CMPE 490 Design Project

Rotationally Refreshed Display

Patrick Boyd
pboyd@ualberta.ca
Tuesdays, Mondays (some), Wednesdays (some)

Daniel Hill
drhill@ualberta.ca
Available all days

Preferred Lab day will be Tuesday

Abstract

The main aim of this project will be to create a working display that consists of a column of LEDs rotating in a circle around a fixed point. By controlling if a particular LED is on at a particular point in the circle, we can “draw” an image on what would appear to be a cylinder. This will be run by a DC motor and information about the rotational speed will be relayed back to the display hardware located on the spinning apparatus. A sensor will indicate that the device has completed a full rotation and the time taken to complete the rotation will be saved. This information will be required to ensure that the LEDs are in the correct position when they are turned on or off. The display hardware must be self contained as well; no control wires will be run to the hardware as they would become twisted in the rotation.

The components on the display must be as lightweight as possible; they will be spinning quite quickly after all. This means that we can not use the provided ARM board or the FPGA to control the hardware. Instead a PIC16 micro-controller will be used to control the LEDs and monitor motor speed. This will all be mounted inside of a plastic shield to protect people from the very fast spinning components.

Declaration of Original Content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

Support circuitry for PIC microcontroller taken from [8]
Template for PIC assembly code taken from [8]
Entry and Exit subroutines for ISRs taken from [8]
Format for PROGRAM_Read subroutine from [10]
p16873a.inc Library from [10]
Support circuitry for Hall Sensor taken from [4]

Table of Contents

Abstract	2
Function Requirements of Project	4
Design and Description of Operation	4
Parts List	6
Datasheet	9
Software Design	10
Test Plan	13
Results of Experiments and Characterization	15
Citations	16
Appendix	17
Quick Start Manual	17
Future Work	17
Hardware Schematics	19
Source Code	20

Function Requirements of Project

The main functional requirements of this project will be a working rotationally refreshed display. This will be defined as an apparatus that can:

1) Display an image upon what appears to be a cylinder by rotating a column of LEDs in a circle with the same radius as the perceived cylinder
2) Able to display text that marquess around the perimeter of the display.
3) Able to display graphics and simple animations.

Design and Description of Operation

The spinning arm will be mounted inside of a 5 gallon plastic water bottle that will act as a shield between the device and any users. The diameter of the plastic bottle is 0.254m. From tests done on the motor with the full arm assembly attached, it reached a maximum speed of roughly 630RPM. Assuming an arm length of 0.127m, the tangential velocity of the LEDs is 8.37m/s. This speed is well below our ideal speed of 1800RPM. However, it is still sufficiently fast for persistence of vision effects to occur. The major side effect of the lower speed is a flashing quality to the display.

The height of each pixel as determined from the soldering is 3.5mm. To achieve an ideal pixel aspect ratio of 1:1, we would need to turn on the LEDs for each pixel for 3.5mm as the arm swings. This means that with a radius of 0.127m we would have room for 228 pixels, or a little over 28 characters (assuming a 8x8 area for each character). However, in the interests of saving clock cycles for the PIC, we made the display 256 pixels wide. This results in a pixel width of 3.117mm and an aspect ratio of 0.891:1, which is still an acceptable size for the pixels.

With a tangential speed of 8.37m/s and a pixel width of 3.117mm, the length of time that each column of pixels needs to be turned on is $3.72*10^{-4}$s. Since the external oscillator we are using has a frequency of 4MHz which is then divided by 4 internally by the PIC, this corresponds to 372 clock cycles per pixel column. The software as it stands now, takes about 80-90 cycles to output data to the LED drivers, and 40-60 to load new data from Program memory into registers. This leaves more than 100 cycles to spare per column of pixels.

Static Operation –When the PIC is programmed, code and data is loaded into the non-volatile memory of the PIC microcontroller. The code that runs on the microcontroller then sequentially moves through the data stored in memory, calling subroutines to draw the data to the display. The data is a preset buffer structure consisting of a series of graphical objects, either strings or pictures, encoded into a data format used by the display subroutines, that are to be displayed. When a new graphical object is loaded, an offset into memory is initialized to the start of that object. Then for each column of pixels, the data stored at that memory address is loaded into registers, then the LEDs are turned on and off based on the contents of those registers. After a delay, an positional offset is incremented, which shifts the position of the displayed string or picture around the display.

LED Brightness Calculations
Maximum LED Brightness @150mA, 25°C, 0° viewing angle = 7.2lm
Relative Luminous Intensity @75mA, 25°C, 0° viewing angle = 0.6lm
Relative Luminous Intensity @ 2mA, 25°C, 0° viewing angle = 0.005lm

Swing arm radius = 11.5cm
LED diameter = 2.6mm
Relative Brightness of a spinning pixel:
*(PixelDiameter / 2πr) * MaxBright * RelativeRatio*
@75mA, 25°C, 0° viewing angle = 0.01512lm

Brightness of a stationary pixel:
*MaxBright * RelativeRatio*
Relative Luminous Intensity @ 2mA, 25°C, 0° viewing angle = 0.0361lm

Ratio of spinning to stationary = 0.42

So we can see that by increasing the current we can ensure that the brightness of the spinning pixels will be about half of the stationary ones.

Parts List

Parts ordered from DigiKey:

http://www.digikey.com/

DC Motor - salvaged from past project.

2 Line Commutator - salvaged from past project.

Motor Base and Power Connections - salvaged from past project.

Bright LEDs – We have chosen 20 bright “hyper orange” LEDs.

Part No: 754-1382-1-ND

Price: 1.08 each

Quantity: 20

Data Sheet: http://www.kingbrightusa.com/images/catalog/SPEC/AA3535SEL1Z1S.pdf

Reason for Choice: These are cheap, easily visible and with a wide viewing angle. There were green LEDS that were brighter, but were 4.50 a piece. The specs for these come out at 9 lumens, which should be plenty bright.

Update: Upon testing to see how bright these are, their maximum output is almost painful to look at directly.

Hall Effect Sensor – This is a new choice, but due to their application in use with velocity sensors in industry I feel this is a safe choice for our project. I have also been researching similar projects on the Internet and a kit like version of this uses a hall effect sensor. Which is where I got the idea. We will place a magnet just outside of the radius of the swing arm and this should trigger only at that point.

Part No: TLE4906LINCT-ND

Price: 2.00 each

Quantity: 2

Data Sheet: TLE4906LINCT-ND Datasheet

Reason for Choice: This is a hall effect switch. As opposed to a latch, this will only provide current when the sensor is in the presence of a magnetic field that is stronger than the earth's is applied. The field required is not so strong that a fridge magnet or maybe a bit stronger won't do the job though.

PIC16 Microcontroller – We do not need anything particularly heavy duty. Just something to turn on our LEDs with respect to how fact the device thinks it's going. A PIC is a perfect choice for this, the static operation information can be pre-loaded into non-volatile memory or we can connect this to a wireless device.

Part No: PIC16F873A-I/SP-ND

Price: 5.72 a piece

Quantity: 2

Data Sheet: http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf

Reason for Choice: I was looking for something with the DIP package and quite a few IO channels so each of the LEDs could be connected to it's own individual line. Everything else on this chip is pretty standard and will work fine for our purposes.

PICStartPlus Development Programmer

Data Sheet: http://ww1.microchip.com/downloads/en/DeviceDoc/51028f.pdf

Reason for Choice: This programmer was chosen because it was what Ed Tiong had in stock for us to use. It was also compatible with the PIC we were using.

LED Drivers – These are used to control the state of the LEDs and to drive appropriate current to them.

Part No: 620-1165-5-ND

Price: 1.86

Quantity: 4

Data Sheet: http://www.allegromicro.com/en/Products/Part_Numbers/6278/6278.pdf

Reason for Choice: High output current and allows up to 8 LEDs to be driven. As well as supporting a wide range of input voltages and output currents.

We also required a variety of small capacitors and resistors as well as a 4MHz external oscillator, all of which were acquired from the stores provided in the lab and from the store room.

DataSheet

Parameter	Symbol	Operation Values		Units	Conditions
Supply Voltage	Vcc	Min	Max	V	Recommended 5V, as DC motor is connnected to same supply
		3	5		
Supply Current DC Motor	I_{Motor}	Min	Max	A	If the motor stalls, it will attempt to draw more current than the lab supplies can provide. The min value occurs when the motor is at speed and ecountering it's lowest friction.
		1.24	>3		
Circuit	I_C	Min	Max	A	Max occurs when all LEDs are on simultaneously. Min is all off.
		0.02	0.93		
Power Consumption Idle	P_{IDLE}	6.3		W	P = (0.02A + 1.24A)(5V) = 6.3W
Max (Motor at constant speed of ~630 rpm)	P_{MAX}	10.85		W	P = (0.93 A + 1.24 A)(5V) = 10.85

Device	Line	Signals	Function	Direction
PIC16F	2 LED Serial Data Interface Lines	VCC – High Gnd - Low	These lines will be read into the shift register in each LED driver	Output
	1 LED Driver Clock	VCC – High Gnd - Low	Controls the rate at which the LED driver shifts in new signals to it's shift register	Output

	1 LED Latch Enable	VCC – High Gnd - Low	Allows the shift register to read the Serial Data Interface Lines	Output
	1 IRQ/Interrupt	VCC – High Gnd - Low	Will connect to the hall effect sensor and call an interrupt routine to adjust the speed variable	Input

Software Design

The software design for our project can be broken down into 3 major components: the main process loop, the external hall sensor triggered interrupt and the internal timer interrupt.

Main Process Loop

This is the process that is running in the background at all times in our project. Its two main responsibilities are to ensure that the data needed to be displayed for each column is present in the assigned registers, and updating positional offest to ensure the object to be displayed scrolls across the screen. It is also responsible for switching to a new object whenver that is needed.

When a column of pixels has been sent to the LED drivers by the timer interrupt, a bit in a user-defined status register is set high. The main process loop polls this bit, and when it is high, the main loop calculates which memory location to access. This is based on a base address for the current object, the positional offset of the displayed object and the current column that is being drawn. It then goes into program memory using a subroutine, stores the data there into the registers and clears the status bit.

When a certain number of rotations of the arm have passed, the main loop also will increment a positional offset stored in a register. As this offset grows, the position where the current object will begin to be drawn is shifted across the display. Since our display is 256 pixels wide, when this register overflows, it will return to the starting position.

External Interrupt Handler

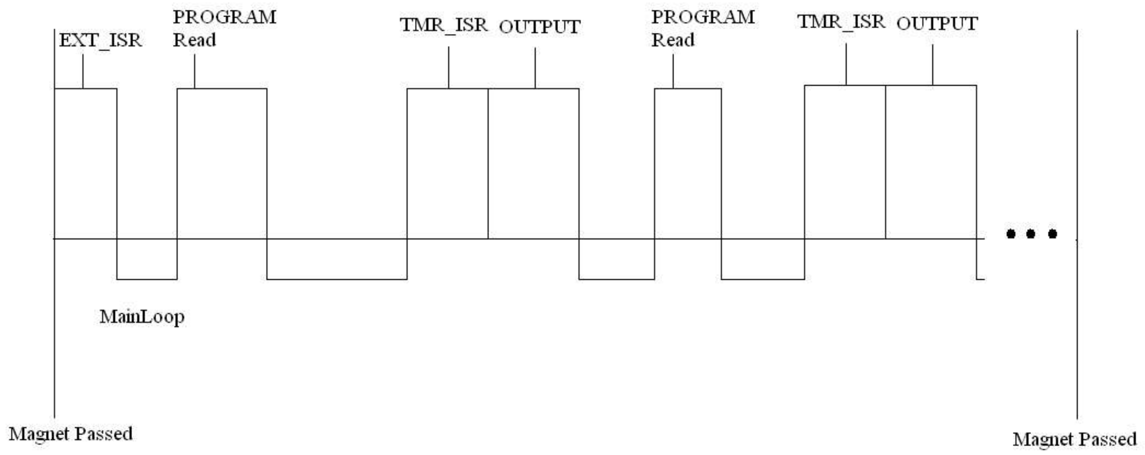
The external interrupt is triggered by the Hall Effect Sensor passing within range of the magnet mounted on the protective shield. Therefore it will run once per rotation of the swing arm. Its primary purpose is to calculate the speed of the motor and set the timing of the pixel columns accordingly.

It calculates the speed of the motor through the use of an onboard 16-bit timer. This timer uses a prescaler that scales the clock input 2:1. This adjusts the range of timings that the timer can measure to a range that the motor actually runs at. This also sets a minimum rotational speed for the motor. If the motor falls below 457RPM the 16-bit timer will overflow before the external interrupt is triggered causing undesired behaviour. The current value of the timer is stored in two registers. Since there are 256 pixels that these cycles need to be divided amongst, we simply take the high register as our value. This register is equal to half the number of cycles that each column of pixels gets. This value is subtracted from 256 to determine the starting value of the 8-bit timer used to control the column timing. Finally the timers are reset and the rotations counter is incremented.

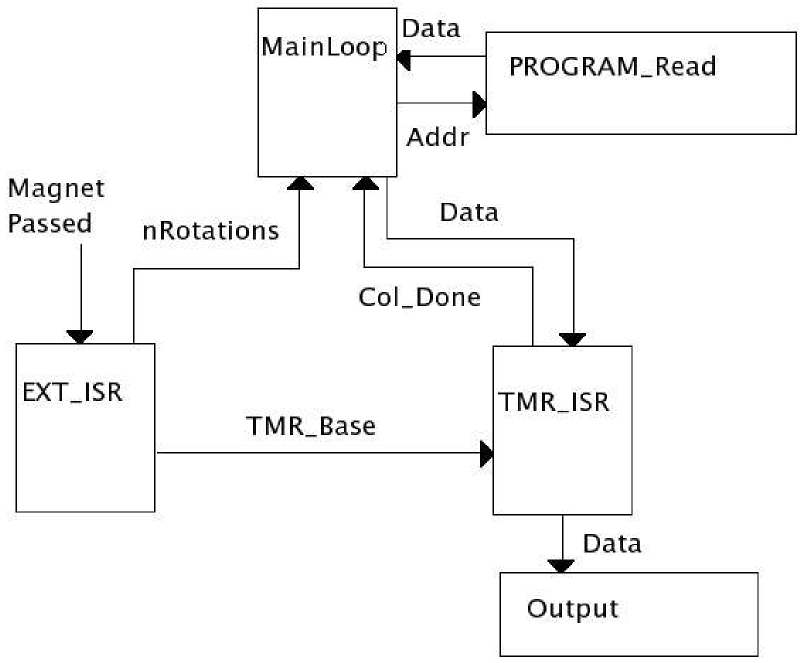
Internal Timer Interrupt

This internal interrupt is triggered when the onboard 8-bit timer, with a starting value calculated in the external interrupt hander, overflows. This signals to the processor that a new column of pixels needs to be sent to the LED drivers. This handler calls an output subroutine to send new data out, resets the 8-bit timer to the base value calculated in the external handler, and sets a status bit to indicate to the main loop that a new column needs to be loaded.

Timing Diagram:



Data Flow Diagram:



Initalization and Data Storage

The PIC requires a large number of register to be initialized for its peripherals. For this project we must initialize the control registers for external and peripheral interrupts, Timer0 and Timer1, and for the EEPROM memory. This is all done at the start of the program code, before the main loop is entered into for the first time.

The data is stored in memory as a series of 8-bit short integers. Each bit corresponds to one LED on the LED driver that it is being sent to. Since the characters are 8 pixels high, and 8 pixels wide, the are stored as 8 short ints. Graphics can also be stored this way using alternating short ints for the high and low LED banks. All of these values must be hard coded onto the PIC beforehand.

Test Plan

Software:

For the software we will do incremental tests, with each one adding a new layer of functionality.

Test 1: Stationary Line
Draw a stationary vertical line where the hall sensor passes the magnet.

Test 2: 4 Stationary Lines
Draw 4 lines quartering the display.

Test 3: Stationary Test Pattern
Draw a stationary test pattern that fills the entire display.

Test 4: Draw Stationary Message
Draw a message loaded from memory, that doesn't move.

Test 5: Draw a Scrolling Message
Draw a message from memory that marquees around the display.

Hardware:

For the hardware, we will test each stage of the circuit from the operations of the LEDs to the ability of the PIC to accurately send out control signals.

LEDs: We will first test the correct operation of the LEDs by providing them with enable signals independent of the PIC to ensure that they will turn on and off, given the correct enable signal.
Status: All of the LEDs have been properly connected to their array and are functional.

LED Driver: A test rig will be assembled to provide the clock and input signals to the LED driver. This will be connected to the LED array and will be checked with various configurations of the LED output to ensure that all of the LEDs are working with the driver. This test also aims to prove that the LED drivers are working before we connect them to the PIC microcontroller.
Status: Both drivers provided the expected functionality once wired to the LED arrays.

PIC: Once we are sure that the LEDs and the LED drivers are operating correctly; we will attach the PIC and test some sample output signals to test whether the PIC can correctly output control signals to the LEDs. A second item used to test was a software based PIC16 simulator we obtained from the internet. We could run our PIC software and monitor the output much easier than doing it in hardware. Once the code was verified to work in software, we could port it over to the hardware.

Hall Sensor: We have already constructed a simple ciruit consisting of a hall sensor attached to an LED to see the hall sensor in action. This worked fine, and also showed us the range that was required of the hall sensor for a weak magnet (1-2cm away minimum). Further tests will be undertaken with stronger magnets.

It is difficult to test whether the hall sensor is accurately measuring the rotational speed of the motor directly. However, if the sensor is not providing accurate measurements, the test case for drawPixel will fail, as a vertical line will not be able to be maintained without a horizontal translation.

Results of Experiments and Characterization

Software Tests:

All tests succesfully completed. After test number 4, the size of the characters was reduced from 16x8 to 8x8. This was due to the protective shielding having a opaque strip that made reading the 16 pixel high characters difficult.

Test of LEDs:

The LEDs were soldered to the frame that will be attached to the spinning arm. With 75mA they were sufficiently bright to be seen from a distance in a lit room.

Test of Hall Sensor:

The test of the Hall Effect sensor was succesful. When hooked up to the circuit detailed in [4], the Hall sensor correctly detected the presence of the magnet at a range well within what we require for our sensor. Using a pair of rare earth magnets we have determined that the hall sensor is capable of detecting the field from 2-3 cm away. This will be acceptable for our needs.

Test of PIC:

The simulator managed to find a fault in our original hardware wiring. The plan was to have the LED driver's CLK line connected to RA4. This turned out to be a problem due to the hardware not displaying the same behaviour as the software simulator. We were able to determine that the RA4 line was incapable of driving the LEDs driver's CLK line. Once we changed this over to the RB1 line on the PIC, the hardware started displaying the same behaviour as the simulator.

Test of Motor:

The test of the motor was reasonably succesful. The approximate RPM of the motor was determined to be 630RPM.

Citations

[1] http://www.robotroom.com/PWM4.html
[2] http://www.ladyada.net/make/spokepov/
Both websites used for parts inspiration
[3] PIC16F87xA Family Datasheet
http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf
[4] Melexis Hall Effect Sensor Datasheet

http://www.melexis.com/prodfiles/0004824_US5881_rev007.pdf

[5] Infineon Technologies Hall Effect Sensor Datasheet

http://www.infineon.com/dgdl/TLE4906DataSheet_20V1_1.pdf?folderId=db3a304412b407950112b408e8c90004&fileId=db3a304412b407950112b4091cd800e5

[6] LED Drivers Datasheet

http://www.allegromicro.com/en/Products/Part_Numbers/6278/6278.pdf

[7] LED Datasheet

http://www.kingbrightusa.com/images/catalog/SPEC/AA3535SEL1Z1S.pdf

[8] E. Tiong, "Resources – EE400/401"

http://www.ece.ualberta.ca/~ee401/resource_circuits.html

[9] OshonSoft PIC Simulator IDE Homepage

http://www.oshonsoft.com/pic.html

[10] Microchip Technology Inc. Homepage

http://www.microchip.com

Appendix

Quick Start Manual

The only thing required for our project to run is a power supply plugged into it. It takes 5V with at least 2.2A. To load a new message onto the display the PIC needs to be reprogrammed. Run the StringEncoder to encode a new message into assembly code. The generated code will be stored in the file stringcode.txt. Copy the contents of the file into the bottom of the main.asm file in the memory location you want to overwrite. Move the two lines with the form:

```
        movlw       0xXX
        movwf       MSG_COUNT
```

Into the section labelled MessageXInit replacing the two lines of that form.

When plugging the PIC into the board on the swing arm, the socket is wired backwards, so the indent on the socket does not line up with the indent on the PIC.

To change the font, change one of the files in the characters folder and then run the CharEncoder program. The next time you run StringEncoder, the encoded string will use the new font.

All work done on the PIC assembly code was done in the MPLAB suite available from the Microchip website. This IDE comes packaged with all necessary libraries.

Future Work

The backup plan for this project, if it appears that the main requirements are in danger of being left uncompleted, is to switch to a 2 dimensional array of LEDs with data being fed in dynamically from the ARM processor. The 2 dimensional array's hardware controller would have a buffer to store images to be displayed, which can be filled dynamically. The controller would be in charge of going through the buffer sequentially and displaying the images. The ARM processor would be used to grab the content to be displayed from an RSS feed on the Internet, Google news for example.

The optional/extension portions of this project would be similar to the backup plan outlined above, but with the rotationally refreshed display used in place of the 2d array.

This brings about other challenges, as for the buffer to be filled dynamically, we will need to create a serial interface with the ARM board along the commutator. Once the device is communicating with a controller, we can go on to use the ARM as a more powerful data processing tool. Longer messages and more inventive graphics can be created using the ARM board, rather than the PIC16. The swing-arm mounted components will fill the role of a video driver and display. They will serve to output a buffer that is filled by the ARM board. From this we would aim to add Ethernet functionality to the ARM board and create an application to harvest information to be displayed from the Internet. For the Internet option we will maintain a list of RSS feeds that can be accessed to provide a continuously updated supply of messages to be shown on the display.

Optional Tasks:

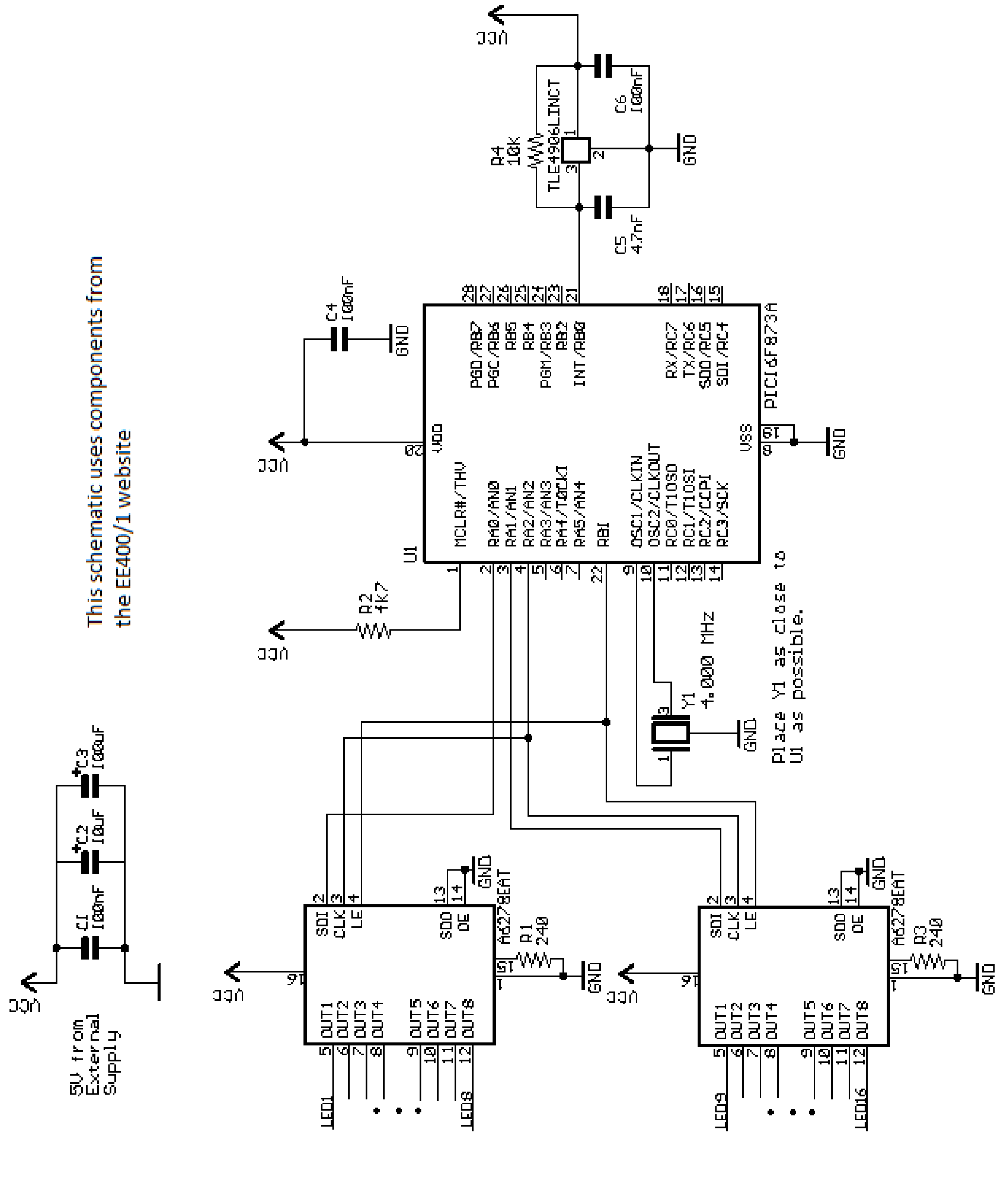
1) Ensure that the image can be dynamically updated
2) Grab the dynamic display information from the Internet
3) Pulse-width Modulation could be utilized to allow for varying LED brightness
4) Full color LEDs could be used in place of the monochromatic ones

Dynamic Operation – The dynamic operation is designed to be quite similar to the static operation. The major difference that is present is the use of the third contact of the commutator (possibly a forth for bi-direction communication). This will be used to receive transmissions of new data coming from the ARM board. When new data is received, an interrupt will be sent to the PIC microcontroller, which will then read the data from the serial interface, and find a place in the memory buffer to store the new data. This allows the data that is being displayed to be constantly changing. Our ultimate goal is also to have the ARM board be synched up to RSS feeds using an Ethernet controller and the internet. This allows for various different themes of messages to be displayed, such as news or stock prices.

Varying LED Brightness – This can be done via pulse-width modulation using only software on the PIC with any drastic changes to the hardware. This would allow for more complicated images to be displayed.

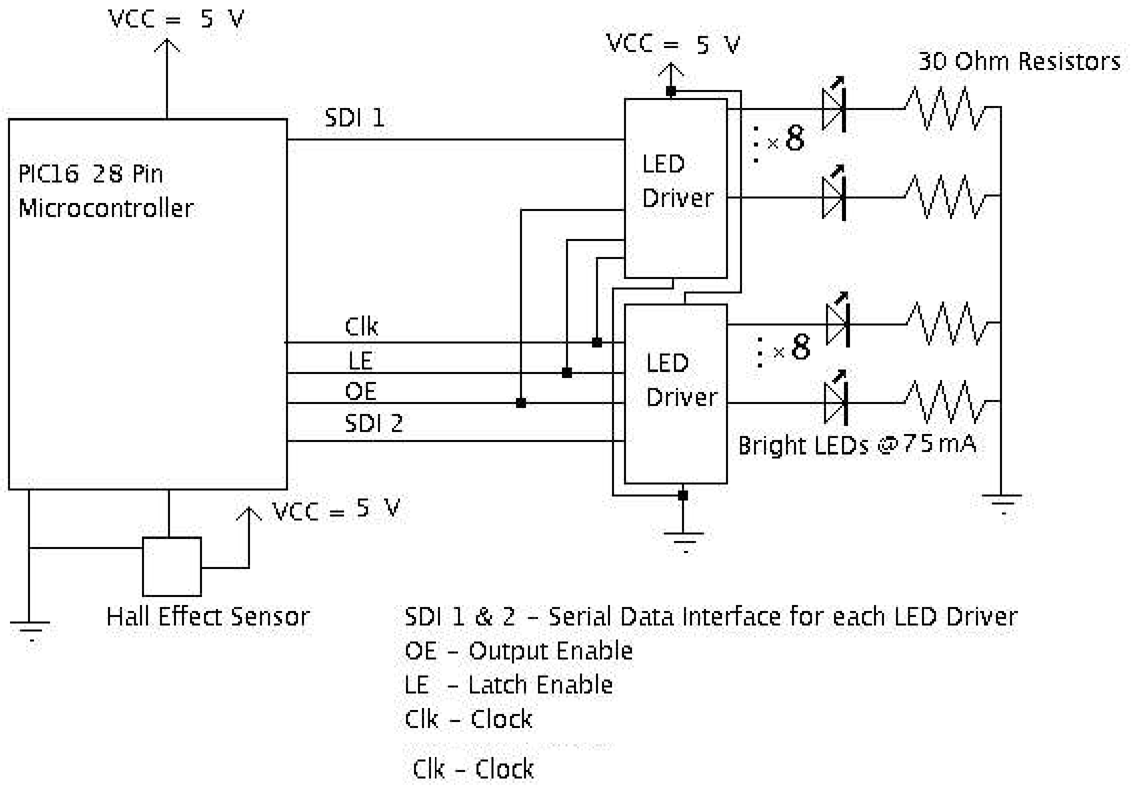
Full Color LEDs – If we had a larger budget we could implement a new version of this project with color by using color LEDs. This does add more complexity to the software and hardware, but the effect would be likely worth it.

Hardware Schematic

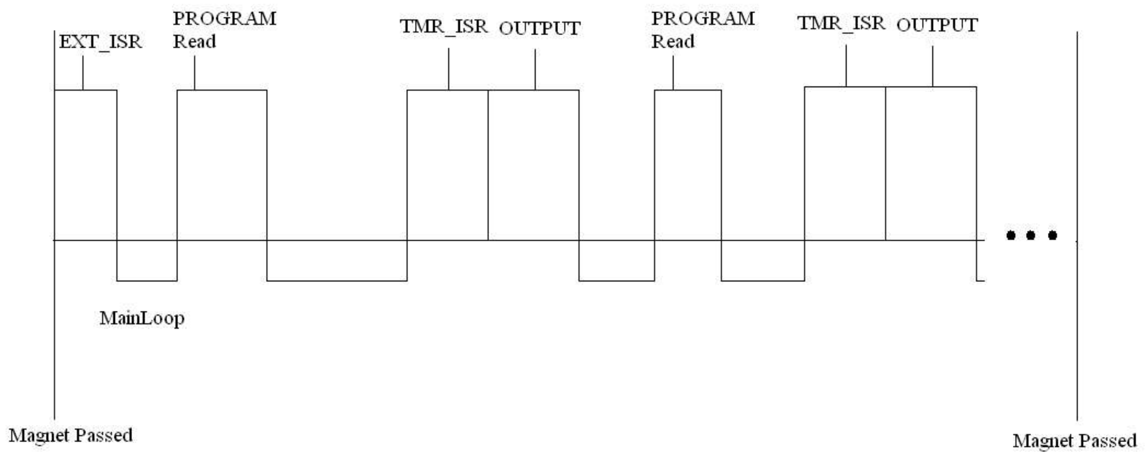


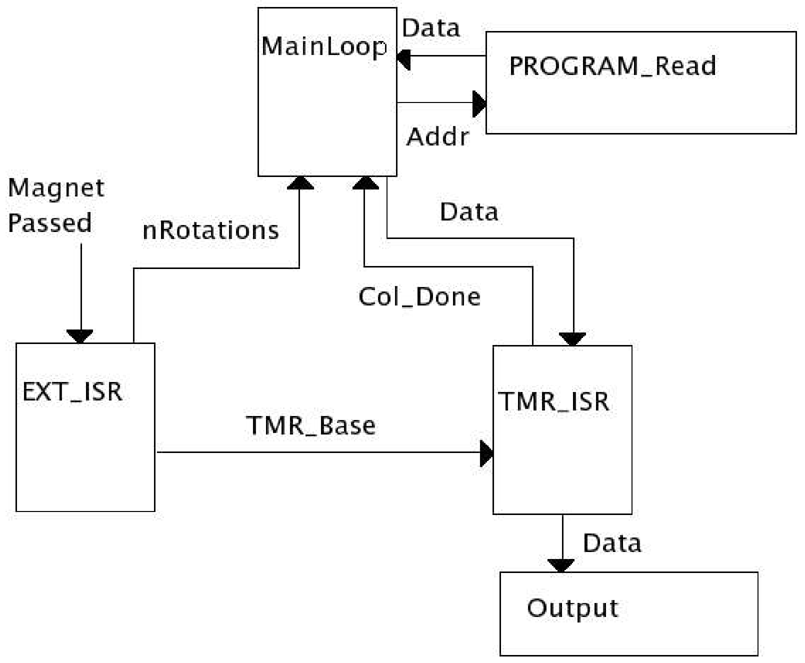
Note: LED1-16 have the same configuration here as in the block diagram below

Block Diagram



Source Code





main.asm – Code flashed onto PIC micro-controller for controlling LEDs and performing timing calculations. Status: **T**
StringEncoder.cpp – Takes in a string and using the output of a previously run CharEncoder creates assembly code to store a string in memory. Status: **T**
CharEncoder.cpp – Takes a collection of text files defining a font and encodes them into integers. Status: **T**