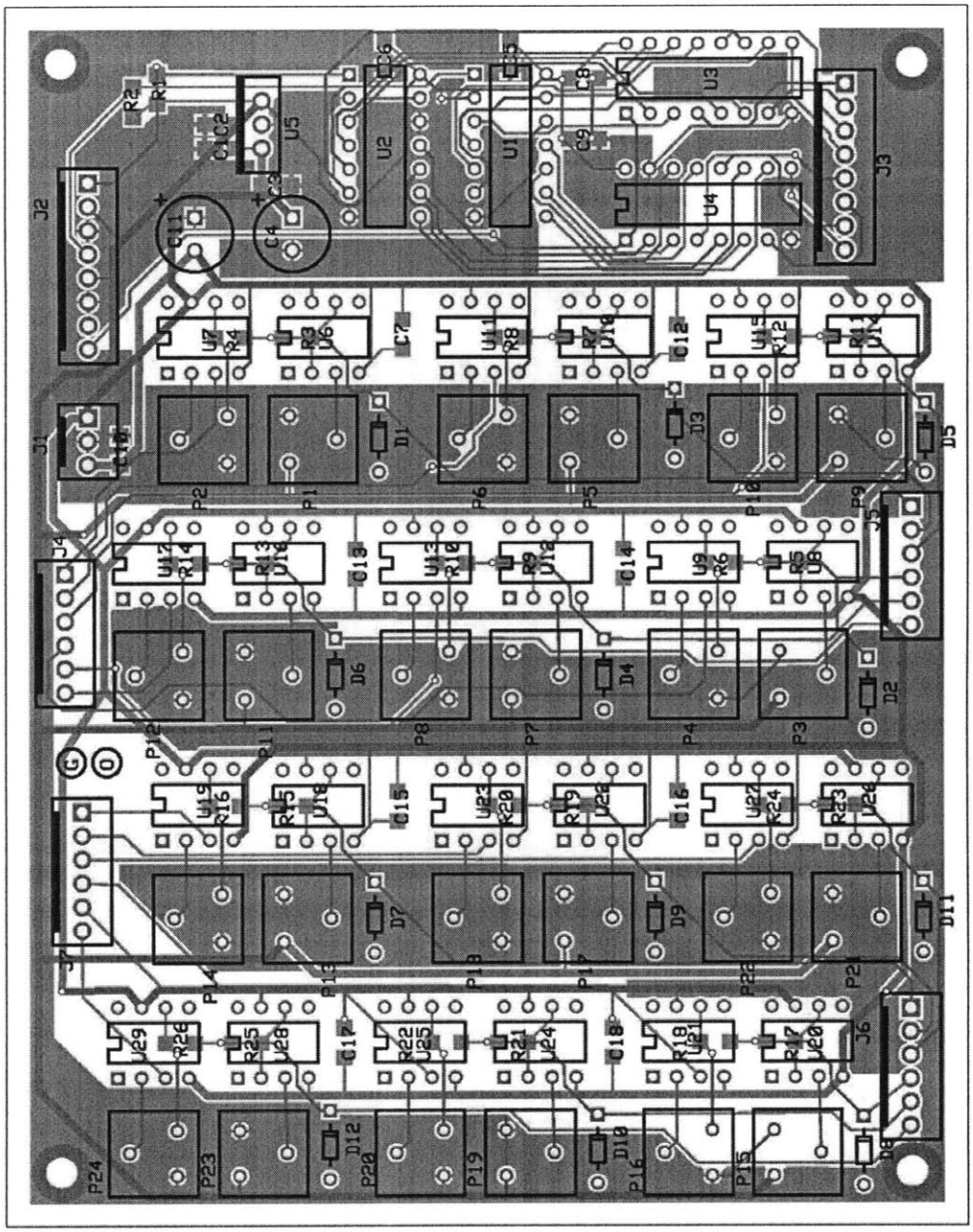



A1_Curi
U7
LF356
R4
1M
P2
1k
+12v
-12v
P1
1k
+5v
U6
LF356
R3
1M
D1
A1_Curo
BZX79-C4V7
B1_Curi
U11
LF356
R8
1M
P6
1k
P5
1k
U10
LF356
R7
1M
B1_Curo
BZX79-C4V7
D3
C1_Curi
U15
LF356
R12
1M
P10
1k
P9
1k
U14
LF356
R11
1M
D5
C1_Curo
BZX79-C4V7
A1_Vol_i
U9
LF356
R6
1M
P4
1k
P3
1k
U8
LF356
R5
1M
A1_Vol_o
BZX79-C4V7
D2
B1_Vol_i
U13
LF356
R10
1M
P8
1k
P7
1k
U12
LF356
R9
1M
B1_Vol_o
BZX79-C4V7
D4
C1_Vol_i
U17
LF356
R14
1M
P12
1k
P11
1k
U16
LF356
R13
1M
C1_Vol_o
D6
BZX79-C4V7
DC Offset +1.5V
3V Zener Protects ADC
C7
C12
C13
C14
0.1u
0.1u
0.33u
0.33u
MIT LEES
Generator Emulator
Rachel Chaney
Rev 1.1
12/17/2009
Sensor Amp 1


A2_Curi
B2_Curi
C2_Curi
A2_Vol_i
B2_Vol_i
C2_Vol_i
U19 LF356
U23 LF356
U27 LF356
U21 LF356
U25 LF356
U29 LF356
U18 LF356
U22 LF356
U26 LF356
U20 LF356
U24 LF356
U28 LF356
R16 1M
R20 1M
R24 1M
R18 1M
R22 1M
R26 1M
R15 1M
R19 1M
R23 1M
R17 1M
R21 1M
R25 1M
P14 1k
P18 1k
P22 1k
P16 1k
P20 1k
P24 1k
P13 1k
P17 1k
P21 1k
P15 1k
P19 1k
P23 1k
A2_Curo
B2_Curo
C2_Curo
A2_Vol_o
B2_Vol_o
C2_Vol_o
D7
D9
D11
D8
D10
D12
BZX79-C4V7
+12v
-12v
+5v
C17 0.1u
C18 0.1u
C16 0.33u
C15 0.33u
MIT LEES
Generator Emulator
Rachel Chaney
Rev 1.1
12/17/2009
Sensor Amp 2



MIT LEES		
Generator Emulator		
Rachel Chaney	Rev 1.1 12/17/2009	Connectors



Part No	Value	Tootprint	Company	Product No.
C1	0.33u	1206	Digikey	445-4010-1-ND
C2	0.1u	1206	Digikey	493-2336-1-ND
C3	100p	1206	Digikey	311-1161-1-ND
C4	100u	3.5mm	Digikey	P13459-ND
C5	10n	1206	Digikey	311-1174-1-ND
C6	10n	1206	Digikey	311-1174-1-ND

C7	0.1u	1206	Digikey	493-2336-1-ND
C8	10n	1206	Digikey	311-1174-1-ND
C9	10n	1206	Digikey	311-1174-1-ND
C10	100p	1206	Digikey	311-1161-1-ND
C11	100u	3.5mm	Digikey	P13459-ND
C12	0.1u	1206	Digikey	493-2336-1-ND
C13	0.33u	1206	Digikey	445-4010-1-ND
C14	0.33u	1206	Digikey	445-4010-1-ND
C15	0.33u	1206	Digikey	445-4010-1-ND
C16	0.33u	1206	Digikey	445-4010-1-ND
C17	0.1u	1206	Digikey	493-2336-1-ND
C18	0.1u	1206	Digikey	493-2336-1-ND
D1	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D2	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D3	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D4	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D5	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D6	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D7	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D8	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D9	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D10	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D11	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
D12	BZX79-C4V7	DO-35	Digikey	568-1670-1-ND
J1	Molex 3 pin	WM4201	Digikey	WM4201-ND
J2	Molex 8 pin	WM4206	Digikey	WM4206-ND
J3	Molex 8 pin	WM4206	Digikey	WM4206-ND
J4	Molex 6 pin	WM4204	Digikey	WM4204-ND
J5	Molex 6 pin	WM4204	Digikey	WM4204-ND
J6	Molex 6 pin	WM4204	Digikey	WM4204-ND
J7	Molex 6 pin	WM4204	Digikey	WM4204-ND
P1	1k	3386F	Digikey	3386F-102LF-ND
P2	1k	3386F	Digikey	3386F-102LF-ND
P3	1k	3386F	Digikey	3386F-102LF-ND
P4	1k	3386F	Digikey	3386F-102LF-ND
P5	1k	3386F	Digikey	3386F-102LF-ND
P6	1k	3386F	Digikey	3386F-102LF-ND
P7	1k	3386F	Digikey	3386F-102LF-ND
P8	1k	3386F	Digikey	3386F-102LF-ND
P9	1k	3386F	Digikey	3386F-102LF-ND
P10	1k	3386F	Digikey	3386F-102LF-ND

P11	1k	3386F	Digikey	3386F-102LF-ND
P12	1k	3386F	Digikey	3386F-102LF-ND
P13	1k	3386F	Digikey	3386F-102LF-ND
P14	1k	3386F	Digikey	3386F-102LF-ND
P15	1k	3386F	Digikey	3386F-102LF-ND
P16	1k	3386F	Digikey	3386F-102LF-ND
P17	1k	3386F	Digikey	3386F-102LF-ND
P18	1k	3386F	Digikey	3386F-102LF-ND
P19	1k	3386F	Digikey	3386F-102LF-ND
P20	1k	3386F	Digikey	3386F-102LF-ND
P21	1k	3386F	Digikey	3386F-102LF-ND
P22	1k	3386F	Digikey	3386F-102LF-ND
P23	1k	3386F	Digikey	3386F-102LF-ND
P24	1k	3386F	Digikey	3386F-102LF-ND
R1	10k	1206	Digikey	RMCF1/810KJRCT-ND
R2	10k	1206	Digikey	RMCF1/810KJRCT-ND
R3	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R4	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R5	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R6	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R7	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R8	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R9	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R10	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R11	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R12	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R13	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R14	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R15	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R16	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R17	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R18	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R19	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R20	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R21	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R22	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R23	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R24	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R25	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
R26	1M	1206	Digikey	CRT1206-BY-1004ELFCT-ND
U1	74LS393	DIP-14	Digikey	296-1665-5-ND

U2	74LS393	DIP-14	Digikey	296-1665-5-ND
U3	74LS157	DIP-16	Digikey	296-1645-5-ND
U4	74LS157	DIP-16	Digikey	296-1645-5-ND
U5	LM7805	T0-220	Digikey	LM7805ACT-ND
U6	LF356	DIP-8	Digikey	LF356N-ND
U7	LF356	DIP-8	Digikey	LF356N-ND
U8	LF356	DIP-8	Digikey	LF356N-ND
U9	LF356	DIP-8	Digikey	LF356N-ND
U10	LF356	DIP-8	Digikey	LF356N-ND
U11	LF356	DIP-8	Digikey	LF356N-ND
U12	LF356	DIP-8	Digikey	LF356N-ND
U13	LF356	DIP-8	Digikey	LF356N-ND
U14	LF356	DIP-8	Digikey	LF356N-ND
U15	LF356	DIP-8	Digikey	LF356N-ND
U16	LF356	DIP-8	Digikey	LF356N-ND
U17	LF356	DIP-8	Digikey	LF356N-ND
U18	LF356	DIP-8	Digikey	LF356N-ND
U19	LF356	DIP-8	Digikey	LF356N-ND
U20	LF356	DIP-8	Digikey	LF356N-ND
U21	LF356	DIP-8	Digikey	LF356N-ND
U22	LF356	DIP-8	Digikey	LF356N-ND
U23	LF356	DIP-8	Digikey	LF356N-ND
U24	LF356	DIP-8	Digikey	LF356N-ND
U25	LF356	DIP-8	Digikey	LF356N-ND
U26	LF356	DIP-8	Digikey	LF356N-ND
U27	LF356	DIP-8	Digikey	LF356N-ND
U28	LF356	DIP-8	Digikey	LF356N-ND
U29	LF356	DIP-8	Digikey	LF356N-ND