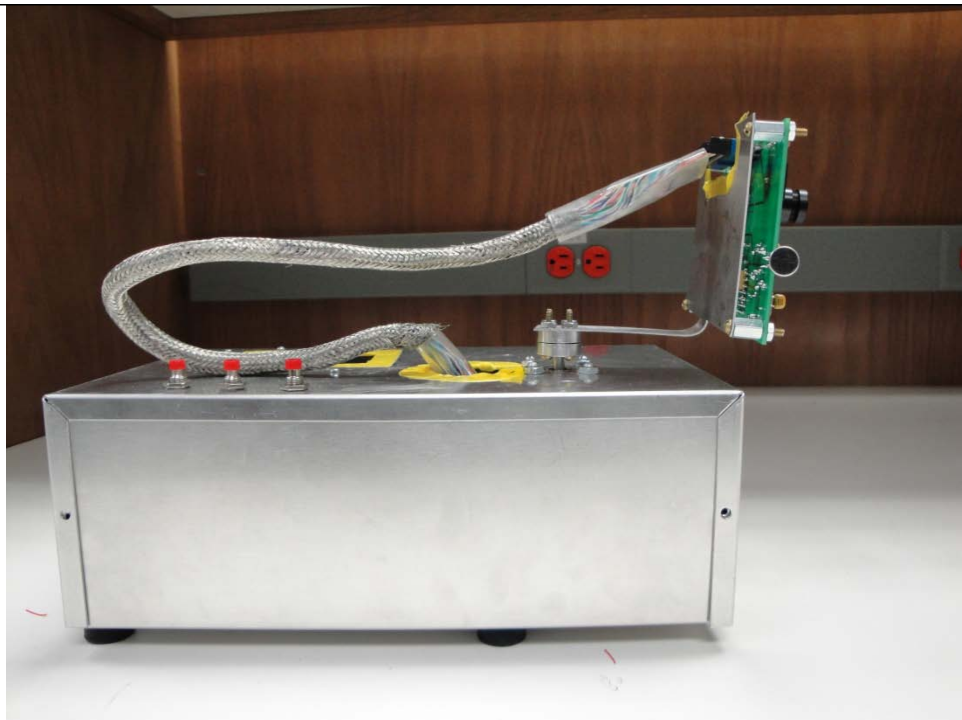
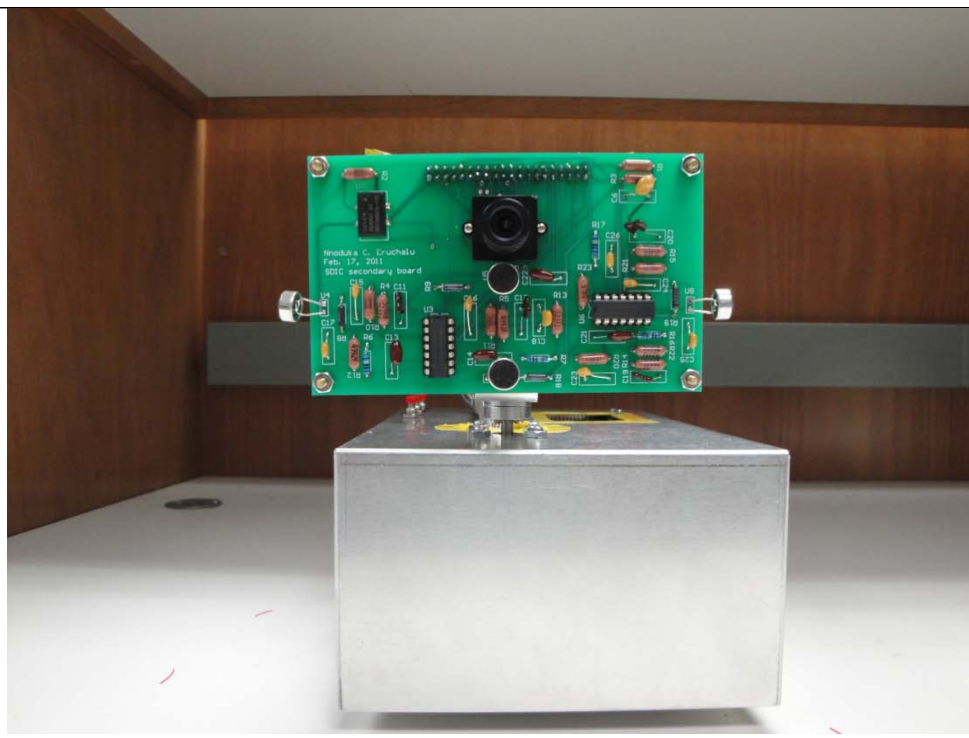
# Sound Detect and Image Capture Project
# EE91b, Winter 2011






**Nnoduka C. Eruchalu**

**MSC 271, Caltech**

**(626)-660-4830**

**nnodukae@caltech.edu**

# Sound Detect and Image Capture Project

**Nnoduka C. Eruchalu**
**MSC 271, Caltech**
**(626)-660-4830**
**nnodukae@caltech.edu**

# Table of Contents

## General Safety Summary

To avoid fire, personal injury, or even worse total destruction of the SDIC

**Use Proper Power Cord.** Use only the power cord specified for this project and certified for use in the U.S.A. This power supply should be stable to avoid erratic behaviors of the SDIC.

**Connect and Disconnect Properly.** Do not just pull out the power cord.
This could break solder joints, destroy the connectors or break the board.

**Do Not Operate With Suspected Failures.** If you suspect there is damage to this product, have it inspected by the maker of this project, Nnoduka C. Eruchalu.

**Do Not Operate at Extreme Temperatures.** Room temperature is best. It is easiest to appreciate the art of sound detection and image capturing at decent temperatures.

**Handle SDIC with Caution.** While it is a sturdy product careful treatment will increase its lifetime. So do not move it around in quick and jerky motions, and
store it in a safe location.

**Keep Out of the Reach of Children.** But do not deprive them of the joy of seeing a quality product in action.

**Do Not Operate in Wet/Damp Conditions.**

**Keep SDIC Clean and Dry.**

# Introduction and Project Overview

## Motivation

The idea behind this project was to put together my skills at systems design in a hybrid analog and digital project. After about 3.5 years at Caltech, this shows how well I can propose, design, and execute a project which results in a useful product.  This gives me an opportunity to once again go through an entire product development cycle- design, construction, debugging, and packaging in a professional manner.

## System Description and Specification

### Project Description

The project for this term was the design and construction of a Sound Detector and Image Capture system. The project was implemented as an analog/digital hybrid system.

This project is to simulate a security system product.

There are 4 electret microphones spinning on a stepper motor. The microphones have accompanying amplification circuits designed to give a gain of 1000. This gain of 1000 means that it works best at detecting “long” distance sounds which is reasonable for a security feature which is usually mounted out of sight of people.

Also on this stepper motor, an image sensor + lens setup is mounted. When the user presses **GO**, the stepper motor does a full 360 degree rotation. During the rotation it records the sound at each position and at the end of the rotation it returns to the location where the maximal sound was. With this the user can see the source of this maximal sound. With the use of the **MODE** button, the user can get either a video or a picture from the image sensor. This picture or video is displayed on a 2.4” Graphics LCD (320x240 resolution).

The stepper motor moves 9 degrees before every sound level measurement, so the accuracy of the final product is a multiple of this 9 degrees.
As it turns out the worst case scenario at decent ranges is +/- 18 degrees.

The design requirements were to get decent imagery of the sound sources with error in sound detection being minimized as much as possible ( < 10% error). I hit this requirement, because an error margin of +/- 18 degrees, provides for an error margin of +/-5%. At some sound levels within some range I am able to get accurate sound detection. In any event the imagery is always crisp and clear.

Of importance also is the final project packaging and its ranging abilities in its finished form. The project features, packaging and dimensions are discussed in detail:

**Project Features**

The Sound Detect and Image Capture (SDIC) device has the following noteworthy features:

1. Portable -The size of the SDIC and the compact wall-mount power supply makes it convenient to carry around and place anywhere (for security purposes)
2. Mountable on any surface - SDIC has rubber feet to keep it fairly well mounted on most surfaces, even rugs.
3. Sturdy - The sheet aluminum metal casing makes SDIC sturdy. The rubber feet/stands also

Help protect SDIC from many impulsive forces.

4. 120VAC wall power - The user can conveniently connect at any wall power outlet.

5. Imaging Modes - The user can conveniently get still images in the picture mode or get continuous image streaming in video mode.

6. Accuracy – SDIC is accurate to +/-18 degrees. This margin of +/-5% makes it a decent security device. This value of +/-18 degrees comes out of the fact that the stepper motor is setup to make 9 degree turns before measuring sound levels. Going any smaller would have given more accuracy at the expense of speed, because it will then take more time to do a single round of sound detection and image capture. An error bar of +/-5% as is the case now seemed to be the worst acceptable error, so I stuck with it to get decent speed.

**Project Packaging**

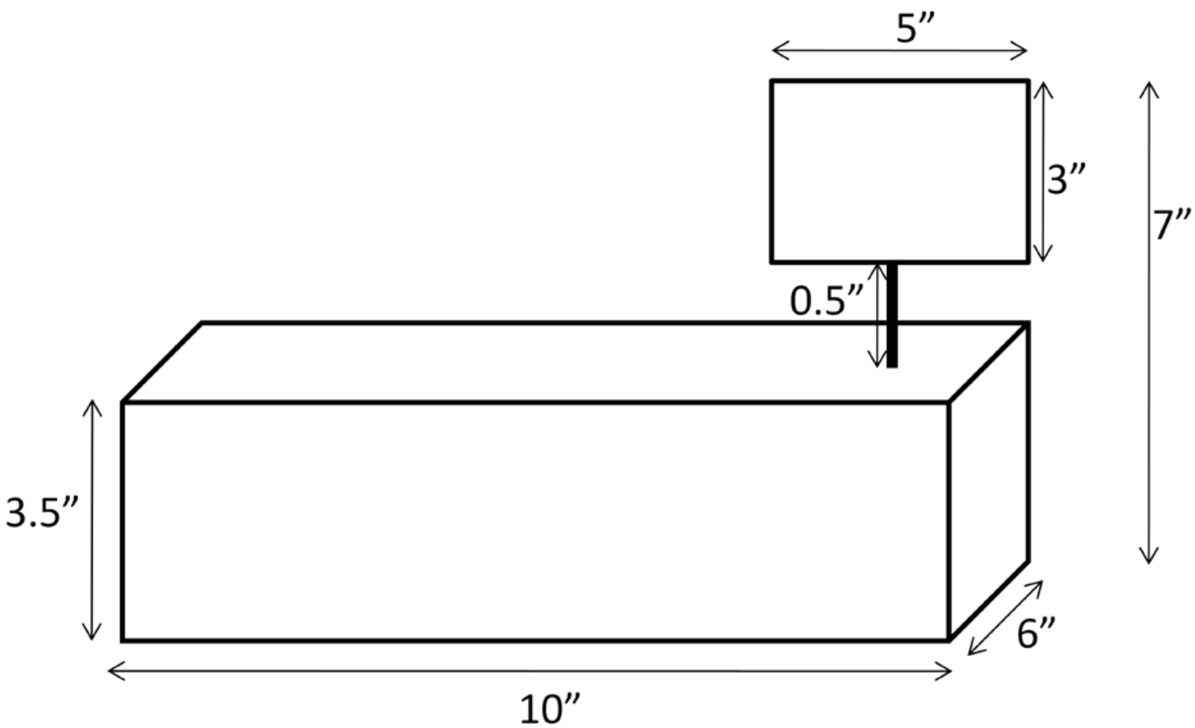
The circuitry is all done on two printed circuit boards (pcbs) – one is the main system control board, and the other is the secondary board rotating on the stepper motor. The external peripherals that the system has to communicate with is:

- A wall power pack.
- Switches
- An image sensor + lens + mount
- A Graphic LCD
- A stepper motor

Using the materials available to me in the Caltech EE stockroom I decided on using an aluminum sheet metal box. The 2.1mm power jack hole is on one side of the box. The control switches, graphic LCD, and stepper motor are all on the top side of the box. The switches and the stepper motor protrude through the class, while the LCD is safely mounted inside the box and is seen through a cutout from the top side.

**Project Dimensions**

Below is a block image of the dimensions of the final product.




This does not take into account the which protrude by 0.4”  on the front panel, and the lens (plus mount) which protrudes by 1.5” from the rotating board attached to the shaft.

# Characterization of Electrical Performance

Below is a table characterizing the electrical performance of the various important components of the SDIC

| Parameter | Minimum | Typical | Maximum | Unit |
|---|---|---|---|---|
| Input Voltage | 7.5 | 9 | 30 | V |
| Input Current | 1600 | 1900 | 2500 | mA |
| Operating Temperature | -20 | 20 | 60 | °C |
| Detected sound frequency range | 100 | | 12000 | Hz |
| Sensitivity to sound levels | 20 | 23 | 26 | dB |
| Microphones module SNR | - | 69 | - | dB |
| Camera module Responsivity | - | 1.9 (at 550nm) | - | V/lux-sec |
| Camera module Dynamic Range | - | 60 | - | dB |
| Camera module SNR | - | - | 45 | dB |
| Camera module Frame Rate | - | 5 | - | fps |

# Characterization of Product Performance

Below is a table characterizing the product performance of the SDIC.

| Parameter | Minimum | Typical | Maximum | Unit |
|---|---|---|---|---|
| Light intensity for image capture | 6 | - | 15 | lux |
| Minimum detectable ranges with accuracy | 0.01 | - | 30 | m |
| Motor Accuracy | 9 | 9 | 9 | degree(°) |
| Error | -5 | 0 | +5 | % |

Note some formulas for working with sound:

1 Pa = 1 $N/m^2$ = 10 $dyne/cm^2$ = 10ubar = 94 dB SPL

1 ubar = 74 dB SPL

dB SPL = 20* log(P/P0) where P is the measured pressure and P0 is a reference pressure in the same system of units:

P0 = 20uPa (or $uN/m^2$)

The microphone sensitivity is -37dB at 1V/Pa. This means that if I get an output that is 37dB below 1V (i.e. 14mV) then the sound pressure is 1Pa = 94 dB SPL

The microphone amplifier circuit has a gain of 1000 = 60dB

This means that the system sensitivity is 23dB at 1V/Pa. This means that if I get an output from the microphone amplifier stage that is 23dB above 1V (i.e. 14V) then the sound pressure is 1Pa = 94 dB SPL.

This means if u feed it 74 db SPL, expect 1.4V.

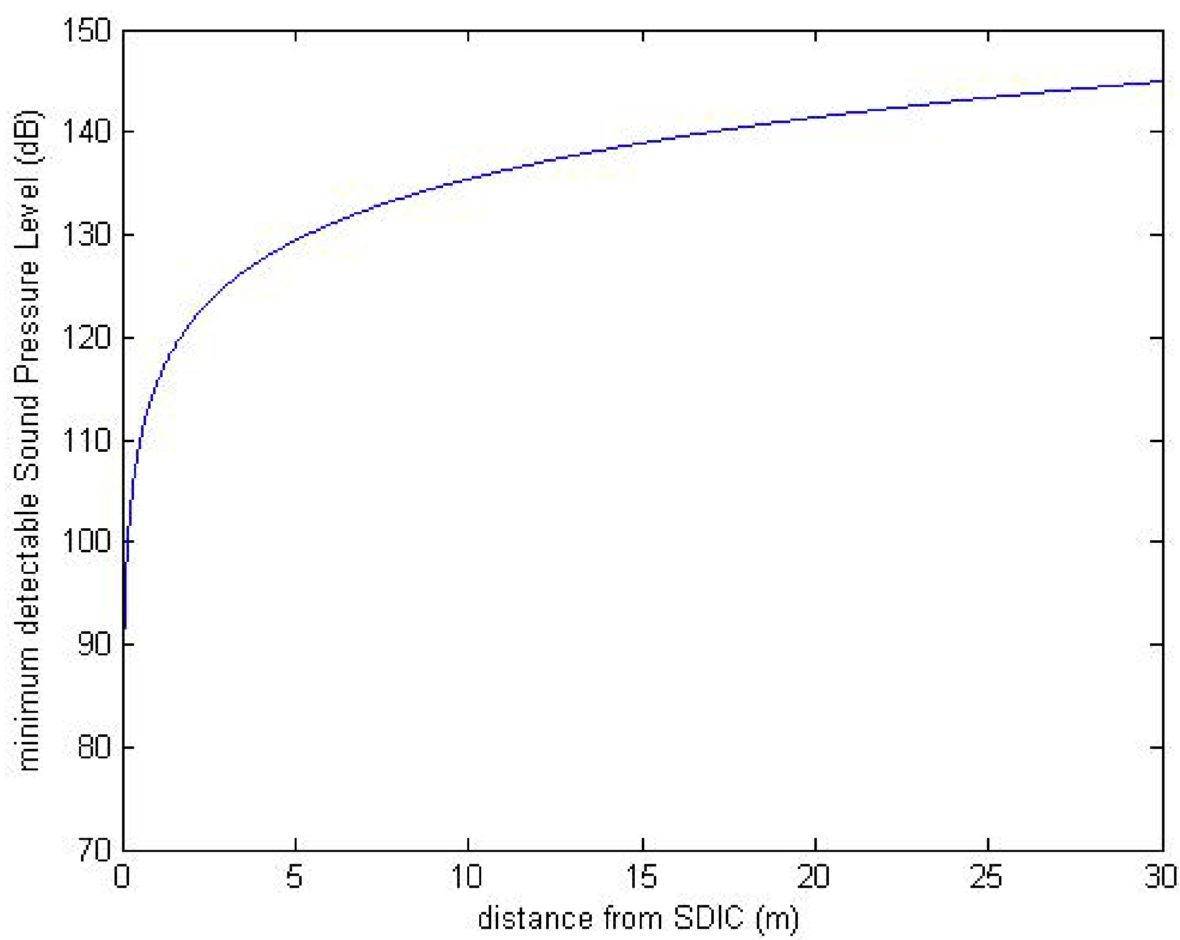
Also this means that if you expect the minimum detectable voltage of 1.65V, you will need 94-20*log(14/1.65 = 75.4 dB SPL right at surface of the microphone. Assuming this reference is 0.01m away

So the minimum SPL required at the surface of the microphone (x=0.01) is always 75.4dB.

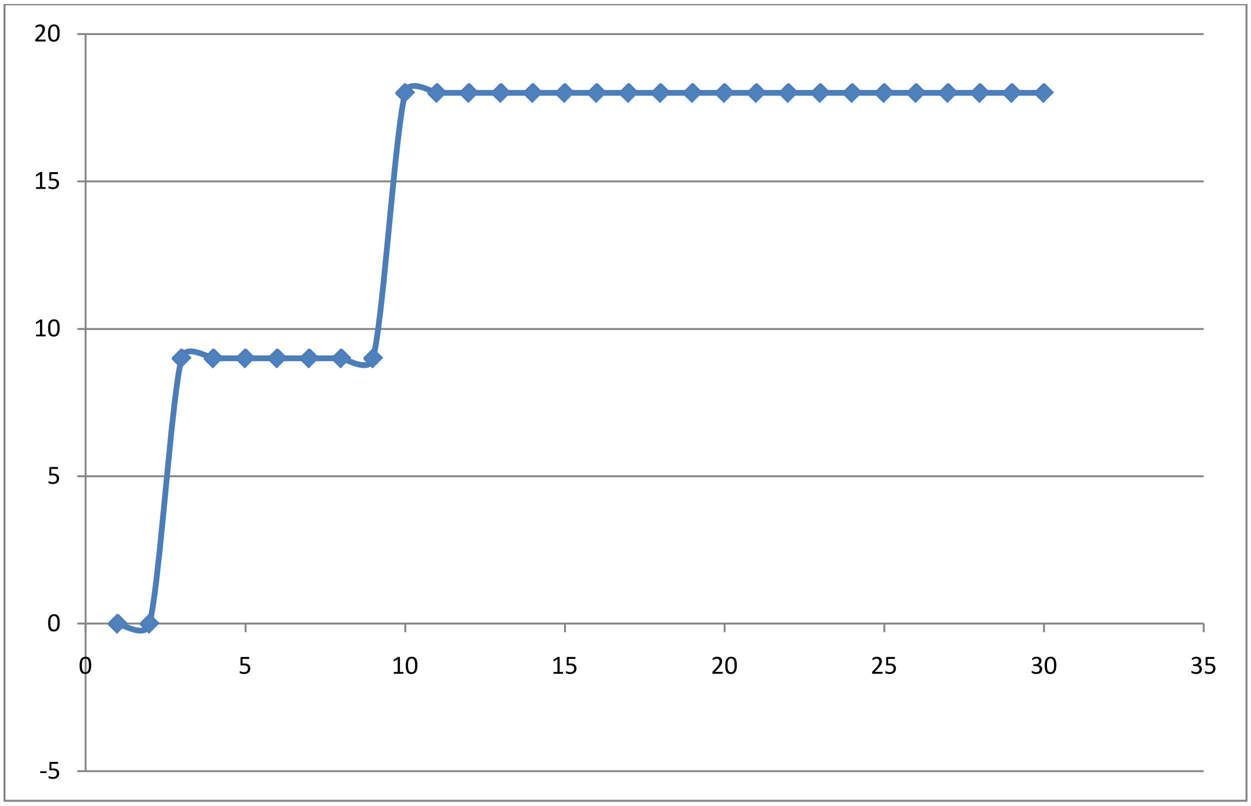
Then at any other distance x > 0.01, the minimum required SPLreq = 75.4 + 20*log10(x/0.1)

[Note log10 = log base 10]

**Below is a plot which shows the minimum detectable Sound Pressure Level from a source at any distance from SDIC:**



**Below is a plot of absolute number of degrees off in SDIC detection versus source distance from SDIC [accuracy (°) vs. distance (m)]**



## Challenges faced

**The biggest circuit challenge faced was getting the microcontroller/CPLD interface between the camera module and the graphic LCD to work**.

The camera module (image sensor + lens-mount + lens) starts up in default settings that did not work for the system. So I first had to hook up the digital circuitry to interface the PIC microcontroller to the camera module. The PIC communicates with the image sensor over an i2c interface and puts the camera module into a state that can be used by the rest of the system. When doing the i2c interface I had to be sure to use the appropriate pull-up resistors as shown on the schematics. Note that in the schematics, the pull-up resistors are pulled up to the 2.8V voltage (which is the power voltage value for the image sensor). This is necessary because the digital signals from the PIC are at 3.3V. This difference in maximum voltage levels between the PIC and the image sensor also calls for a buffer between them as there is another signal going between them, the Camera Reset.

With the image sensor setup, there is the issue of converting the bayer raw data from the sensor to the RGB format required by the graphic LCD. The initial attempts to do conversions from bayer to RGB required saving a lot of pixels from the image sensor in an SRAM. An additional SRAM just for image buffering seemed like a lot of overkill. This is because such solutions required the additional use of large FIFOs. The complications in all this required the use of logic devices with high I/O pin counts. The PIC microcontroller used in the project was already stretched thing on I/O pin usage. This required the addition of a new logic device, and so I decided to go with a Lattice 1016E CPLD. But even the I/O pin count of 32 on the CPLD would not be enough for an SRAM and two FIFOs. One solution was to get a bigger CPLD like a lattice 2032. This was impractical as I had to use what was available in the EE stockroom.

The 1016E CPLD also has a limited amount of logic it can take so I needed to come up with a simpler way of converting the Bayer raw data to 16-bit (5-6-5) RGB. After trying many variations of the initial solution I came up with a memory-less solution. I had figured out how to eliminate the complications of an SRAM and FIFOs.

The solution was that, the Bayer raw data assumes only one color per pixel, and gives 10 bit data for each color in each pixel. The CPLD then takes each pixel, and places the high 5 or 6 data-bits of the camera output into the appropriate bit-space for the color, while clearing out the other 11 or 10 bits for the 2 unused colors. Since bayer raw data has a concentration of 2-Green, 1-Blue, 1-Red pixel for every 4 pixels, the final 16-bit RGB picture has a green-tinge to it.

This is a crude solution, but for a security product like SDIC it works just fine.

Also it should be noted that for this solution to work, the image sensor's pixel output has to be slowed down by a factor documented in the CPLD Abel code. This factor is how many clocks it takes for the CPLD to grab the image, do my crude Bayer-to-RGB transformation, and write it to the LCD using an 8-bit interface. Of course this assumes that the image sensor and the CPLD run at the same speeds, which they do. This product uses 25MHz clocks only.

Now that we have the CPLD outputting to the LCD, we need to be able to decide on who gets to write to the LCD between the CPLD and the PIC. The PIC writes to the LCD during system initialization. The PIC has on-system memory so the initialization commands and values are saved in program memory. So when the PIC is done initializing the LCD, it flips the Mux select line. This mux select line goes to the multiplexers which then output either the CPLD or the PIC data and control lines to the LCD.

This leaves one final problem. The multiplexers run at 5V (all that was available in the EE stockroom), so their outputs are sent through 3.3V buffers which is the voltage the LCD runs at.

This is how I solved my problem of a digital control interface between the camera module and the graphic LCD.

## Lessons Learned

The biggest thing I learned here was that in coming up with a proposal for a product/project, I need to think hard about its usefulness and the desired customer base. I should then use this information to design and complete the project within the proposed time frame. I learned of the importance of creating weekly milestones and diligently following them. This project also gave me a chance to showcase my skills in putting together a hybrid analog-digital project which could have applications beyond the classroom/lab.

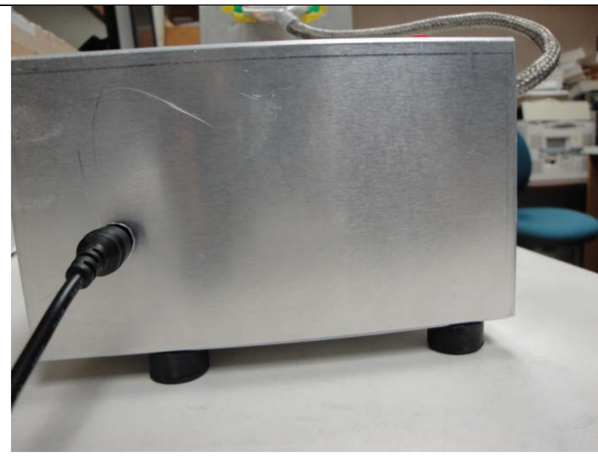
Another key lesson learned were how image sensors used in high end devices like cell phones and webcams work. They output bayer raw data and have to be converted to appropriate formats using complicated demosaicing algorithms. Also with the electrets microphones, I was forced into understanding the math behind sound pressure levels and their relationship with distance, and how all these relate to microphone sensitivity.

Of course I was also forced to improve my skills with a drill press and other tools in the EE machining shop so as to be able to create my packaging. It was another tough and painful experience but now with a finished and well packaged product it feels well worth it.

# User Manual

## Connecting Power

The system power is a wall supply unit which outputs at least 7.5V. The wall supply unit must have a 2.1mm jack. Below is a sample of the best option for the wall power supply unit, and where to connect it.



## Controlling and Operating the SDIC

Now that we know how to connect the appropriate power supply, the next step is discussing the inputs and outputs:

**Inputs:**
Below is a table with the input devices and their descriptions

| Input Device | Description |
|---|---|
| GO switch | This push button switch starts up the sound detection routine. It makes the microphones, do a full 360 degree scan of its surrounding and then point the camera at the sound source with maximum sound pressure level. This switch is located on the top |

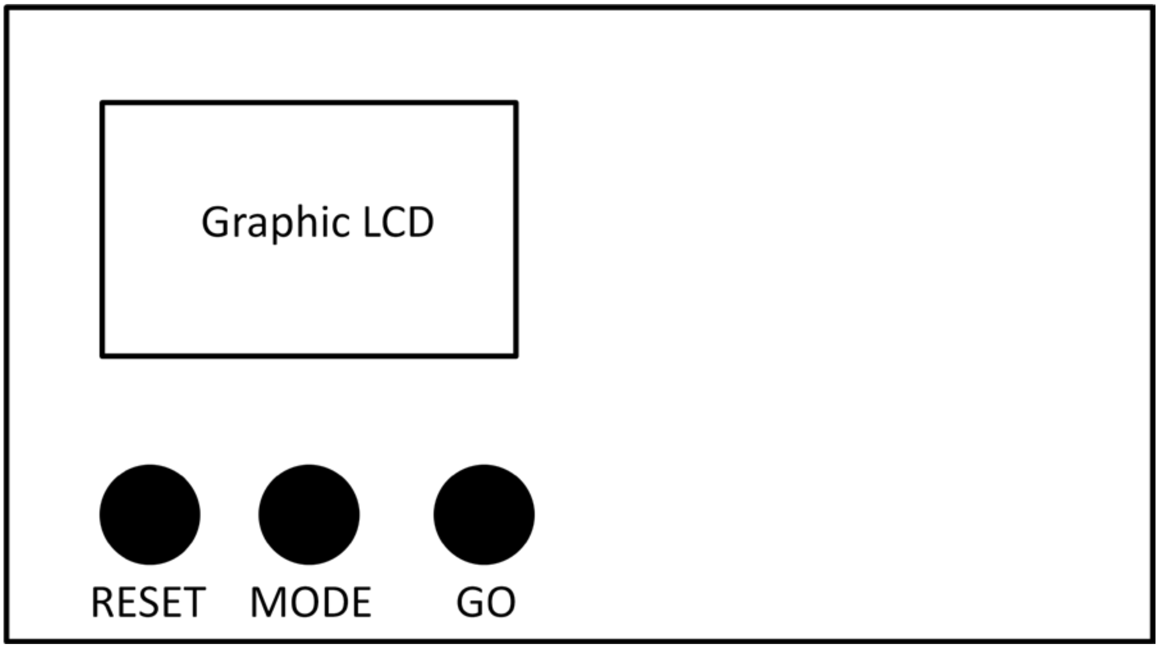
| | |
|---|---|
| | panel of the product. |
| MODE switch | This push button switch toggles the display mode on the graphic LCD between picture and video mode. This switch is located on the top panel of the product. |
| RESET switch | This push button switch resets the system. Really it resets the microcontroller and the CPLD in the system which is effectively resetting the system.<br><br>This also could be used to set the 0-orientation position of the motor. Because of where the connecting cable comes from, the zero orientation position should be as shown in figure 1.<br>To reset it, all that has to be done is manually spin the secondary PCB to this position, then hold down the reset till you see the Graphic LCD resetting. This should take about 3 to 5 seconds.<br><br>This switch is located on the top panel of the product. |

**Outputs:**
Below is a table with the output devices and their descriptions

| Output Device | Description |
|---|---|
| Graphic LCD | This is a 2.4” color graphic LCD with resolution 320x240. It displays what the camera currently points at when the system is in video mode. When the system is in picture mode it displays the last taken image. The Display is located on the top panel of the product. |

**User Interface**

Below is a depiction of the top of the box showing the user interface. This ignores the rotating board and the motor shaft it is placed on.




## Example Images

Below are some images of the final product.

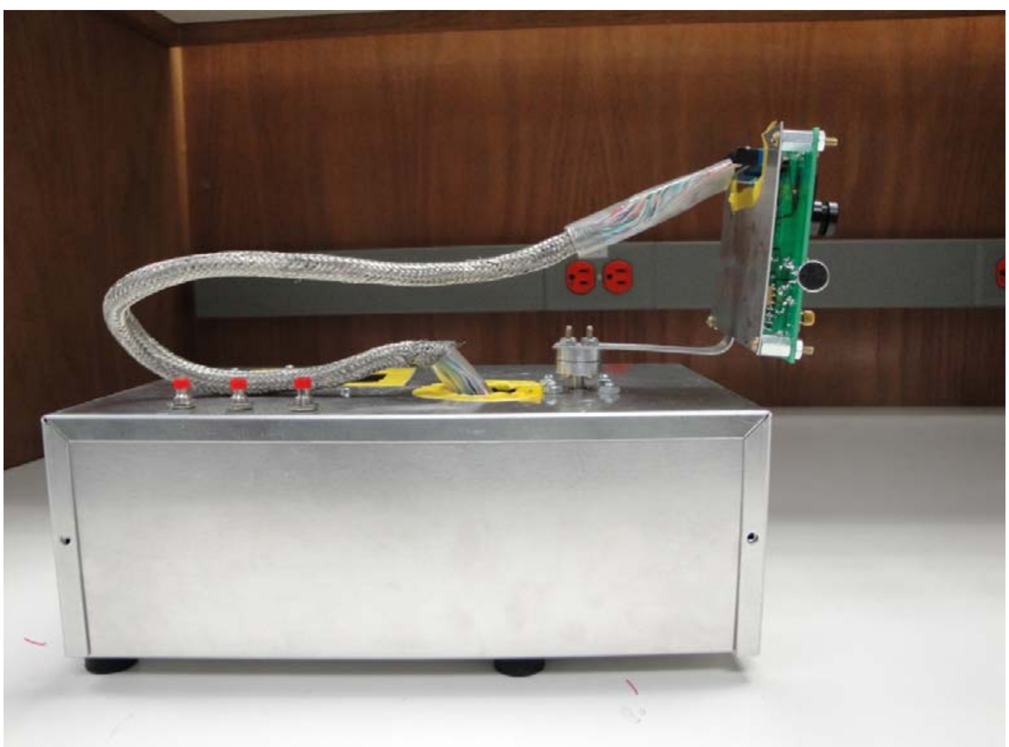


Figure 1- Side View


**Figure 2- Front View**


**Figure 3- Side View (motor has moved)**

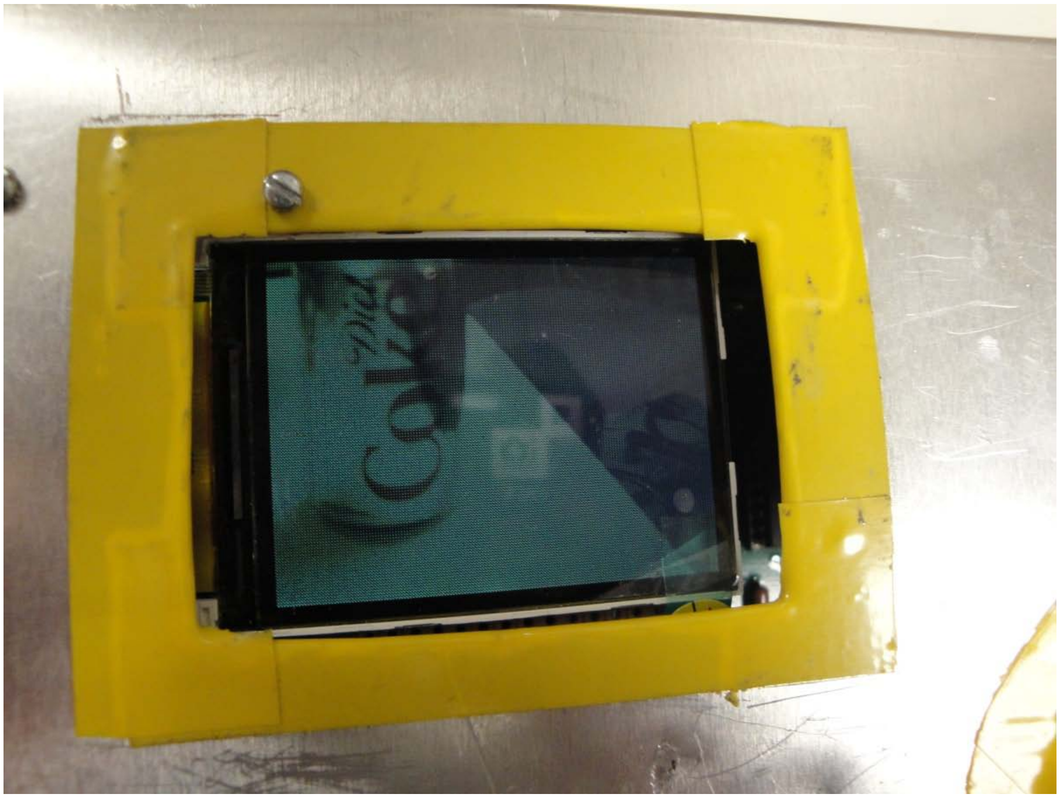


Figure 4- LCD with image on it

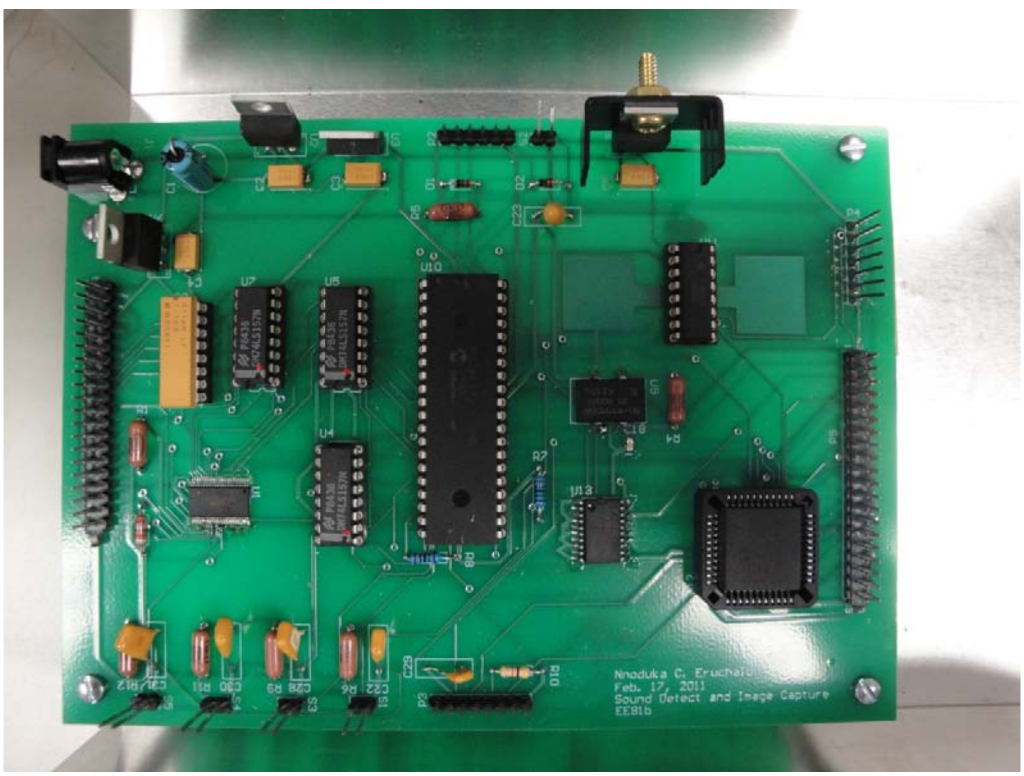



Figure 5- Main PCB

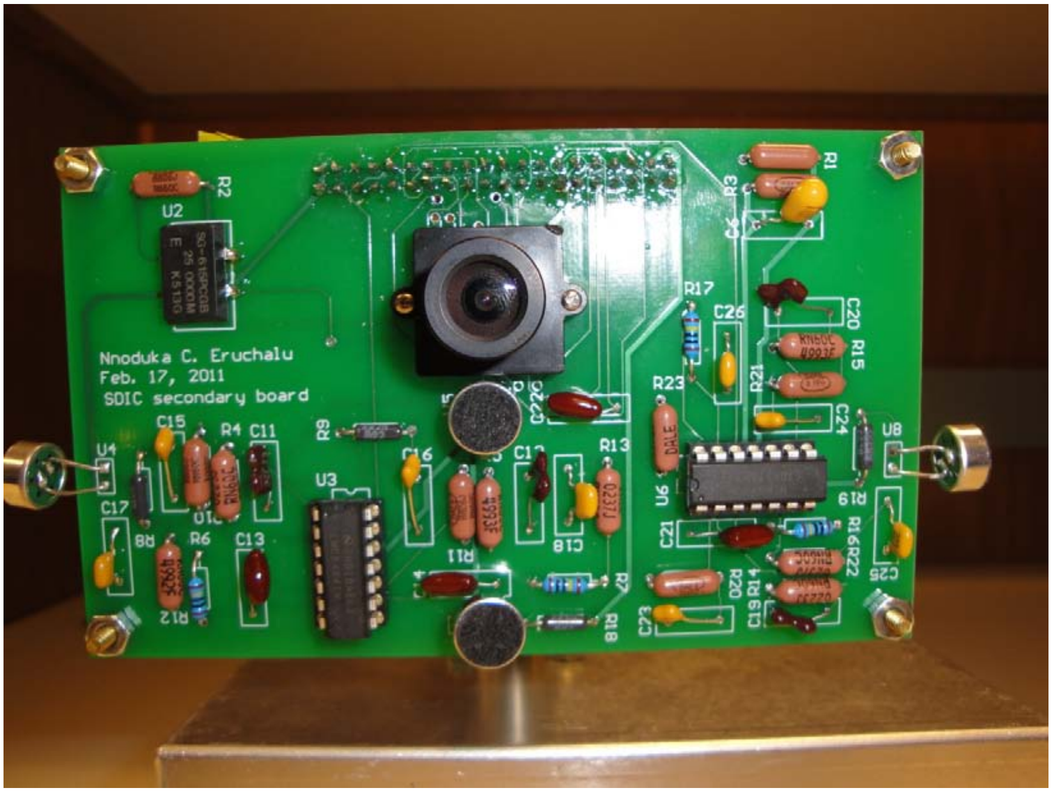



**Figure 6- Secondary PCB**

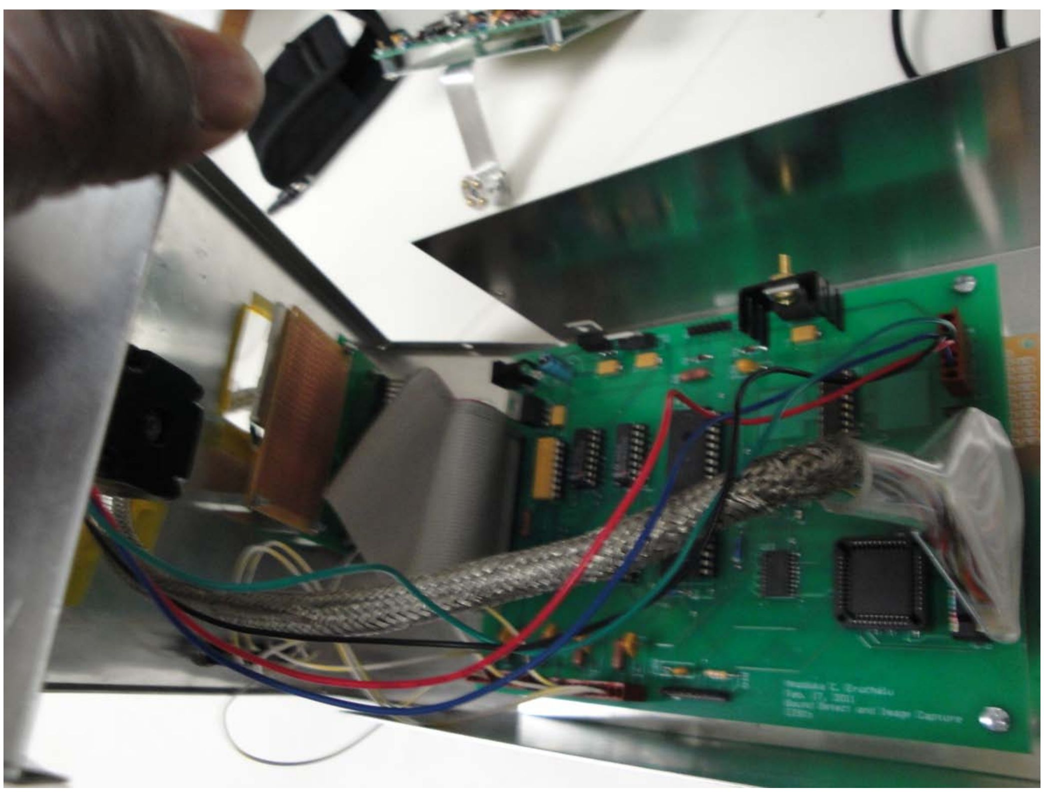


**Figure 7- All connections made, and about to close up product**

# Detailed Project Description

## Component Listing

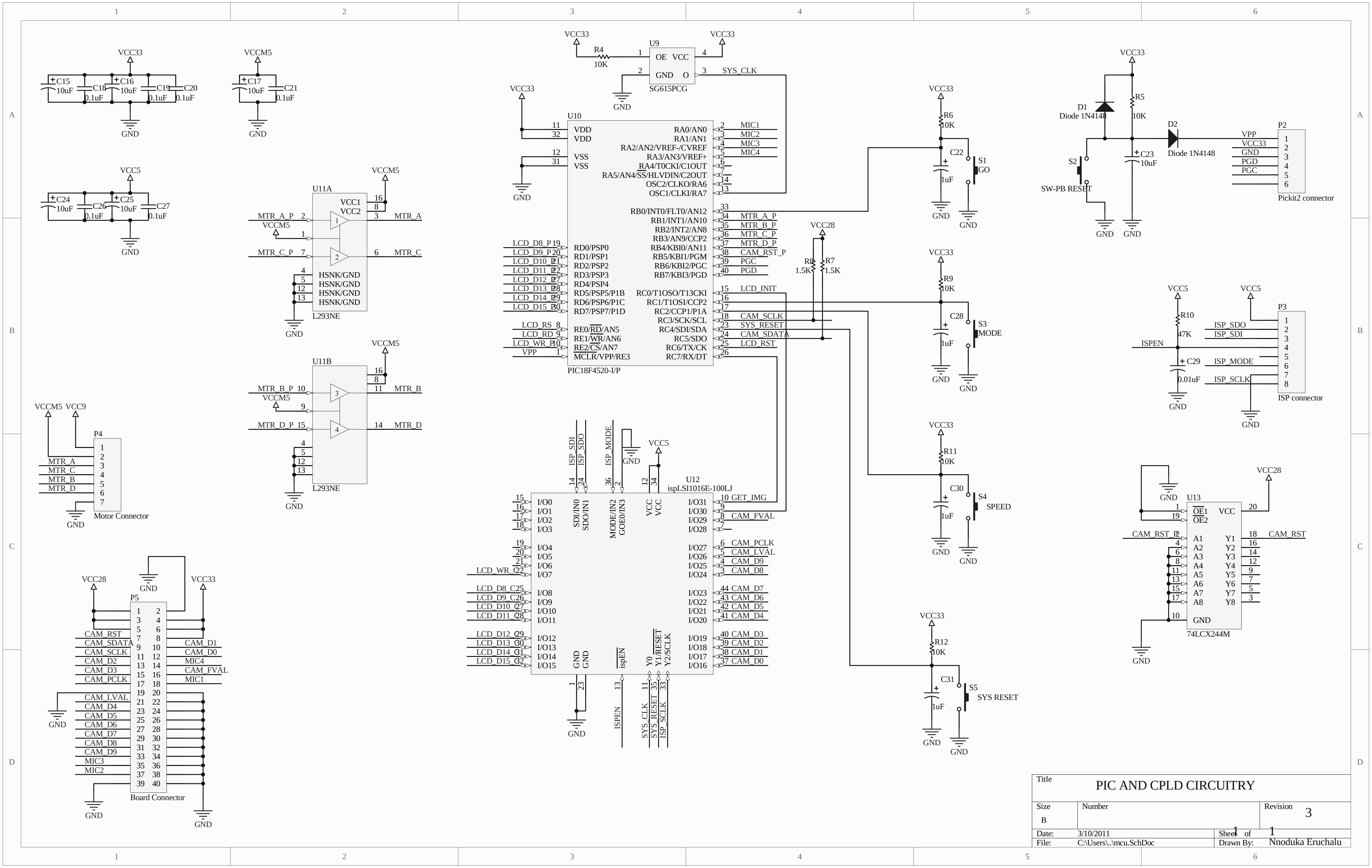
Below is a table showing the Bill of Materials used for the main PCB:

| Comment | Description | Designator | Footprint | LibRef | Quantity |
|---|---|---|---|---|---|
| 47uF | Polarized Capacitor (Radial) | C1 | RB.2/.4X | Cap Pol1 | 1 |
| 10uF | Polarized Capacitor (Radial) | C2, C3, C4, C5, C15, C16, C17, C24, C25 | TC7343-2917 | Cap Pol1 | 9 |
| 0.1uF | Capacitor | C6, C7, C8, C9, C10, C11, C12, C13, C14, C18, C19, C20, C21, C26, C27 | C0805 | Cap | 15 |
| 1uF | Polarized Capacitor (Radial) | C22, C28, C30, C31 | RAD-0.3 | Cap Pol1 | 4 |
| 10uF | Polarized Capacitor (Radial) | C23 | RAD-0.3 | Cap Pol1 | 1 |
| 0.01uF | Polarized Capacitor (Radial) | C29 | RAD-0.3 | Cap Pol1 | 1 |
| Diode 1N4148 | 1 Amp General Purpose Rectifier | D1, D2 | DIO7.4-4x2 | Diode 1N4007 | 2 |
| WALL POWER | Low Voltage Power Supply Connector | J1 | PWR2.1 | PWR2.5 | 1 |
| LCD connector | Header, 20-Pin, Dual row | P1 | HDR2X20 | Header 20X2 | 1 |
| Pickit2 connector | Header, 6-Pin | P2 | HDR1X6 | Header 6 | 1 |
| ISP connector | Header, 8-Pin | P3 | HDR1X8 | Header 8 | 1 |
| Motor Connector | Header, 7-Pin | P4 | HDR1X7 | Header 7 | 1 |
| Board Connector | Header, 20-Pin, Dual row | P5 | HDR2X20 | Header 20X2 | 1 |
| 10K | Resistor | R1, R4, R5, R6, R9, R11, R12 | AXIAL-0.4 | Res1 | 7 |
| 180 | Resistor | R2 | AXIAL-0.4 | Res1 | 1 |
| Res Pack3 | Isolated Resistor Network | R3 | DIP-16 | Res Pack3 | 1 |
| 1.5K | Resistor | R7, R8 | AXIAL-0.4 | Res1 | 2 |
| 47K | Resistor | R10 | AXIAL-0.4 | Res1 | 1 |
| GO | Switch | S1 | HDR1X2 | SW-PB | 1 |
| SW-PB RESET | Switch | S2 | HDR1X2 | SW-PB | 1 |
| MODE | Switch | S3 | HDR1X2 | SW-PB | 1 |
| SPEED | Switch | S4 | HDR1X2 | SW-PB | 1 |
| SYS RESET | Switch | S5 | HDR1X2 | SW-PB | 1 |
| 74LVT16245 | 16-Bit Bus Transceiver | U1 | TSSOP48 | 74LVT16245 | 1 |
| LM1086IT-2.85 | 1.5A Low Dropout Positive Regulators | U2 | T03B | LM1086CT-ADJ | 1 |
| LM1086CT-5.0 | 1.5A Low Dropout Positive Regulators | U3, U8 | T03B | LM1086CT-5.0 | 2 |
| DM74LS157N | Quad 2-Line to 1-Line Data Selector/Multiplexer | U4, U5, U7 | N16E | DM74LS157N | 3 |
| LM1086CT-3.3 | 1.5A Low Dropout Positive Regulators | U6 | T03B | LM1086CT-3.3 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| SG615PCG | | U9 | SG615PCG | SG615PCG | 1 |
| PIC18F4520-I/P | Enhanced Flash Microcontroller with 10-Bit A/D and nanoWatt Technology, 32K Flash, 40-Pin PDIP, Industrial Temperature Range | U10 | PDIP600-P40 | PIC18F4520-I/P | 1 |
| L293NE | Quadruple Half-H Driver | U11 | NE016 | L293NE | 1 |
| ispLSI1016E-100LJ | In-System Programmable High-Density PLD | U12 | PLCCS44 | ispLSI1016E-100LJ | 1 |
| 74LCX244M | Low Voltage CMOS Octal Bus Buffer (3-State) with 5V Tolerant Inputs and Outputs | U13 | SO-20L | 74LCX244M | 1 |

Below is a table showing the Bill of Materials used for the secondary PCB:

| Comment | Description | Designator | Footprint | LibRef | Quantity |
|---|---|---|---|---|---|
| 10uF | Polarized Capacitor (Radial) | C1, C7, C8 | TC7343-2917 | Cap Pol1 | 3 |
| 0.1uF | Capacitor | C2, C3, C4, C5, C9, C10 | C0805 | Cap | 6 |
| 1uF | Capacitor | C6 | RAD-0.3 | Cap | 1 |
| 24pF | Capacitor | C11, C12, C19, C20 | RAD-0.3 | Cap | 4 |
| 60pF | Capacitor | C13, C14, C21, C22 | RAD-0.3 | Cap | 4 |
| 4.7uF | Capacitor | C15, C16, C17, C18, C23, C24, C25, C26 | RAD-0.3 | Cap | 8 |
| Board Connector | Header, 20-Pin, Dual row | P1 | HDR2X20 | Header 20X2 | 1 |
| 10K | Resistor | R1, R2, R3, R10, R11, R12, R13, R20, R21, R22, R23 | AXIAL-0.4 | Res1 | 11 |
| 500K | Resistor | R4, R5, R14, R15 | AXIAL-0.4 | Res1 | 4 |
| 200K | Resistor | R6, R7, R16, R17 | AXIAL-0.4 | Res1 | 4 |
| 2.2K | Resistor | R8, R9, R18, R19 | AXIAL-0.4 | Res1 | 4 |
| MT9V011 | | U1 | MT9V011 | MT9V011 | 1 |
| SG615PCG | | U2 | SG615PCG | SG615PCG | 1 |
| LMC6484IN | CMOS Quad Rail-to-Rail Input & Output Operational Amplifier | U3, U6 | N14A | LMC6484IN | 2 |
| mic | | U4, U5, U7, U8 | HDR1X2 | mic | 4 |



PIC AND CPLD CIRCUITRY
Size B
Revision 3
Date: 3/10/2011
File: C:\Users\..\mcu.SchDoc
Sheet 1 of 1
Drawn By: Nnoduka Eruchalu
U9 SG615PCG
OE VCC GND O
SYS_CLK
R4 10K
U10 PIC18F4520-I/P
VDD VDD VSS VSS
RA0/AN0
RA1/AN1
RA2/AN2/VREF-/CVREF
RA3/AN3/VREF+
RA4/T0CKI/C1OUT
RA5/AN4/SS/HLVDIN/C2OUT
OSC2/CLKO/RA6
OSC1/CLKI/RA7
RB0/INT0/FLT0/AN12
RB1/INT1/AN10
RB2/INT2/AN8
RB3/AN9/CCP2
RB4/KBI0/AN11
RB5/KBI1/PGM
RB6/KBI2/PGC
RB7/KBI3/PGD
RC0/T1OSO/T13CKI
RC1/T1OSI/CCP2
RC2/CCP1/P1A
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0 RD1/PSP1 RD2/PSP2 RD3/PSP3 RD4/PSP4
RD5/PSP5/P1B RD6/PSP6/P1C RD7/PSP7/P1D
RE0/RD/AN5 RE1/WR/AN6 RE2/CS/AN7 MCLR/VPP/RE3
MIC1 MIC2 MIC3 MIC4
MTR_A_P MTR_B_P MTR_C_P MTR_D_P
CAM_RST_P PGC PGD
LCD_INIT CAM_SCLK SYS_RESET CAM_SDATA LCD_RST
U11A L293NE
U11B L293NE
VCC1 VCC2 HSNK/GND
MTR_A MTR_B MTR_C MTR_D
P4 Motor Connector
P5 Board Connector
U12 ispLSI1016E-100LJ
SDI/IN0 SDO/IN1 MODE/IN2 GOE0/IN3
I/O0 … I/O31
ispEN Y0 Y1/RESET Y2/SCLK
ISP_SDI ISP_SDO ISP_MODE ISP_SCLK ISPEN
GET_IMG CAM_FVAL CAM_PCLK CAM_LVAL
CAM_D0 … CAM_D9
LCD_D8_P … LCD_D15_P
LCD_RS_P LCD_RD_P LCD_WR_P
S1 GO
S2 SW-PB RESET
S3 MODE
S4 SPEED
S5 SYS RESET
D1 Diode 1N4148
D2 Diode 1N4148
P2 Pickit2 connector
VPP VCC33 GND PGD PGC
P3 ISP connector
U13 74LCX244M
OE1 OE2 A1–A8 Y1–Y8 VCC GND
CAM_RST
VCC33 VCC5 VCCM5 VCC9 VCC28 GND
C15 10uF C16 10uF C17 10uF C18 0.1uF C19 0.1uF C20 0.1uF C21 0.1uF
C22 1uF C23 10uF C24 10uF C25 10uF C26 0.1uF C27 0.1uF
C28 1uF C29 0.01uF C30 1uF C31 1uF
R5 10K R6 10K R7 1.5K R8 1.5K R9 10K R10 47K R11 10K R12 10K

```
DESCRIPTION OF THE MICROCONTROLLER UNIT SCHEMATIC

The digital control section of this project comprises of:
1. PIC microcontroller (U10, PIC18F4520)
2. Lattice CPLD (U12, ispLSO1016E)
3. System Control Interface (Switches)
4. Motors Interface (using U11, L293(D))
5. Camera Interface (i2c interface, control, data and clock lines, U13-74LCX244)
6. LCD Interface (control and data lines)

--------------------------------------------------------------------------------
1. Description of the PIC microcontroller:
This schematic shows how the PIC18F4520 is setup for in-circuit serial
programming (icsp) and for general use in this project.
Diode D2 stops high Vpp causing contention.

The icsp is done using a pickit2.

The pins on the pickit2 connector (in order from 1 to 6) are:
VPP      - Programming voltage (using 13V)
VCC33    - 3.3V power supply
GND      - Ground
PGD      - Data
PGC      - Clock
PGM      - LVP enable (left unconnected)

Note that PGD and PGC are left as dedicated programming pins so as to not
interfere with the rest of the circuit during in-circuit programming.

The PIC microcontroller is used to control the LCD by writing to the DataBus
and the Control signals, and it receives instructions via the switches.



--------------------------------------------------------------------------------
2. Description of the Lattice CPLD
This schematic shows how the ispLSO1016E is setup for in-system programming
(isp) and for general use in this project.

The isp is done using the appropriate ispDOWNLOAD cable and the ispVM software

The pins on the isp connector (in order from 1 to 8) are:
VCC5     - 5V power supply
SDO      - Used to shift data out via the IEEE1149.1 (JTAG) programming standard
SDI      - Used to shift data in via the IEEE1149.1 programming standard
ispEN\   - Enable device to be programmed
NC
MODE    - Used to control the IEEE1149.1 state machine
GND     - Connect to ground plane of the target device
SCLK    - Used to clock the IEEE1149.1 state machine



--------------------------------------------------------------------------------
3. Description of the System Control Interface (Switches):
There are 4 switches:
```

```
GO    - On a button press, the microphones start spinning to detect a sound and
        take a still picture or get a continuous video.
MODE  - Each button press toggles the display mode of the detected sound. The
        Two mode options are still image or continuous video.
SPEED - UNIMPLEMENTED
RESET - This is the system reset button.
```

--------------------------------------------------------------------------------

4. Description of the Motors Interface:
The Motor used in this project is a bipolar stepper motor. So an L293(D) (U11) is used to drive the motor at the high currents it requires.
Note that if an L293D is used then there will be no need for protection diodes, but if an L293 is being used then the protection diode board will need to be plugged in at the connection point (P4)

Note that the motor connector P4 has a pin for the 9V board power supply, as well as the 5V supply dedicated to the motor (VCCM5), and a GND line.
The other 4 pins on the motor connector are for the 4 terminals of the stepper motor coils.

--------------------------------------------------------------------------------

5. Description of Camera Interface:
The camera interface consists of the i2c interface, control, data and clock lines.

the i2c interface is implemented inside the PIC using the PICs SCL and SDO data lines. It should be noted however that those pins were only used for traditional reasons. The project does not use the PIC's i2c module but instead implements its own, which does work on any other pins.
The important part of this interface is the 1.5KOhm pullups to the 2.8V power supply. The code is written such that the high values on the interface will be the 2.8V used by the camera and not the 3.3V used by the PIC. The way this works is that to get a high voltage on the line, the PIC makes the pin go into a high impedance state, thus the SCL or SDA line will then float to 2.8V.

The only other control line going to the camera is the reset, CAM_RST. This is sent through the 74LCX244 (U13), which has 5V tolerant inputs (good enough for the 3.3V input coming from the PIC) but outputs using 2.8V

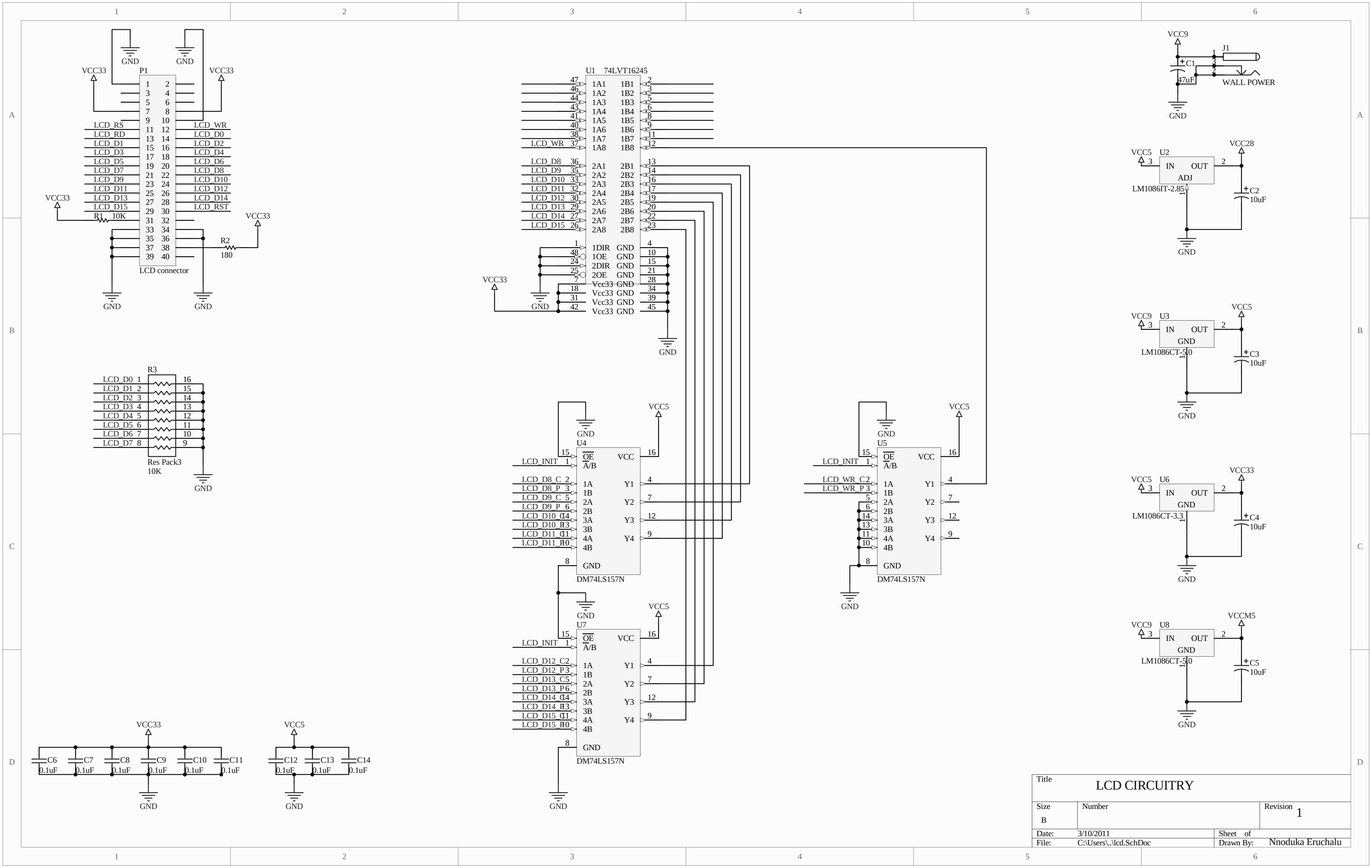
The data and clock lines output from the camera all go directly to the CPLD, which uses this info to output images appropriately unto the LCD.

--------------------------------------------------------------------------------

6. Description of LCD Interface:
The LCD interface consists of control and data lines.

Both the PIC and the CPLD write to the LCDs control and data lines (which are all inputs to the LCD). This control by two different components is sync'd by the PIC, in the signals it sends to the CPLD and the MUXes
(in the LCD circuitry).

```
--------------------------------------------------------------------------
```



Title
LCD CIRCUITRY
Size
B
Number
Revision
1
Date: 3/10/2011
Sheet of
File: C:\Users\..\lcd.SchDoc
Drawn By: Nnoduka Eruchalu

DESCRIPTION OF THE LCD SCHEMATIC

The lcd section of this project comprises of:
1. A TFT LCD module (NHD-2.4-240320SF-CTXI connects at P1)
2. MUXes (74LS157- U4, U5, U7)
3. Buffer (74LVT16245- U1)

The power section of this project comprises of:
4. Wall Power
5. Voltage Regulators

--------------------------------------------------------------------------------
1. Description of the TFT LCD module:
The datasheet can be found at
http://www.newhavendisplay.com/specs/NHD-2.4-240320SF-CTXI.pdf

This is a 2.4" Graphic LCD with a 320x240 resolution. It has a backlight.

The LCD is only used in 8 bit mode, which means we use D8..D15.
So D0..D7 are unused and should not be left floating so they are pulled down usng the 8-resistor resistor pack, R3.
These unused data lines are pulled down (and NOT pulled up!) because they could be outputs if the LCD is read from.

--------------------------------------------------------------------------------
2. Description of the MUXes:
Both the PIC and the CPLD try to control the LCDs data lines and the strobe the Write line. So the PIC sync's up all this by the use of the LCD_INIT signal which is also the mux select signal. This way the PIC gets to decide whose control and data lines go through the MUXes and into the LCD.
There are a grand total of 9 lines being multiplexed, hence the use of 3 4-bit muliplexers (74LS157)

The muxes are always enabled so OE\ is tied to ground for all 3 of them.

--------------------------------------------------------------------------------
3. Description of the 16-Bit Buffer:
The 74LS157 muxes are 5V chips, but the LCD runs at 3.3V. So the 9 output lines from the 3 multiplexers need to be passed through 3.3V buffers which have 5V tolerant inputs. To use only one buffer for all the LCD lines, the 16-bit buffer, 74LVT16245, is used.

This buffer is actually bidirectional, but the direction is to be fixed, as all the signals going to the LCD are LCD inputs. The direction is fixed from B->A, and so 1DIR and 2DIR are grounded. We obviously want both halves of this 16-bit transceiver to go in the same B->A direction.

Also we want both halves of this 16-bit buffer to be always active. So 1OE and 2OE are also grounded.

--------------------------------------------------------------------------------

4. Description of the Wall Power:
This is a 2.1mm power jack connector. The supplied power supply for this project outputs 9V. There is a 47uF bypass capacitor across this 9V power line.
This is to prevent droops in voltage. For this reason this capacitor is preferrably electrolytic.

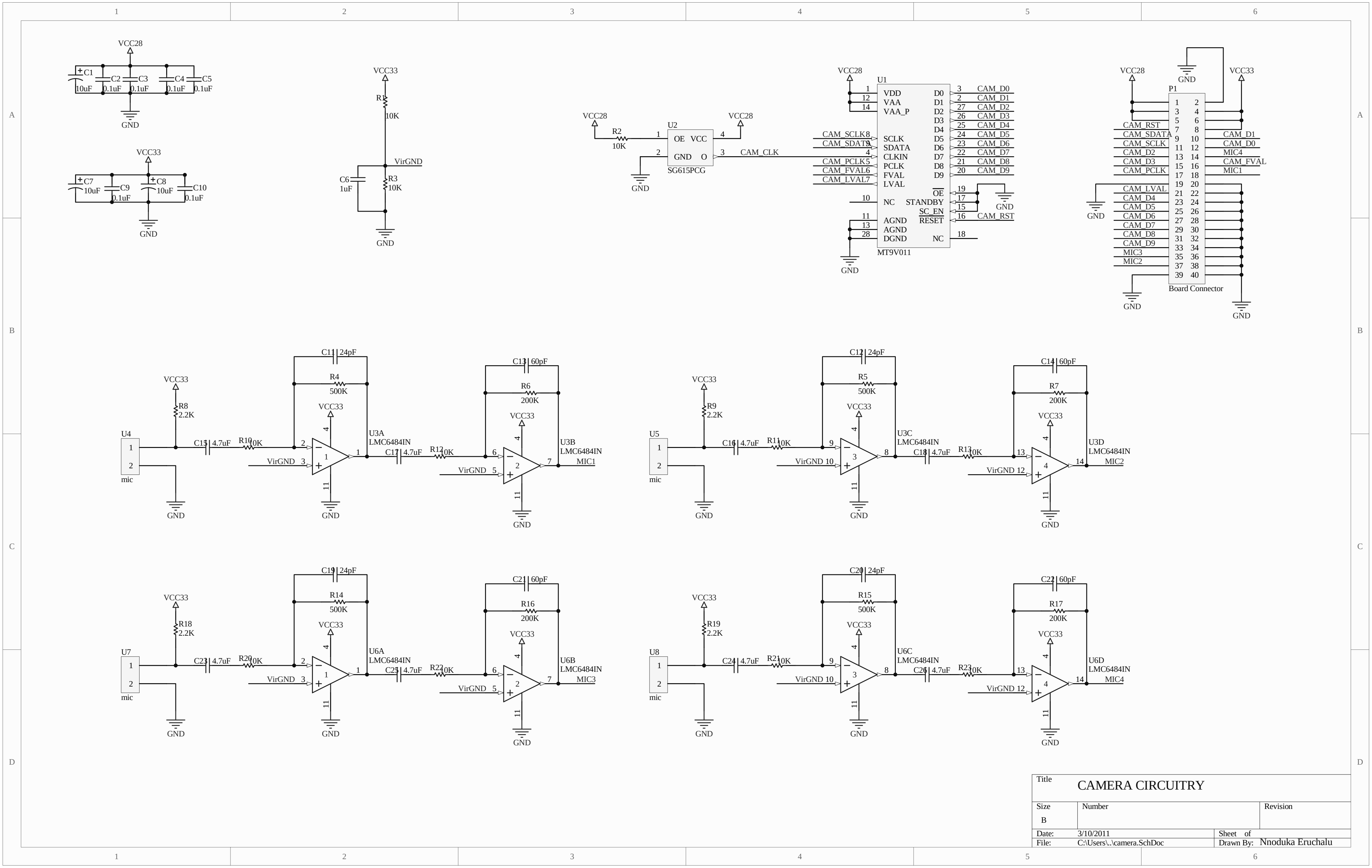
--------------------------------------------------------------------------------
5. Description of the Voltage Regulators
The 9V output of the 2.1mm power jack is passed into two 5V regulators, LM1086CT-5.0, U3 and U8. U3 outputs VCC5 and U8 outputs VCCM5.
VCC5 is the 5V supply for all the 5V components in this product. VCCM5 is the 5V supply for the servo motor and its motor driver. The motor section gets its own 5V supply so that it wont cause noise in the digital circuitry. Another reason for VCCM5 being separate from VCC5 is because the motor section draws a lot of current which is close to the limit of any one LM1086CT-5.0 regulator.

VCC5 is passed into a 3.3V regulator, LM1086CT-3.3 to generate VCC33, the 3.3V supply for all the 3.3V components in this product.

VCC5 is passed into a 2.85V regulator, LM1086CT-2.85 to generate VCC28, the 2.8V supply for all the 2.8V components in this product. In this product 2.85V has been assumed approximate to 2.8V

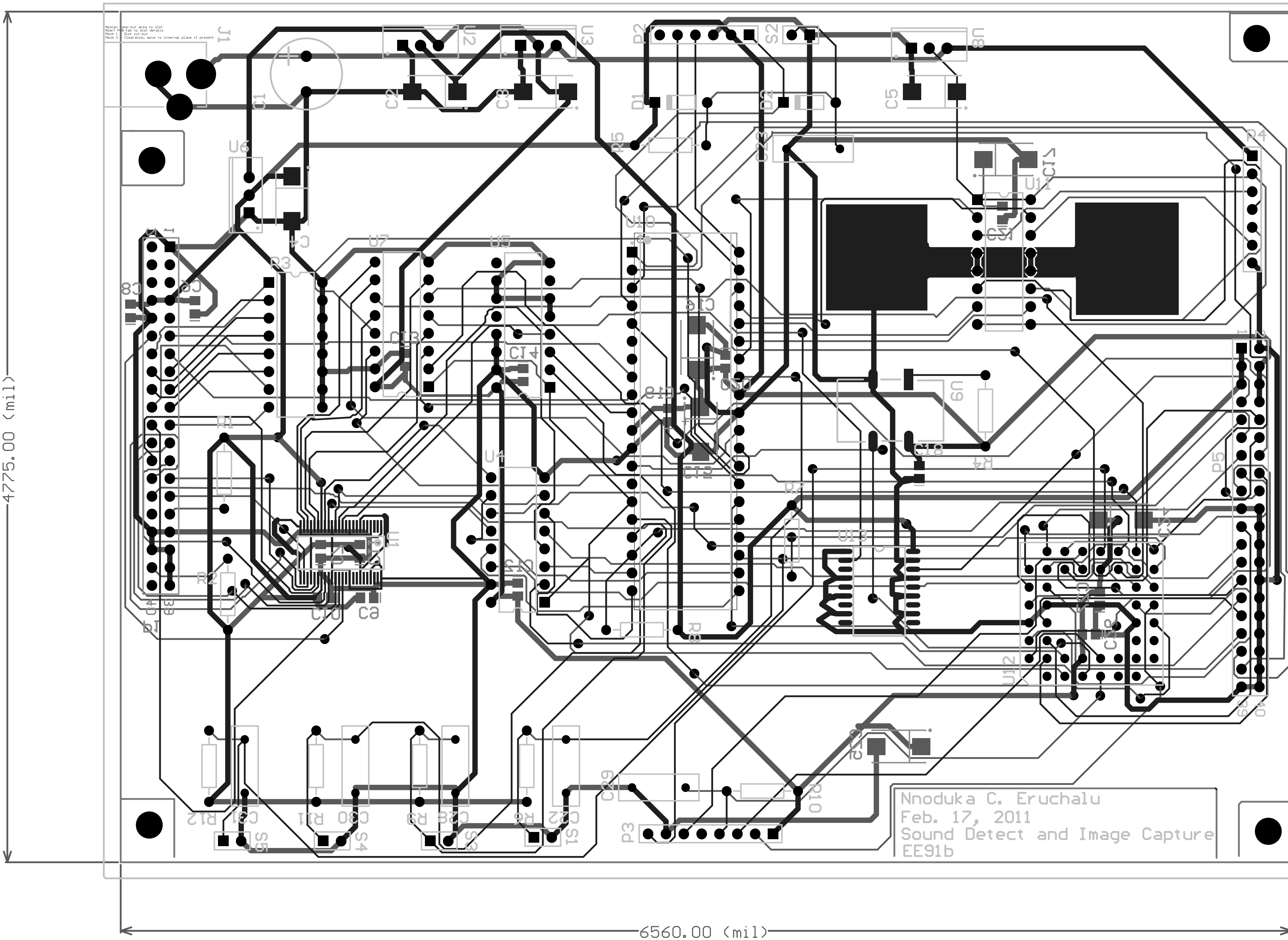
All voltage regulators use 10uF bypass capacitors to block noise in the voltages.

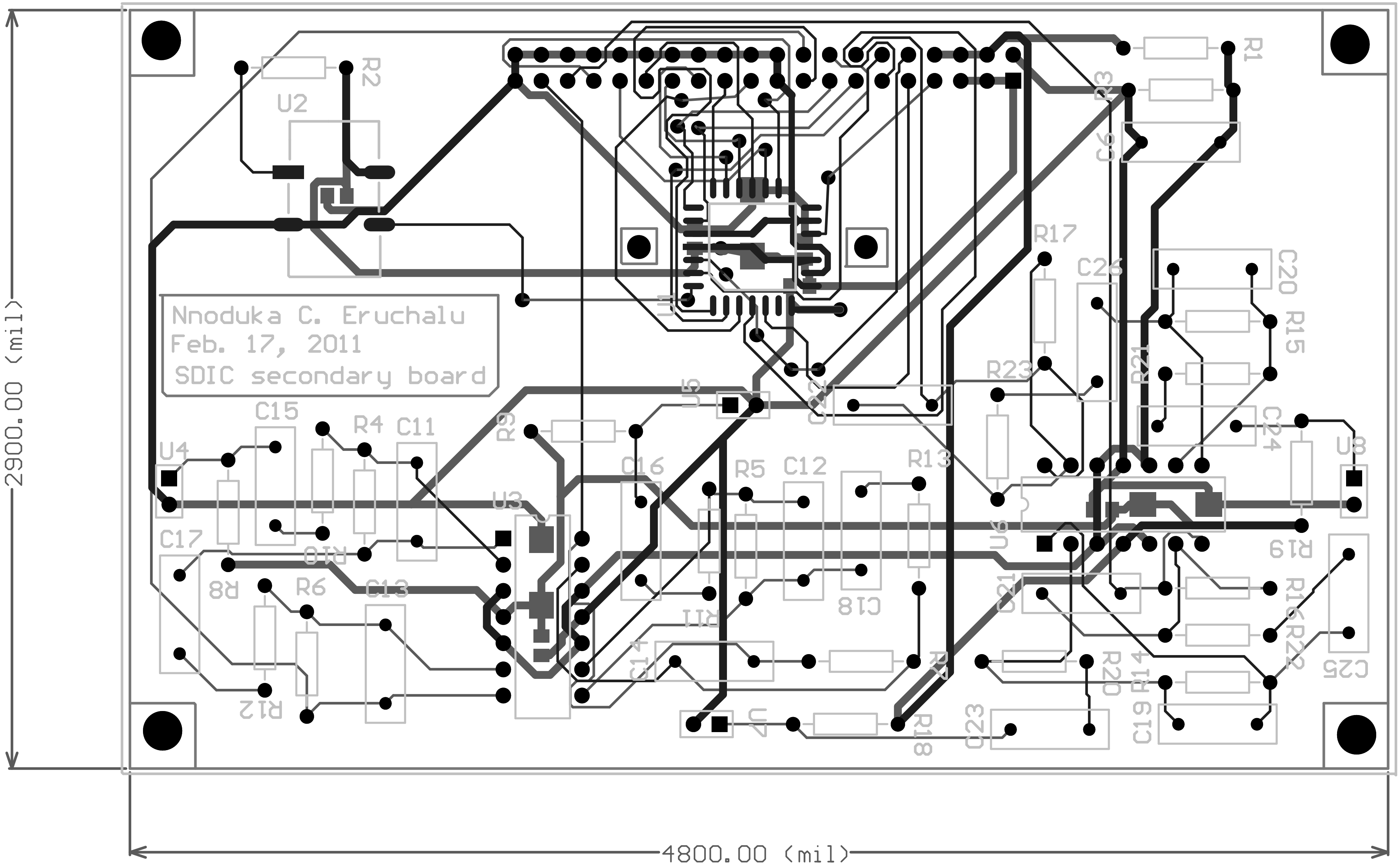
--------------------------------------------------------------------------------


VCC28
VCC33
GND
VirGND
U1
MT9V011
U2
SG615PCG
CAM_CLK
P1
Board Connector
U3A LMC6484IN
U3B LMC6484IN
U3C LMC6484IN
U3D LMC6484IN
U6A LMC6484IN
U6B LMC6484IN
U6C LMC6484IN
U6D LMC6484IN
U4 mic
U5 mic
U7 mic
U8 mic
MIC1
MIC2
MIC3
MIC4
Title
CAMERA CIRCUITRY
Size
B
Number
Revision
Date: 3/10/2011
Sheet of
File: C:\Users\..\camera.SchDoc
Drawn By: Nnoduka Eruchalu

```
DESCRIPTION OF THE CAMERA SCHEMATIC

The camera section of this project comprises of:
1. Clock (SG615PCG- U2)
2. Image Sensor (MT9V011- U1)
3. Microphone Section (LMC6484IN- U3, U6;   electret mics- U4, U5, U7, U8)



--------------------------------------------------------------------------------
1. Description of the Clock:
The clock is 25MHz, and it is always enabled so OE is pulled high.
It is to output a clock signal at 2.8V so it's VCC line is connected to VCC28.

This runs the image sensor so it's output is connected to the image sensor chip.

--------------------------------------------------------------------------------
2. Description of the image sensor:
This is a CMOS image sensor which has a resolution of 640x480, but it can be
reprogrammed to a different window size. For this project it was reprogrammed
to 320x240 to match the graphic LCD.

The output pixel data rate is also reprogrammable and for this project it was
reprogrammed to be 1/10th the input clock rate of 25MHz. This gives the CPLD
enough time to write this to the LCD while meeting all timing requirements.

This CMOS image sensor runs at 2.8V so its power lines are connected to VCC28.

It is always enabled so OE\, standby, and SC_EN are all tied to ground.



--------------------------------------------------------------------------------
3. Description of the microphone section.
There are 4 identical microphone sections as can be seen in the schematics, so
only one will be discussed.

U7 is an electret microphone that has to be biased by a 2.2 pullup resistor,
R18. The output of this is then amplified by a two-stage op-amp amplifier.
The required gain is 1000, but the Gain-Bandwidth Product of the LMC6484IN
running at 3.3V is 1.0MHz. So at a gain of 1000 will have a maximum frequency of
1KHz.

This will not work for the audio range of 0 - 20KHz.
To get the full audio range the maximum gain should be 50 so as to fit in the
GBW of 1.0MHz.

Thus two inverting op-amps are used to get a gain of 1000. One with a gain of 50
and another with a gain of 20.

Both of these are configured as bandpass filters.
The bandpass filters further enforce the filtering of the electret microphone
which has an audio range of 100Hz to 12KHz.This filtering is to account for the
fact that when signals are amplified there is always high frequency noise
created by the op-amps which is eliminated by a low-pass filter. Also the op-amp
creates a DC offset which is eliminated by the high-pass filter. This combined
```

```
high-pass and low-pass filter is a bandpass filter.

For the first inverting op-amp, U6A, the gain is -500/10 = -50.
The low-pass cutoff is 1//(2*pi*R14*C24) = 1/(2*pi*500K*24p) = 12.262 KHz
The high-pass cutoff is 1/(2*pi*R20*C23) = 1/(2*pi*10K*0.1u) = 159.15 Hz

For the second inverting op-amp, U6B, the gain is 200/10 = -20, bringing total
gain to -50 * -20 = 1000.
The low-pass cutoff is 1/(2*pi*R16*C21) = 1/(2*pi*1M*12p) = 12.262 KHz
The high-pass cutoff is 1/(2*pi*R22*C25) = 1/(2*pi*50K*0.02u) = 159.15 Hz

--------------------------------------------------------------------------------
```


Nnoduka C. Eruchalu
Feb. 17, 2011
Sound Detect and Image Capture
EE91b
4775.00 (mil)
6560.00 (mil)


Nnoduka C. Eruchalu
Feb. 17, 2011
SDIC secondary board
2900.00 (mil)
4800.00 (mil)
U1
U2
U3
U4
U5
U6
U7
U8
R1
R2
R3
R4
R5
R6
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19
R20
R21
R22
R23
C6
C11
C12
C13
C14
C15
C16
C17
C18
C19
C20
C21
C22
C23
C24
C25
C26

```
MODULE      sdicintf
TITLE      'SDIC Interface'


" SDICInterface  DEVICE  'ispLSI1016E'

"
"            Sound Detect and Image Capture
"                    EE91b, 2011
"
" Description:  Interfaces the Camera and the LCD with the PIC18F4520
"               MicroController running at 25 MHz.
"
" Assumptions:  Camera runs at a factor of 10 slower than CPLD.
"               10 was chosen as a safe bet for the 9 states (not counting Idle)

" Revision History:
"   Feb. 6 2011   Nnoduka Eruchalu     Initial revision
"   Feb. 11 2011  Nnoduka Eruchalu     Finally got it working!



" Pins

"GND        pin 1                                 supply power ground
"           pin 2                    input      GOE (should be tied to ground)
Cin8        pin 3;                  "input      Camera DB8
Cin9        pin 4;                  "input      Camera DB9
Cam_lval_in pin 5;                  "input      Camera Line Valid
Cam_pclk_in pin 6;                  "input      Camera Pixel Clock
"           pin 7                    in/out     unused
Cam_fval_in pin 8;                  "input      Camera Frame Valid
Lcd_init_in pin 9;                  "input      Is LCD being initialized?
Get_img_in  pin 10;                 "input      Get Image Command from the PIC18
Clock       pin 11;                 "input      System Clock (24MHz)
"VCC        pin 12                              supply power Vcc
"!ISPEN     pin 13                   input      ISP enable
"ISP_SDI    pin 14                   input      ISP SDI
"           pin 15                   in/out     unused
"           pin 16                   in/out     unused
"           pin 17                   in/out     unused
"           pin 18                   in/out     unused
"           pin 19                   in/out     unused
"           pin 20                   in/out     unused
"           pin 21                   in/out     unused
Lcd_wr      pin 22 ISTYPE 'reg';    "output     LCD Write Line (Active Low)
"GND        pin 23                   supply     power ground
"ISP_SDO    pin 24                   input      ISP SDO
L8..L15     pin 25..32 ISTYPE 'com';"output     LCDs DataBus
"ISP_SCLK   pin 33                   input      ISP SCLK
"VCC        pin 34                              supply  power Vcc
"!SYS_RESET pin 35                   input      reset (from SYS_RESET switch)
"ISP_MODE   pin 36                   input      ISP MODE
Cin0..Cin7  pin 37..44;             "input      Camera Databus bits 0 to 7
```

```
" Internal Nodes

" synchronize to clocks
Cam_lval    node  ISTYPE 'reg';
Cam_lval1   node  ISTYPE 'reg';
Cam_pclk    node  ISTYPE 'reg';
Cam_pclk1   node  ISTYPE 'reg';
Cam_fval    node  ISTYPE 'reg';
Cam_fval1   node  ISTYPE 'reg';
Lcd_init    node  ISTYPE 'reg';
Get_img     node  ISTYPE 'reg';
C0..C9      node  ISTYPE 'reg';

" Row Counter (bits 0 to 7) 240 rows max

Row0        node  ISTYPE 'reg';     "row counter bit 0
Row1        node  ISTYPE 'reg';     "row counter bit 1
Row2        node  ISTYPE 'reg';     "row counter bit 2
Row3        node  ISTYPE 'reg';     "row counter bit 3
Row4        node  ISTYPE 'reg';     "row counter bit 4
Row5        node  ISTYPE 'reg';     "row counter bit 5
Row6        node  ISTYPE 'reg';     "row counter bit 6
Row7        node  ISTYPE 'reg';     "row counter bit 7


" Column Counter (bits 0 to 8) 320 columns max

Col0        node  ISTYPE 'reg';     "column counter bit 0
Col1        node  ISTYPE 'reg';     "column counter bit 1
Col2        node  ISTYPE 'reg';     "column counter bit 2
Col3        node  ISTYPE 'reg';     "column counter bit 3
Col4        node  ISTYPE 'reg';     "column counter bit 4
Col5        node  ISTYPE 'reg';     "column counter bit 5
Col6        node  ISTYPE 'reg';     "column counter bit 6
Col7        node  ISTYPE 'reg';     "column counter bit 7
Col8        node  ISTYPE 'reg';     "column counter bit 8


" Row and Column boundary check flags, should be NODES
ColEq320    node  ISTYPE 'com';     "at column 320 {1-indexed}
RowEq240    node  ISTYPE 'com';     "at row 240 {1-indexed}

ColEq1      node  ISTYPE 'com';     "at column 1 {1-indexed}
RowEq1      node  ISTYPE 'com';     "at row 1 {1-indexed}

" Pixel Color flags, should be NODES
RedPx       node  ISTYPE 'com';     "at a red pixel
GreenPx     node  ISTYPE 'com';     "at a green pixel
BluePx      node  ISTYPE 'com';     "at blue pixel

" Latched Data from camera
Cdb0..Cdb5  node  ISTYPE 'reg';
```

```
" What is coming up, LCD High or Low Byte write?
Lcd_l_byte  pin 17  ISTYPE 'reg';

" State machine state bits
St0         pin 16  ISTYPE 'reg';
St1         pin 15  ISTYPE 'reg';



" buses

RowCntr  =  [Row7..Row0];       " row counter
ColCntr  =  [Col8..Col0];       " column counter
Lcd_db   =  [L15..L8];          " LCD's Databus (external to CPLD)
Cam_db_E_in = [Cin9..Cin4];
Cam_db_E =  [C9..C4];           " External Camera's Databus
Cam_db   =  [Cdb5..Cdb0];       " Latched data from external Cam Databus




" shorthand for valid data and new frame
ValidData = (Cam_pclk == 1) & (Cam_pclk1 == 0) & (Cam_lval == 1);
"NewFrame = ValidData & RowEq1 & ColEq1;
NewFrame   = (Cam_fval == 1) & (Cam_fval1 == 0);


" state definitions for the image capture and display state machine
IMAGEBITS   = [ St1, Lcd_wr, Lcd_l_byte, St0];  " State bits
Idle        = [ 1,    1,      1,      1 ];  " Waiting to acquire img    15
WaitForFrame = [ 1,   1,      0,      0 ];  " Waiting for new frame     12
WaitForData = [ 1,    1,      0,      1 ];  " Waiting for valid data    13

OutH_Wr0    = [ 0,    0,      0,      0 ];  " Pulse Write line while    0
OutH_Wr00   = [ 0,    0,      0,      1 ];  " outputting high data      1
OutH_Wr1    = [ 0,    1,      0,      0 ];  " byte.                     4
OutH_Wr11   = [ 0,    1,      0,      1 ];  "                           5

OutL_Wr0    = [ 0,    0,      1,      0 ];  " Pulse Write Line while    2
OutL_Wr00   = [ 0,    0,      1,      1 ];  " outputting high data      3
OutL_Wr1    = [ 0,    1,      1,      0 ];  " byte.                     6
OutL_Wr11   = [ 0,    1,      1,      1 ];  "                           7

" Unused states
Rand0       = [ 1,    0,      0,      0 ];  "                           8
Rand1       = [ 1,    0,      0,      1 ];  "                           9
Rand2       = [ 1,    0,      1,      0 ];  "                           10
Rand3       = [ 1,    0,      1,      1 ];  "                           11
Rand4       = [ 1,    1,      1,      0 ];  "                           14



EQUATIONS
```

```
" output enables - enable the used outputs (registered outputs enabled
" by OE\ pin)

L8.OE   = 1;                  " state machine loads data itself
L9.OE   = 1;
L10.OE   = 1;
L11.OE   = 1;
L12.OE   = 1;
L13.OE   = 1;
L14.OE   = 1;
L15.OE   = 1;

Cam_lval.CLK    =   Clock;
Cam_lval1.CLK   =   Clock;
Cam_pclk.CLK    =   Clock;
Cam_pclk1.CLK   =   Clock;
Cam_fval.CLK    =   Clock;
Cam_fval1.CLK   =   Clock;
Lcd_init.CLK    =   Clock;
Get_img.CLK     =   Clock;
Cam_db_E.CLK    =   Clock;

Cam_lval    := Cam_lval_in;
Cam_pclk    :=  Cam_pclk_in;
Cam_pclk1   :=  Cam_pclk;
Cam_fval    :=  Cam_fval_in;
Cam_fval1   :=  Cam_fval;
Lcd_init    :=  Lcd_init_in;
Get_img     :=  Get_img_in;
Cam_db_E    :=  Cam_db_E_in;

" check for when line valid pulses
Cam_lval1   :=  Cam_lval;

" clocks for the registered outputs
" use the global clock pin
"   Row and Column Counter bits already taken care of.
"   All other registered outputs or nodes used as state bits.
IMAGEBITS.CLK   =   Clock;
Cam_db.CLK      =   Clock;




" Row Counter
"    8-bit counter
"    counts from 1 to 240
"    self clearing

" setup the clocks
RowCntr.CLK  =  Clock;

" do the counter
WHEN ((Cam_fval == 0) & (Cam_fval1 == 0))  THEN
    RowCntr = 0;
```

```
ELSE  WHEN ((Cam_lval == 1) & (Cam_lval1 == 0) & (Cam_fval == 1)) THEN
    RowCntr = RowCntr + 1;
ELSE
    RowCntr = RowCntr;

" the first row count (row is 1)
RowEq1 = (RowCntr == 1);

" the final count (row is 240)
RowEq240 = (RowCntr == 240);



" Column Counter
"    9-bit counter
"    counts from 0 to 320
"    self clearing

" setup the clocks

ColCntr.CLK  =  Clock;

" do the counter
WHEN ((Cam_lval == 0) & (Cam_lval1 == 0))  THEN
    ColCntr = 0;
ELSE  WHEN ((Cam_pclk == 1) & (Cam_pclk1 == 0) & (Cam_lval == 1)) THEN
    ColCntr = ColCntr + 1;
ELSE
    ColCntr = ColCntr;


" the first column count (column is 1)
ColEq1 = (ColCntr == 1);

" the final count (column is 320)
ColEq320  =  (ColCntr == 320);



" Figure out the color of the current pixel
GreenPx = !(Row0 $  Col0);      " Green when row and column are both even or odd
RedPx   =   Row0 & !Col0;       " Red when row is even and column is odd
BluePx  =  !Row0 &  Col0;       " Blue when row is odd and column is even




" Latch the Camera's data when Data is Valid
WHEN (ValidData) THEN
    Cam_db =   Cam_db_E;
ELSE
    Cam_db = Cam_db;
```

```
" Update the LCD's databus using this format:
"   The 16-bit data going to the LCD is RGB (5-6-5 format)
"   So the high byte is  RG (5-3 format) and the low byte is GB (3-5 format)
"   When on a particular pixel, make its bits equal to Cam_db and clear all
"   other bits for the other 2 pixels.
"   _____________________________
"   _   _   _   _   _ | _   _   _       5R-3G for high byte
"
"   _____________________________
"   _   _   _ | _   _   _   _   _       3G-5B for low byte
"
"              |
"              |                        ...THIS GIVES...
"             \_/
"   _   _   _   _   _   _   _   _
"   _   _   _ | _   _ | _   _   _
"
"   Lcd_db[7..5] =  Cam_db[5..3] for high byte and red pixel, OR
"                   Cam_db[2..0] for low byte and green pixel.
"
"   Lcd_db[4..3] =  Cam_db[2..1] for high byte and red pixel, OR
"                   Cam_db[5..4] for low byte and blue pixel.
"
"   Lcd_db[2..0] =  Cam_db[5..3] for high byte and green pixel, OR
"                   Cam_db[3..1] for low byte and blue pixel.
"
Lcd_db[7..5] = (Cam_db[5..3] & !Lcd_l_byte & RedPx) #
               (Cam_db[2..0] &  Lcd_l_byte & GreenPx);

Lcd_db[4..3] = (Cam_db[2..1] & !Lcd_l_byte & RedPx) #
               (Cam_db[5..4] &  Lcd_l_byte & BluePx);

Lcd_db[2..0] = (Cam_db[5..3] & !Lcd_l_byte & GreenPx) #
               (Cam_db[3..1] &  Lcd_l_byte & BluePx);

"WHEN (RowCntr == 2) THEN
"   Lcd_db = 0;
"ELSE
"   Lcd_db = 255;




" Image Capture and Display State Machine
STATE_DIAGRAM   IMAGEBITS

STATE   Rand0:
        GOTO    Idle;

STATE   Rand1:
        GOTO    Idle;

STATE   Rand2:
        GOTO    Idle;
```

```
STATE   Rand3:
        GOTO    Idle;

STATE   Rand4:
        GOTO    Idle;

STATE   Idle:
         IF (Lcd_init)  THEN    Idle
    ELSE IF (Get_img)   THEN    WaitForFrame
    ELSE                        Idle;

STATE   WaitForFrame:
         IF (Lcd_init)  THEN    Idle
    ELSE IF (NewFrame)  THEN    WaitForData;
    ELSE                        WaitForFrame;

STATE   WaitForData:
          IF (Lcd_init)  THEN   Idle
    ELSE  IF (ValidData) THEN   OutH_Wr0
    ELSE                        WaitForData;

STATE   OutH_Wr0:
    IF (Lcd_init)       THEN    Idle
    ELSE                        OutH_Wr00;

STATE   OutH_Wr00:
    IF (Lcd_init)       THEN    Idle
    ELSE                        OutH_Wr1;

STATE   OutH_Wr1:
    IF (Lcd_init)       THEN    Idle
    ELSE                        OutH_Wr11;

STATE   OutH_Wr11:
    IF (Lcd_init)       THEN    Idle
    ELSE                        OutL_Wr0;

STATE   OutL_Wr0:
    IF (Lcd_init)       THEN    Idle
    ELSE                        OutL_Wr00;

STATE   OutL_Wr00:
    IF (Lcd_init)       THEN    Idle
    ELSE                        OutL_Wr1;

STATE   OutL_Wr1:
    IF (Lcd_init)       THEN    Idle
    ELSE                        OutL_Wr11;

STATE   OutL_Wr11:
          IF (Lcd_init)             THEN    Idle
    ELSE  IF (ColEq320 & RowEq240)  THEN    Idle
    ELSE  IF (ValidData)            THEN    OutH_Wr0
```

```
    ELSE                                        WaitForData;




END  sdicintf
```

```c
/*
 * ---------------------------------------------------------------------------
 * -----                           ADC.C                                -----
 * -----                           EE90                                 -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is a library of functions for interfacing with the PIC18F4520's ADC
 *  interface
 *
 * Table of Contents:
 *  adc_init        - initialize the ADC channel
 *  adc_read        - returns 10bit number indicative of the input analog level
 *
 * Assumptions:
 *  The code assumes the actual hardware is hooked up as follows:
 *  The input analog signal is connected to ADC_CHANNEL which is 0 to 3 only!
 *  The input analog signal is connected at PORTA.
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC18 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Apr. 28, 2010       Nnoduka Eruchalu        Initial Revision
 *  Feb. 15  2011       Nnoduka Eruchalu        Updated to use any specified
 *                                              channel (an argument)
 */

#include <htc.h>
#include "adc.h"



/*
 * adc_init
 * Description:
 *  This initializes the ADC interface. This must be called before the ADC
 *  interface can be used.
 *  Thus this function has to be called before calling adc_read()
 *
 * Operation:
 *  Really just follows standard ADC initialization sequence.
 *  Check the datasheet.
 *
 * Revision History:
 *  Apr. 28, 2010       Nnoduka Eruchalu        Initial Revision
 *  Feb. 15, 2010       Nnoduka Eruchalu        updated to use all adc channels
 *                                              PORTA 0..3 as fixed adc inputs
 *
 */
void adc_init(void)
{
    TRISA |= (0x0F);                 /* make all used ADC channel bits INPUTs */

    /* CONFIGURE THE A/D MODULE */
```

```
    /* configure analog pins, voltage reference, and digital I/O (ADCON1)
     * AN3:0- analog, AN12:4- digital, Vref- = Vss, Vref+ = Vdd */
    ADCON1 = 0b00001011;

    ADCON0 = (ADC_CHANNEL << 2);  /* Select A/D input channel (ADCON0) */

    /* Select A/D acquisition time (ADCON2)
     * Select A/D conversion clock (ADCON2)
     * A/D Result Format =  Right Justified
     * Acquisition Time = 4T(AD), Conversion clock = Fosc/32 */
    ADCON2 = 0b10010010;


    ADON = 1;       /* Turn on A/D Module (ADCON0) */
    ADIE = 0;       /* Not interrupt driven */
    ADIF = 0;       /* Reset the ADC interrupt bit */
    ADRESL  =   0;  /* Reset the ADRES value register */
    ADRESH  =   0;
}



/*
 * adc_read
 * Description:
 *  Read the analog input on adc channel "chan" and return the converted 10 bit
 *  level indictor
 *
 * Operation:
 *  Really just follows standard ADC reading sequence. Check the datasheet.
 *
 * Arguments:
 *  chan - ADC input channel which has to be on PORTA and should be 0 to 3
 *
 * Revision History:
 *  Apr. 28, 2010       Nnoduka Eruchalu        Initial Revision
 *  Feb. 15  2011       Nnoduka Eruchalu        Updated to use any specified
 *                                              channel (an argument)
 */
unsigned int adc_read(unsigned char chan)
{
  ADCON0 &= 0b11000011;                         /* clear old A/D input channel */
  ADCON0 |= (chan << 2);                       /* Select new A/D input channel */

  CLRWDT();          /* Clear the Watch Dog to allow time to convert b4 reset */

  GODONE = 1;                                      /* Start the ADC conversion */

  while(GODONE) continue;                     /* wait for conversion to finish */

  return (ADRES);                                 /* return the converted value */
}
```

```c
/*
 * ---------------------------------------------------------------------------
 * -----                            ADC.H                               -----
 * -----                            EE90                                -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is the header file for adc.c, the library of functions for interfacing
 *  the PIC18F4520's ADC interface.
 *
 * Assumptions:
 *  The code assumes the actual hardware is hooked up as follows:
 *  The input analog signal is connected to ADC_CHANNEL which is 0 to 3 only!
 *  The input analog signal is connected at PORTA.
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC18 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Apr. 28, 2010       Nnoduka Eruchalu        Initial Revision
 *  Feb. 15  2011       Nnoduka Eruchalu        Updated to use any specified
 *                                              channel (an argument)
 */

#include <htc.h>

#ifndef _ADC_H
#define _ADC_H

/*HARDWARE CONNECTIONS*/
#define ADC_CHANNEL 0       /* this has to be on PORTA and should be 0 to 3 */


/*FUNCTION PROTOTYPES*/
/* initialize the ADC interface to some documented specs */
extern void adc_init(void);

/* read the digital conversion of the Analog input on chan */
extern unsigned int adc_read(unsigned char chan);

#endif
```

```
/*
 * ---------------------------------------------------------------------------
 * -----                          EEPROM.C                              -----
 * -----                           EE91b                                -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is a library of functions for interfacing with the PIC18F4520's EEPROM
 *
 * Table of Contents:
 *  eeprom_wr - write a byte of data to the eeprom
 *  eeprom_rd - read a byte of data from the eeprom
 *
 * Assumptions:
 *  None
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC28 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb. 27, 2010       Nnoduka Eruchalu        Initial Revision
 */

#include <htc.h>
#include "eeprom.h"

/*
 * eeprom_wr
 * Description:  Write a byte of data to the eeprom
 *
 * Operation:    Following sequence given in datasheet
 *               EEADR = address to write to; EEDATA = data to write;
 *               EEPGD = 0; CFGS = 0; WREN = 1;
 *               GIE = 0; EECON2 = 0x55; EECON2 = 0xAA; WR = 1; GIE = 1;
 *               WREN = 0;
 *
 * Arguments:    addr - address to write to
 *               data - data to write to
 *
 * Return:       None
 *
 * Revision History:
 *  Feb. 27, 2010       Nnoduka Eruchalu        Initial Revision
 */
void eeprom_wr(unsigned char addr, unsigned char data)
{
  EEADR = addr;                              /* data memory address to write */
  EEDATA = data;                               /* data memory value to write */

  EEPGD = 0;                                         /* point to data memory */
  CFGS = 0;                                                  /* acess eeprom */
  WREN = 1;                                                 /* enable writes */

  GIE = 0;                                             /* disable interrupts */
```

```
  EECON2 = 0x55;                                          /* Required Sequence */
  EECON2 = 0xAA;
  WR = 1;                                      /* set write bit to begin write */

  while (WR) continue;                           /* wait for write to complete */
  EEIF = 0;                                /* EEIF must be cleared by software */
  WREN = 0;                     /* Disable writes on write complete (EEIF clr) */
  GIE = 1;                                                /* Enable interrupts */
}



/*
 * eeprom_rd
 * Description:  Read a byte of data to the eeprom
 *
 * Operation:    Following sequence given in datasheet
 *               EEADR = address to read from;
 *               EEPGD = 0; CFGS = 0; RD = 1; data = EEDATA; return data
 *
 * Arguments:    addr - address to read from
 *
 * Return:       data- read from eeprom
 *
 * Revision History:
 *  Feb. 27, 2010       Nnoduka Eruchalu        Initial Revision
 */
unsigned char eeprom_rd(unsigned char addr)
{
  unsigned char data;
  GIE = 0;                                               /* disable interrupts */
  EEADR = addr;                                 /* data memory address to read */
  EEPGD = 0;                                           /* point to data memory */
  CFGS = 0;                                                   /* access EEPROM */
  RD = 1;                                                       /* EEPROM read */
  data = EEDATA;                                          /* get the read data */
  GIE = 1;                                                /* enable interrupts */
  return data;
}
```

```
/*
 * ---------------------------------------------------------------------------
 * -----                          EEPROM.H                               -----
 * -----                           EE91b                                 -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is the header file for eeprom.c, library of functions for interfacing
 *  with the PIC18F4520's EEPROM
 *
 * Table of Contents:
 *  function prototypes
 *
 * Assumptions:
 *  None
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC28 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb. 27, 2010       Nnoduka Eruchalu        Initial Revision
 */

#ifndef _EEPROM_H
#define _EEPROM_H

/* FUNCTION PROTOTYPES */
extern void eeprom_wr(unsigned char addr, unsigned char data);
extern unsigned char eeprom_rd(unsigned char addr);

#endif                                                            /* _EEPROM_H */
```

```c
/*
 * ---------------------------------------------------------------------------
 * -----                          I2C.C                                  -----
 * -----                          EE91                                   -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is a library of functions for interfacing with the PIC18F4520's I2C
 *  module.
 *  Output Only in this version
 *
 * Table of Contents:
 *
 *  void i2c_start(void)          - assert Start condition
 *  void i2c_stop(void)           - assert Stop condition
 *  void i2c_clock(void)          - pulse SCLx, (high then low)
 *  void i2c_sendbyte(char byte)  - send byte
 *  char i2c_getack(void)         - get acknowledge from receiver
 *  void i2c_init(void)           - initialize i2c module
 *  char i2c_WriteCam(char reg, int value)  - write to image sensor MT9V011
 *  void cam_init(void)           - initialize the MT9V011
 *
 * Assumptions:
 *  Assuming a standard I2C bus with just two wires SCL and SDA.
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC18 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */

#include <htc.h>
#include "i2c.h"



/*
 * i2c_start
 * Description:  After a bus idle state, a high-to-low transition of the SDAx
 *               line while the SCLx clock line is high determines a Start
 *               condition.
 *
 * Operation:    We want the SDA line to have a high-to-low transition while
 *               SCL line is high. So to ensure that there isn't a scenario
 *               of us setting SDA high when it was previously low during
 *               a SCL high period (thereby creating a stop condition!) start
 *               off by ensuring the clock is low. Then float the data high.
 *               The next step is to now float the SCL high before
 *               finishing the high-to-low transition of the SDA by now setting
 *               SDA low.
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 *
```

```
 */
void i2c_start(void)
{
  SCL_LOW();                                       /* FIRST, ensure clock is low */
  SDA_HIGH();                                             /* ensure data is high */
  __delay_us(I2C_TM_SCL_LOW);

  SCL_HIGH();                                      /* now send clock pulse high */
  __delay_us(I2C_TM_START_SU);          /* for required transition setup time */

  SDA_LOW();                                /* finally the high->low transition */
  __delay_us(I2C_TM_START_HD);
  return;
}



/*
 * i2c_stop
 * Description:  A low-to-high transition of the SDAx line while the clock
 *              (SCLx) is high determines a Stop condition. All data transfers
 *               end with a Stop condition.
 *
 * Operation:    We want the SDA line to have a low-to-high transition while
 *               SCL line is high. So first of all send the data low then
 *               float the SCL line high (can't assume it was already high on
 *               function call). After this finish the low->high transition by
 *               floating the data high.
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */
void i2c_stop(void)
{
  SCL_LOW();                                 /* only change SDA when SCL is low */
  SDA_LOW();                                        /* ensure data is low first */
  __delay_us(I2C_TM_SCL_LOW);

  SCL_HIGH();              /* now float SCL high for required Stop transition */
  __delay_us(I2C_TM_STOP_SU);                                    /*  setup time */

  SDA_HIGH();                               /* finally the low->high transition */
  __delay_us(I2C_TM_BUS_FREE);             /* bus free time before next start */
  return;
}



/*
 * i2c_clock
 * Description:  A clock pulse of SCL floating high then being driven low.
 *               Image below:     ___
 *                            ___|   |___
 *               Used when writing bytes.
 *
 * Operation:    We want the SDA line to have a low-to-high transition while
```

```
 *               SCL line is high. So first of all send the data low then
 *               float the SCL line high (can't assume it was already high on
 *               function call). After this finish the low->high transition by
 *               floating the data high.
 *
 * Assumptions:  When called, the Clock is already in a low state.
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */
void i2c_clock(void)
{
  __delay_us(I2C_TM_SCL_LOW); /* minimum Clock Low Time after data is written */
  SCL_HIGH();                                             /* float clock high */
  __delay_us(I2C_TM_SCL_HIGH);                     /* minimum clock high time */
  SCL_LOW();                              /* drive the clock low for minimum time */
  __delay_us(I2C_TM_SCL_LOW);                       /* before data is written */
  return;
}



/*
 * i2c_sendbyte
 * Description: Data is transferred in sequences of 8 bits. The bits are placed
 *              on the SDA line starting with the MSB. The SCL line is then
 *              pulsed high, then low.
 *
 * Operation:   Loop through all the bits of the byte argument, starting from
 *              bit 7 (MSB), and if a bit is 1 then float the SDA high, else
 *              drive the SDA low. Then after this pulse the clock.
 *
 * Arguments:   byte - byte value to send.
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */

void i2c_sendbyte(unsigned char byte)
{
  signed char i;
  SCL_LOW();                          /* each loop iteration assumes SCL has been */
  __delay_us(I2C_TM_SCL_LOW);                      /* low for the required time */

  for(i=7; i>=0; i--)
  {
    if((byte>>i) & 0x01)                  /* if bit is 1, then float data pin */
    {
      SDA_HIGH();
    }
    else                                  /* else drive data pin low          */
    {
      SDA_LOW();
    }
```

```
    i2c_clock();                                              /* pulse the clock */
  }
}

/* i2c_getack
 * Description: All data byte transmissions must be acknowledged or
 *              not acknowledged by the receiver. The receiver will pull the
 *              SDAx line low for an acknowledge or release the SDAx line for a
 *              "not acknowledge". THe acknowledge is a one-bit period using the
 *              SCLx clock.
 *
 * Operation:   drive the clock low, then make data line an input to listen for
 *              ack. Then float the clock high and then you can now extract the
 *              ACK\ or NACK info.
 *
 * Return:      1 if ACK\   or  0 if NACK
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */
unsigned char i2c_getack(void)
{
  unsigned char ack;

  SCL_LOW();                                                /* drive the clock low */
  SDA_TRIS = I2C_IN;                     /* make SDA an input so can read for ack */
  __delay_us(I2C_TM_SCL_TO_DATA);                           /* ack data setup time */

  SCL_HIGH();                                              /* float the clock high */
  __delay_us(I2C_TM_DATA_SU);

  ack = SDA_in;                                      /* read ack while still valid */
  __delay_us(I2C_TM_SCL_HIGH);  /* but ensure clock is high for required time */

  SCL_LOW();                     /* do not leave this routine till SDA line is */
  __delay_us(I2C_TM_SCL_TO_FREE);                              /* freed by receiver */

  return (!ack);                    /* remember ACK\ is active low, and I like */
                                                  /* active high return values */
}


/* i2c_init
 * Description: Initialize the PIC24F to use the I2C master module.
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */
void i2c_init(void)
{
  SCL_TRIS = I2C_IN;                      /* the two wire interface should start */
  SDA_TRIS = I2C_IN;                      /* in a high impedance state           */
}
```

```c
/*
 * i2c_WriteCam
 * Description: This writes commands to setup the image sensor MT9V011.
 *
 * Operation:   The I2C interface is
 *              for writing to 16 bit registers. A start bit is given by the
 *              master, followed by the write addrss, starts the sequence. The
 *              image sensor will then give an acknowledge bit and expects the
 *              register address to come first, followed by the 16-bit data.
 *              After each 8-bit the image sensor will give an acknowledge bit.
 *              All 16 bits must be written before the register will be updated.
 *              After 16 bits are transferred, the register address is
 *              automatically incremented. So that the next 16 bits are written
 *              to the neext register. The master stops writing by sending a
 *              start or stop bits.
 *
 * Arguments:   reg   - register to write to
 *              val   - value to write to the specified register.
 *
 * Returns:     0 for fail, 1 for success
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */
char i2c_WriteCam(char reg, int val)
{
  char byte;                                                /* temp storage */

  i2c_start();                                            /* send Start bit */

  i2c_sendbyte(0xBA);        /* send the write address, note LSB==0 ==> WRITE */
  if(!i2c_getack())              /* if cant get acknowledge, then return fail */
  {
    i2c_stop();
    return(0);
  }

  i2c_sendbyte(reg);                              /* send the register number, */
  if(!i2c_getack())              /* if cant get acknowledge, then return fail */
  {
    i2c_stop();
    return(0);
  }

  byte = (val & 0xFF00) >> 8;            /* send the high byte of value first */
  i2c_sendbyte(byte);
  if(!i2c_getack())              /* if cant get acknowledge, then return fail */
  {
    i2c_stop();
    return(0);
  }

  byte = val & 0x00FF;                      /* send the low byte of value next */
```

```
  i2c_sendbyte(byte);
  if(!i2c_getack())              /* if cant get acknowledge, then return fail */
  {
    i2c_stop();
    return(0);
  }

  i2c_stop();                                 /* all went well, return success */
  return(1);
}



/* Description: This initializes the image sensor, the MT9V011 .
 *
 * Operation:   Write to the sensor's registers. Updating the window size and
 *              row and column start addresses.
 *
 * Arguments:   reg   - register to write to
 *              val   - value to write to the specified register.
 *
 * Returns:     0 for fail, 1 for success
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */
void cam_init(void)
{
  i2c_init();                           /* don't assume i2c port is ready to go */

  TRISB5 = 0;    /* camera reset */
  LATB5 = 0;
  __delay_ms(1);
  LATB5 = 1;

  i2c_WriteCam(0x0A, 0x0008);              /* data rate = 1/(2 + 8) of CLKIN */

  /* someday update the row start and column start */

  i2c_WriteCam(0x03, 0x00EF);             /* window height = 240 - 1 = 0x00EF */
  i2c_WriteCam(0x04, 0x013F);             /* window width  = 320 - 1 = 0x013F */
}
```

```
#ifndef I2C_H_
#define I2C_H_
/*
 * ---------------------------------------------------------------------------
 * -----                            I2C.H                                -----
 * -----                            EE91                                 -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is the header file for I2C.c, the library of functions for interfacing
 *  with the PIC18F4520's I2C module.
 *
 * Assumptions:
 * Assumes the image sensor MT9V011 has a master clock of
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC24 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Jan. 24, 2010       Nnoduka Eruchalu        Initial Revision
 */

#include "sdic.h"

/* ---------------------------- REQUIRED DEFINES -----------------------------*/
#define SCL_TRIS  TRISC3                                          /* I2C bus */
#define SCL       LATC3
#define SDA_TRIS  TRISC5
#define SDA       LATC5
#define SDA_in    RC5

#define I2C_LOW  0                                 /* Puts pin into low mode */
#define I2C_OUT  0                              /* Puts pin into output mode */
#define I2C_HIGH 1                                /* Puts pin into high mode */
#define I2C_IN   1                               /* Puts pin into input mode */


/* ----------------------------- USEFUL MACROS ----------------------------- */
#define SCL_HIGH()  SCL_TRIS = I2C_IN
#define SCL_LOW()   SCL = I2C_LOW;  SCL_TRIS = I2C_OUT
#define SDA_HIGH()  SDA_TRIS = I2C_IN
#define SDA_LOW()   SDA = I2C_LOW;  SDA_TRIS = I2C_OUT

/* ---------- TIMING FOR THE I2C BUS (nearest us) -------------------------- */
/* really just using numbers from the MT9V011 datasheet                      */

#define I2C_TM_START_SU     5   /* setup and hold times for the SDA high->low */
#define I2C_TM_START_HD     4  /* transition which indicate a Start condition */

#define I2C_TM_SCL_HIGH     4                         /* minimum clock high time */
#define I2C_TM_SCL_LOW      5                          /* minimum clock low time */

#define I2C_TM_STOP_SU      4   /* setup and hold times for the SDA low->high */
#define I2C_TM_BUS_FREE     5   /* transition of a Stop condition. hold time  */
```

```
                                /* is also the min. time b4 next write cycle  */

#define I2C_TM_SCL_TO_DATA  6     /* during ack listen: SCL low to data valid */
#define I2C_TM_DATA_SU      1     /* grab data at this time after SCL is high */
#define I2C_TM_SCL_TO_FREE  3    /* time for SDA line to be freed by receiver */



 /* ------------------------- FUNCTION PROTOTYPES ------------------------- */
extern void i2c_start(void);                       /* assert Start condition */
extern void i2c_stop(void);                         /* assert Stop condition */
extern void i2c_clock(void);                   /* pulse SCLx, (high then low) */
extern void i2c_sendbyte(char byte);                             /* send byte */
extern char i2c_getack(void);                /* get acknowledge from receiver */
extern void i2c_init(void);                         /* initialize i2c module */
extern char i2c_WriteCam(char reg, int value);/* write 2 image sensor MT9V011 */
extern void cam_init(void);                        /* initialize the MT9V011 */

#define i2c_restart()   i2c_start()

#endif                                                            /* I2C_H_ */
```

```c
/*
 * ---------------------------------------------------------------------------
 * -----                           LCD.C                                -----
 * -----                           EE91                                 -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is a library of functions for interfacing the PIC18F4520 with
 *  the Newhaven Display 2.4" TFT NHD-2.4-240320SF-CTXI# (uses an ILI9328
 *  controller).
 *
 * Table of Contents:
 *  lcd_data_out - outputs 16-bit data using an 8-bit interface
 *  lcd_comm_out - outputs to the registers which have byte indices
 *  FullDisplay  - fill the display with a single color. (Background/Testing?)
 *  lcd_delay    - delay for the lcd routines
 *  lcd_init     - initialize the TFT LCD module
 *
 * Assumptions:
 *  Assumes the HARDWARE CONNECTIONS in lcd.h
 *  Assumes LCD's databus and control bits are always outputs
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC28 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */

#include <htc.h>
#include "lcd.h"

/*
 * lcd_data_out
 * Description:  Write 16 bits of data to 2.4" TFT using an 8-bit interface
 *
 * Operation:    Set RS, Set RD and the write the data bytes (high then low).
 *               Observe the strobing of the WR\ line during writes.
 *
 * Arguments:    h - high byte to be written
 *               l - low byte to be written
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */
void lcd_data_out(unsigned char h, unsigned char l)
{
  SET_RS();
  SET_RD();


  CLEAR_WR();
  LCD_DATA = h;
  SET_WR();
```

```
  CLEAR_WR();
  LCD_DATA = l;
  SET_WR();
}




/*
 * lcd_comm_out
 * Description:  Write a 1 byte command to 2.4" TFT
 *
 * Operation:    Clear RS, Set RD and the write the data byte.
 *               Remember to send a high byte of 0x00, followed by the
 *               actual data byte. Also observe the strobing of the WR\
 *               line during writes.
 *
 * Arguments:    c - command byte to be written
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */
void lcd_comm_out(unsigned char c)
{
  CLEAR_RS();
  SET_RD();

  CLEAR_WR();
  LCD_DATA = 0x00;
  SET_WR();

  CLEAR_WR();
  LCD_DATA = c;
  SET_WR();
}



/*
 * FullDisplay
 * Description:  Fill the TFT LCD with just 1 color. Creates a background
 *               and good for testing
 *
 * Operation:    Loop through the 320 rows and 240 columns of the LCD and keep
 *               on doing data_outs
 *
 * Arguments:    h - high byte of fill color
 *               l - low byte of fill color
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */
void FullDisplay(unsigned char h, unsigned char l)
{
  unsigned int i, j;
```

```
  SET_RS();
  SET_RD();
  LCD_DATA = l;
  for (i=0; i<320; i++)
    {
      for (j=0; j<240; j++)
        {
          CLEAR_WR();
          LCD_DATA = h;
          SET_WR();

          CLEAR_WR();
          LCD_DATA = l;
          SET_WR();
        }
    }
}



/*
 * TestDisplay
 * Description:  Test the LCD by writing a simple pattern to the LCD
 *
 * Operation:    Loop through the 320 rows and 240 columns of the LCD and keep
 *               on doing data_outs
 *
 * Arguments:    None
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */
void TestDisplay(void)
{
  unsigned int i, j;
  unsigned char h, l;
  SET_RS();
  SET_RD();
  LCD_DATA = l;
  for (i=1; i<=240; i++)
    {
      for (j=1; j<=320; j++)
        {
          if (i == 2)
            {
              h = 0xF8;
              l = 0;
            }
          else
            {
              h = l = 0xFF;
            }
          CLEAR_WR();
          LCD_DATA = h;
          SET_WR();
```

```
          CLEAR_WR();
          LCD_DATA = l;
          SET_WR();
        }
    }
}




/*
 * lcd_delay
 * Description:  delay for the LCD routines.
 *
 * Operation:    just a for-loop on __delay_ms()
 *
 * Arguments:    n - used for scaling the delay
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */
void lcd_delay(unsigned int n)
{
  unsigned int i, j;
  for (i=0; i<n; i++)
    {
      __delay_ms(10);
    }
}


/*
 * lcd_init
 * Description:  Initialize the LCD for use
 *
 * Operation:    Setup the LCD pins as outputs from the PIC18.
 *               Reset the LCD, go into 8-bit data write mode, and
 *               write appropriate commands.
 *               It is important to leave this function with the system
 *               in 16-bit data mode.
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */
void lcd_init(void)
{
  LCD_DATA_TRIS = 0x00;
  LCD_RS_TRIS   = 0;
  LCD_RD_TRIS   = 0;
  LCD_WR_TRIS   = 0;
  LCD_MODE_TRIS = 0;
  LCD_RST_TRIS  = 0;
  LCD_MODE = MODE_INIT;                                /* start in init mode */
```

```
    CLEAR_RST();                                                /* reset the LCD */
    lcd_delay(100);
    SET_RST();
    lcd_delay(100);

    lcd_comm_out(0xE5); lcd_data_out(0x80,0x00);
    lcd_comm_out(0x00); lcd_data_out(0x00,0x01);
    lcd_comm_out(0x01); lcd_data_out(0x01,0x00);
    lcd_comm_out(0x02); lcd_data_out(0x07,0x00);
    lcd_comm_out(0x03); lcd_data_out(0x10,0x88);
    lcd_comm_out(0x04); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x08); lcd_data_out(0x02,0x02);
    lcd_comm_out(0x09); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x0A); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x0C); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x0D); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x0F); lcd_data_out(0x00,0x00);

    lcd_comm_out(0x10); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x11); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x12); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x13); lcd_data_out(0x00,0x00);
    lcd_delay(200);
    lcd_comm_out(0x10); lcd_data_out(0x17,0xB0);
    lcd_comm_out(0x11); lcd_data_out(0x01,0x37);
    lcd_delay(50);
    lcd_comm_out(0x12); lcd_data_out(0x01,0x3B);
    lcd_delay(50);
    lcd_comm_out(0x13); lcd_data_out(0x19,0x00);
    lcd_comm_out(0x29); lcd_data_out(0x00,0x07);
    lcd_comm_out(0x2B); lcd_data_out(0x00,0x20);
    lcd_delay(50);
    lcd_comm_out(0x20); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x21); lcd_data_out(0x00,0x00);

    lcd_comm_out(0x30); lcd_data_out(0x00,0x07);
    lcd_comm_out(0x31); lcd_data_out(0x05,0x04);
    lcd_comm_out(0x32); lcd_data_out(0x07,0x03);
    lcd_comm_out(0x35); lcd_data_out(0x00,0x02);
    lcd_comm_out(0x36); lcd_data_out(0x07,0x07);
    lcd_comm_out(0x37); lcd_data_out(0x04,0x06);
    lcd_comm_out(0x38); lcd_data_out(0x00,0x06);
    lcd_comm_out(0x39); lcd_data_out(0x04,0x04);
    lcd_comm_out(0x3C); lcd_data_out(0x07,0x00);
    lcd_comm_out(0x3D); lcd_data_out(0x0A,0x08);

    lcd_comm_out(0x50); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x51); lcd_data_out(0x00,0xEF);
    lcd_comm_out(0x52); lcd_data_out(0x00,0x00);
    lcd_comm_out(0x53); lcd_data_out(0x01,0x3F);
    lcd_comm_out(0x60); lcd_data_out(0x27,0x00);
    lcd_comm_out(0x61); lcd_data_out(0x00,0x01);
    lcd_comm_out(0x6A); lcd_data_out(0x00,0x00);
```

```
  lcd_comm_out(0x90); lcd_data_out(0x00,0x10);
  lcd_comm_out(0x92); lcd_data_out(0x00,0x00);
  lcd_comm_out(0x95); lcd_data_out(0x01,0x10);
  lcd_comm_out(0x97); lcd_data_out(0x00,0x00);
  lcd_comm_out(0x07); lcd_data_out(0x01,0x73);
  lcd_delay(10);
  lcd_comm_out(0x22);


  TestDisplay();                                          /* Test the LCD */

  SET_RS();                                   /* now setup for writes only */
  SET_RD();

  LCD_MODE = MODE_READY;          /* set into ready to use mode before leaving */
}
```

```
/*
 * ---------------------------------------------------------------------------
 * -----                            LCD.H                               -----
 * -----                            EE91                                -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is the header file for lcd.c, the library of functions for interfacing
 *  the PIC18F4520 with the Newhaven Display 2.4" TFT NHD-2.4-240320SF-CTXI#
 *  (uses an ILI9328 controller).
 *
 * Table of Contents:
 *
 *
 * Assumptions:
 *  Assumes the HARDWARE CONNECTIONS below
 *  Assumes LCD's databus and control bits are always outputs
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC28 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb. 4, 2010       Nnoduka Eruchalu        Initial Revision
 */

#ifndef _LCD_H
#define _LCD_H

#include "sdic.h"

/* HARDWARE CONNECTIONS */
#define LCD_DATA         LATD             /* use LAT for writing to i/o ports */
#define LCD_RS           LATE0
#define LCD_RD           LATE1
#define LCD_WR           LATE2
#define LCD_MODE         LATC0
#define LCD_RST          LATC6

#define LCD_DATA_TRIS    TRISD          /* set a bit to 1 to make it an input */
#define LCD_RS_TRIS      TRISE0        /* set a bit to 0 to make it an output */
#define LCD_RD_TRIS      TRISE1
#define LCD_WR_TRIS      TRISE2
#define LCD_MODE_TRIS    TRISC0
#define LCD_RST_TRIS     TRISC6

#define MODE_INIT        1                  /* values for the LCD_MODE output */
#define MODE_READY       0



/* LCD DATBUS MANIPULATION */
#define write_data ()

/* LCD CONTROL BIT MANIPULATIONS */
/* Assumes LCD control bits are always outputs */
```

```c
#define SET_RS()      (LCD_RS = 1)
#define SET_RD()      (LCD_RD = 1)
#define SET_WR()      (LCD_WR = 1)
#define SET_RST()     (LCD_RST = 1)

#define CLEAR_RS()    (LCD_RS = 0)
#define CLEAR_RD()    (LCD_RD = 0)
#define CLEAR_WR()    (LCD_WR = 0)
#define CLEAR_RST()   (LCD_RST = 0)

/* remember WR\ is active low, so its default is high */
#define LCD_STROBE_WR()  CLEAR_WR(); __delay_us(1); SET_WR(); __delay_us(1)


/* FUNCTION PROTOTYPES */
extern void lcd_data_out(unsigned char h, unsigned char l);
extern void lcd_comm_out(unsigned char c);
extern void FullDisplay(unsigned char h, unsigned char l);
extern void lcd_delay(unsigned int n);
extern void lcd_init(void);
extern void TestDisplay(void);

#endif                                                               /* _LCD_H */
```

```c
/* ---------------------------------------------------------------------------
 * -----                           MOTOR.C                               -----
 * -----                            EE91b                                -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is a library of functions for interfacing with any Bipolar Stepper
 *  Motor.
 *  All that has to be done is declare what line, each phase A,A',B,B' is
 *  connected to in the header file motor.h.
 *
 * Table of Contents:
 *  motor_write     - write values to the motor phases.
 *  setRelDirection - turn the motor by a fixed number of degrees
 *  motor_init        - initialize the motor
 *
 * Assumptions:
 *  The code assumes the actual hardware is hooked up as described in the
 *  include file, motor.h
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC18 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb. 26, 2011       Nnoduka Eruchalu        Initial Revision
 */

#include <htc.h>
#include "motor.h"

char stepTblIdx;

/*
 * StepTable
 *
 * Description: This is the step table for full steps.
 *              Full Step Sequence is AB --> A'B --> A'B' --> AB'--> loop again
 *
 * Revision History:
 *  Feb. 26, 2011       Nnoduka Eruchalu        Initial Revision
 */
char StepTable[NUM_STEPS] = {
  /*   xxxx B' A' B A*/
  0x03, /* 0000 0011 */
  0x06, /* 0000 0110 */
  0x0C, /* 0000 1100 */
  0x09  /* 0000 1001 */
};


/*
 * motor_init
 * Description:  Initialize the motor for use
 *
```

```
 * Operation:    Setup the motor pins as outputs from the PIC18.
 *               Set the motor direction as clockwise, the step table index to
 *               0 (should start from beginning of a table).
 *
 * Revision History:
 *  Feb. 26, 2010       Nnoduka Eruchalu        Initial Revision
 */
void motor_init(void)
{
  /* set all motor phase ports to outputs, by setting the TRIS to 0 */
  MOTOR_TRIS &= ~MOTOR_MASK;
  stepTblIdx = 0;

  return;
}



/*
 * motor_write
 * Description:  write a given value to the motor phase lines
 *
 * Operation:    read in the current value at the motor port, save it in a temp
 *               variable, update only the bits of interest, then write the
 *               updated port value back to the motor port.
 *
 * Assumptions:  Assumes input value is in format (xxxx B' A' B A)
 *
 * Revision History:
 *  Feb. 26, 2010       Nnoduka Eruchalu        Initial Revision
 */
void motor_write(char value)
{
  char val;
  char reg;
  reg = MOTOR_PORT;
  reg &= ~MOTOR_MASK;  /* clear motor mask */
  reg |= ((value << MOTOR_BITs_OFFSET) & MOTOR_MASK);  /* write value */
  MOTOR_LAT = reg;
}



/*
 * setRelDirection
 * Description:  Move the motor in the decided direction.
 *
 * Operation:    Figure out the number of steps required for such a move.
 *               For those number of steps, loop through the step table and
 *               keep on outputting the table contents to the motor.
 *               Note there is a delay after each step... this is essential
 *               or the stepper motor wont work. Write now it is configured
 *               for a frequency of 50KHz... Yes a magic number I know...
 *
 * Revision History:
 *  Feb. 26, 2010       Nnoduka Eruchalu        Initial Revision
```

```
 */
void setRelDirection(signed int relAngle)
{
  char i;
  signed int stepCnt = relAngle/1.8;

  while(stepCnt)
    {
      motor_write(StepTable[stepTblIdx]); /* write value at current index */
      if (stepCnt > 0)
        {
          stepTblIdx++;
          stepCnt--;
        }
      else
        {
          stepTblIdx--;
          stepCnt++;
        }

      /* wrap around table, using fact MODULO 2^N = AND (2^N - 1) */
      stepTblIdx &= (NUM_STEPS - 1);


      __delay_ms(20); /* step frequency = 50 Hz */
    }
}
```

```c
/* --------------------------------------------------------------------------
 * -----                          MOTOR.H                              -----
 * -----                           EE91b                               -----
 * --------------------------------------------------------------------------
 *
 * File Description:
 *  This is the header file for motor.c, thelibrary of functions for interfacing
 *  with any Bipolar Stepper Motor.
 *  All that has to be done is declare what line, each phase A,A',B,B' is
 *  connected to.
 *
 * Table of Contents:
 *  motor_write     - write values to the motor phases.
 *  SetRelDirection - turn the motor by a fixed number of degrees
 *  motor_init        - initialize the motor
 *
 * Assumptions:
 *  The code assumes the actual hardware is hooked up as described in the
 *  HARDWARE CONNECTIONS section
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC18 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb. 26, 2011       Nnoduka Eruchalu        Initial Revision
 */

#ifndef  _MOTOR_H
#define  _MOTOR_H
#include "sdic.h"

/* HARDWARE CONNECTIONS */
#define MOTOR_LAT       LATB              /* use LAT for writing to i/o ports */
#define MOTOR_PORT      PORTB

#define MOTOR_TRIS     TRISB            /* set a bit to 1 to make it an input */
                                       /* set a bit to 0 to make it an output */

/* assuming your ports are all arranged sequentially, what is the lowest bit */
/* number... assumed format is  B' A' B A */
#define MOTOR_BITs_OFFSET   1

/* good constants to have */
#define NUM_STEPS       4              /* This must be  2^N, think about it...*/
#define MOTOR_MASK      0x01E       /* mask to select motor bits of MOTOR_LAT */

/* MOTOR BIT MANIPULATIONS */
#define CLEAR_BIT(reg, bit)   (reg &= (~(1 << bit)))

/* set bit to value given in val, 0 or 1 */
#define SET_BITval(reg, bit, val)  CLEAR_BIT(reg,bit); (lat |= (val << bit))

/* FUNCTION PROTOTYPES */
extern void motor_init(void);
```

```
extern void motor_write(char value);
extern void setRelDirection(signed int relAngle);


#endif                                                          /* _MOTOR_H */
```

```
/*
 * ---------------------------------------------------------------------------
 * -----                         SDIC.C                                  -----
 * -----                          EE91b                                  -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is the main file for the EE91 Sound Detect and Image Capture Project.
 *  (SDIC).
 *  The theory of operation behind this is rather simple. Detect peak sound,
 *  Take a picture and show it on the LCD.
 *
 * Assumptions:
 *  None
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC18 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb. 3, 2010       Nnoduka Eruchalu        Initial Revision
 */

#include <htc.h>
#include <stdio.h>
#include "sdic.h"
#include "lcd.h"
#include "i2c.h"
#include "adc.h"
#include "motor.h"
#include "eeprom.h"


/* chip settings
 *  for more info on these, look at section 23.1 (Configuration Bits) of the
 *  PIC184520 datasheet.
 *  As of today, this datasheet can be found here:
 *      http://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf
 */

__CONFIG(1,0x0F00);                              /* Using EC clock input mode */
__CONFIG(2,0X1E1F);
__CONFIG(3,0X8100);           /* CCP2 input/output is multiplexed with RC1    */
__CONFIG(4,0X00C1);
__CONFIG(5,0XC00F);


/* EEPROM data */
__EEPROM_DATA(0,1,0,0,0,0,0,0);



/* global variables -- sucks, but has to be done */
volatile char go;             /* active high signal initializes sound seeking */
volatile char mode;                                /* start by taking video */
```

```
volatile char need_new_pic;                     /* need to take a new picture */

/*
 * sdic_init
 * Description:  initialization routine for SDIC
 *
 * Operation:    setup system level inputs and outputs. For the outputs
 *               setup the initial values.
 *               setup the interrupts for the switches/buttons
 *
 * Arguments:    None
 *
 * Return:       None
 *
 * Revision History:
 *  Feb. 27, 2010       Nnoduka Eruchalu        Initial Revision
 */
void sdic_init(void)
{
  GET_IMAGE_TRIS = 0;                            /* make get image pin an output */
  GET_IMAGE_LAT = 0;                      /* don't want to get image just yet... */

  /* setup interrupts for switches */
  GIEH = 1;                                                /* enable interrupts */
  GIEL = 1;

  /* GO Button -- uses INT0 */
  TRISB0 = 1;                                               /* always an input */
  INT0IF = 0;                                        /* Will be using INT0 for GO */
  INT0IE = 1;
  INTEDG0 = 0;                          /* a switch, so trigger on falling edge */


  /* Setup timers just to ensure that CCP pins work... */
  T3CON = 0b00000001;               /* Setup T3CON to choose Timer 1 for CCP2 */
  TMR3 = 0x0000;                            /* Load initial value into timer3 */
  TMR3ON = 1;

  T1CON = 0b00000001;               /* since using Timer 1, have to set ut up */
  TMR1 = 0;                                 /* Load initial value into timer1 */
  TMR1ON = 1;

  /* MODE button -- uses CCP2 on pin RC1 */
  TRISC1 = 1;                                               /* always an input */
  CCP2CON = 0x04;                         /* now actually initialize CCP2 */
  CCPR2 = 0;                                     /* to trigger on falling edge */
  CCP2IF = 0;                                      /* use 0x05 for rising edge */
  CCP2IE = 1;                                      /* use 0x04 for falling edge*/

}



/*volatile unsigned int temp = 0;         /* temporary storage variable       */
```

```
void main(void)
{
  /* important variables */
  volatile signed char dir;           /* direction of the motor 1->CW, -1->CCW*/
  volatile signed int pos;                  /* position relative to 0-degrees */
  volatile unsigned int mics_curr[4];          /* microphones' current values */
  volatile unsigned int mics_min[4];                          /* mics' mins  */
  volatile unsigned int mics_max[4];              /* microphones' max. values */
  volatile signed int mics_max_pos[4];             /* REL pos. of max capture */

  /* system parameters */
  volatile unsigned char reset;                    /* active low reset button */


  /* mic reading variables */
  volatile signed char mic_i;                             /* index for motors */
  volatile signed int mic_read_i;                /* index into averaging loop */
  volatile unsigned long res, temp, div;

  /* motor rotation variables */
  volatile unsigned char half_rots;
  volatile signed int rel_angle;


 Startup:
  /* first initialize variables */
  go = 0;                              /* don't go on system startup or reset */
  mode = VIDEO_MODE;                                  /* start by taking video */
  need_new_pic = 0;                    /* will not need to take a new picture */

  /* reset mic read and position arrays */
  for(mic_i = 0; mic_i < NUM_MICS; mic_i++)
    {
      mics_min[mic_i] = 1024;
      mics_max[mic_i] = 0;
      mics_max_pos[mic_i] = 0;
    }
  half_rots = 0;                         /* so far no half-rotation completed */
  reset = 1;                                         /* reset MUST be inactive */


  /* then call system init routines */
  sdic_init();                                        /* system initialization */
  motor_init();                                /* stepper motor initialization */
  lcd_init();                                            /* lcd initialization */
  cam_init();                             /* cmos image sensor initialization */
  adc_init();                                     /* PIC's ADC initialization */


  /* initializations done, so get image!... currently video mode */
  GET_IMAGE_LAT = 1;

  /* figure out last motor position as of last system shutdown */
  /* this is saved in the EEPROM. Note that if any of these values */
```

```
/* are greater than the max allowed in th SYSTEM CONSTANTS section of sdic.h
 * then it is a sign that the PIC just got reprogrammed and the EEPROM
 * has been reset to all 1s.
 */
pos = eeprom_rd(EEPROM_POS_LOC);
dir = eeprom_rd(EEPROM_DIR_LOC);

while(1)
  {
    /* The user gets to use the SYS_RESET button to set the 0 degrees       */
    /* orientation of the motor unit to where it is currently facing        */
    /* This SYS_RESET button shouldnt be made readily available to a user.  */
    /* Keep this inside the packaging!                                      */
    reset = SYS_RESET;                                     /* read reset */
    if (reset == 0)                          /* is active low reset active? */
      {
        reset = 1;                                  /* make reset inactive */
        pos = 0;                       /* current position becomes 0 degrees */
        dir = 1;                       /* and restart with CW motion         */
        eeprom_wr(EEPROM_POS_LOC, pos);             /* save current status */
        eeprom_wr(EEPROM_DIR_LOC, dir);
        goto Startup;
      }


    /* don't do anything till GO is allowed */
    /* while waiting be sure to be in the correct mode */
    while(go == 0)                   /* stay here and wait until GO is allowed */
      {

        /* first check for reset.... */
        reset = SYS_RESET;                                 /* read reset */
        if (reset == 0)                      /* is active low reset active? */
          {
            reset = 1;                              /* make reset inactive */
            pos = 0;                   /* current position becomes 0 degrees */
            dir = 1;                   /* and restart with CW motion         */
            eeprom_wr(EEPROM_POS_LOC, pos);         /* save current status */
            eeprom_wr(EEPROM_DIR_LOC, dir);
            goto Startup;
          }

        if (mode == VIDEO_MODE)
          {                     /* video mode is simply continuous streaming */
            GET_IMAGE_LAT = 1;
          }
        else if ((mode == PICTURE_MODE) && (need_new_pic == 1))
          {
            GET_IMAGE_LAT = 1;   /* picture mode is simply last video frame */
            __delay_ms(20);
            GET_IMAGE_LAT = 0;
            need_new_pic = 0;      /* don't want to keep on taking pictures */
          }                        /* coz  it becomes video... don't want that */
        else
```

```
            GET_IMAGE_LAT = 0;
        }

    /* GO is allowed, so start/continue the process */

    /* handle motor rotation */
    setRelDirection(dir * DEGS_PER_MOVE);         /* make move based on dir */
    pos += (dir * DEGS_PER_MOVE);                 /* and update position    */

    eeprom_wr(EEPROM_POS_LOC, pos);                  /* save current status */
    eeprom_wr(EEPROM_DIR_LOC, dir);



    if(pos >= MAX_POS)                              /* if hit max position, */
      {
        dir = -1;                                      /* reverse direction */
        half_rots++;                        /* and update half-rotations count. */
      }

    if(pos <= MIN_POS)                              /* if hit min position, */
      {
        dir = +1;                                      /* reverse direction */
        half_rots++;                        /* and update half-rotations count. */
      }



    /* wait for rotating board (with camera and mics) to stop vibrating */
    for(mic_read_i = 100; mic_read_i > 0; mic_read_i--)
      __delay_ms(20);


    /* now that motor has changed position, read in the mic values */
    res = 0;
    div = 0;

    for(mic_i = 0; mic_i < NUM_MICS; mic_i++)
      {
        /* for each microphone take a lot of reads (div reads to be exact)  */
        /* and average all the reads to get a result for mic_i (in res)     */
        /* averaging over 2000 reads takes 100ms                            */
        for( mic_read_i = 2000;  mic_read_i > 0;  mic_read_i--)
          {
            temp = adc_read(2);
            if(temp > 512)
              {
                res += temp;
                div++;
              }
          }
        res /= div;

        mics_curr[mic_i] = res;              /* save current read for mic_i */
```

```
        }


      /* now have current mic reads, so update the mins and maxs*/
      for(mic_i = 0; mic_i < NUM_MICS; mic_i++)
        {

          if (mics_curr[mic_i] < mics_min[mic_i]) /* if curr less than min,   */
            mics_min[mic_i] = mics_curr[mic_i];   /* curr becomes new min     */

          if (mics_curr[mic_i] > mics_max[mic_i]) /* if curr greater than max,*/
            {
              mics_max[mic_i] = mics_curr[mic_i];   /* curr becomes new max   */
              mics_max_pos[mic_i] = pos;
            }
        }
      if (half_rots >= 2)
        {
          /* go to where max sound was */
          rel_angle = mics_max_pos[MID_MIC] - pos;

          pos = mics_max_pos[MID_MIC];

          if (rel_angle > 0)
            dir = +1;

          if (rel_angle < 0)
            dir = -1;

          /* if rel_angle == 0, then dir is unchanged */

          setRelDirection(rel_angle);                /* get to new position */
          eeprom_wr(EEPROM_POS_LOC, pos);            /* save current status */
          eeprom_wr(EEPROM_DIR_LOC, dir);

          /* now that have found sound, reset all system parameters */
          go = 0;                /* user will have to press GO to repeat seek */
          for(mic_i = 0; mic_i < NUM_MICS; mic_i++)
            {
              mics_min[mic_i] = 1024;
              mics_max[mic_i] = 0;
              mics_max_pos[mic_i] = 0;
            }
          half_rots = 0;

        }
    }



  /* Loop Forever, shouldnt get here */
  while(1) continue;
}
```

```
/* interrupt service routines */
void interrupt isr(void)
{
  if((INT0IE)&&(INT0IF))                      /* interrupt from GO has occured */
    {
      go = 1;                                          /* start a sound search */
      INT0IF=0;                                              /* clear the flag */
    }



  if((CCP2IE)&&(CCP2IF))                    /* interrupt from MODE has occured */
    {
      mode ^= 1;                                            /* toggle the mode */
      need_new_pic = 1;                         /* obviously only for pict mode */
      CCP2IF=0;                                              /* Clear the flag */
    }


  if((CCP1IE)&&(CCP1IF))
    {
      /* do something */
      CCP1IF=0;                                              /* Clear the flag */
    }
}
```

```
#ifndef SDIC_H_
#define SDIC_H_

/*
 * ---------------------------------------------------------------------------
 * -----                          SDIC.H                                -----
 * -----                           EE91b                                -----
 * ---------------------------------------------------------------------------
 *
 * File Description:
 *  This is the header file for the EE91 Sound Detect and Image Capture Project.
 *  (SDIC).
 *  The theory of operation behind this is rather simple. Detect peak sound,
 *  Take a picture/video and show it on the LCD.
 *
 * Assumptions:
 *  None
 *
 * Compiler:
 *  HI-TECH C Compiler for PIC18 MCUs (http://www.htsoft.com/)
 *
 * Revision History:
 *  Feb.  3, 2010      Nnoduka Eruchalu        Initial Revision
 *  Feb. 27, 2010      Nnoduka Eruchalu        Updated for final project
 */

#define _XTAL_FREQ 24000000UL             /* the PIC clock frequency is 24MHz */

/* SYSTEM LEVEL HARDWARE CONNECTIONS */
#define GET_IMAGE_LAT      LATC7         /* Get Image signal from PIC to CPLD */
#define GET_IMAGE_TRIS     TRISC7
#define SYS_RESET          RC4


/* SYSTEM CONSTANTS */
#define EEPROM_POS_LOC      0
#define EEPROM_DIR_LOC      1

#define DEGS_PER_MOVE       9       /* should be an int and a multiple of 1.8 */
#define MAX_POS           180       /* relative to 0 degrees and also ints    */
#define MIN_POS          -135       /* that are multiples of 1.8              */

#define NUM_MICS            4                        /* number of microphones */
#define MID_MIC             2      /* index of middle mic, on a 0-based scale */

#define VIDEO_MODE          1      /* only 2 modes allowed, and must be bit */
#define PICTURE_MODE        0      /* bit values.*/


#endif                                                           /* SDIC_H_ */
```