College of Engineering and Applied Science (CEAS)

University of Cincinnati

Advertising Infinity Mirror Display

Randon Rose & James Linneman

June 7, 2012

Submitted in partial fulfillment of the degree of Bachelor of Science in Electrical Engineering Technology

Advisor: Elvin Stepp
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>Purpose</td>
<td>3</td>
</tr>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Project Scope</td>
<td>3</td>
</tr>
<tr>
<td>Problem</td>
<td>3</td>
</tr>
<tr>
<td>Solutions/Methodology</td>
<td>4</td>
</tr>
<tr>
<td>Implementation</td>
<td>5</td>
</tr>
<tr>
<td>Electronic Design</td>
<td>6-7</td>
</tr>
<tr>
<td>Code Design</td>
<td>8-37</td>
</tr>
<tr>
<td>Hardware Design</td>
<td>37-49</td>
</tr>
<tr>
<td>Budget</td>
<td>50</td>
</tr>
<tr>
<td>Conclusion</td>
<td>50</td>
</tr>
<tr>
<td>Appendix</td>
<td>51-53</td>
</tr>
</tbody>
</table>
Abstract:
This project was performed to demonstrate the acquired knowledge gained through the Electrical Engineering Technology program of the University of Cincinnati by means of purposing, designing, and making a senior design project. The project of choice has been titled an Advertising Infinity Mirror Display.

Purpose:
The purpose of an Advertising Infinity Mirror Display is to stand out from the current methods of advertisement by creating a unique visual affect that draws attention to the device and the message that is being presented through it. The project also features a dynamic social network interface that allows the users to quickly and easily update desired advertisements.

Introduction:
The Advertising Infinity Mirror stands out from other forms of advertisement by creating an optical illusion in the form of false depth. The device is designed to attract attention where other forms of advertisement fail, by means of creating an optical illusion with hovering text supplied by a user’s Twitter account.

Project Scope:
- Internet Enabled User Interface:
  - Adjustable Lighting Affects
  - Custom Message Entry
  - Status Updates
- RGB LED Technology:
  - Dynamic Lighting over Full Color Spectrum
- Direct & Indirect Advertising:
  - Direct via Displayed Message
  - Indirect via Word of Mouth
- Credibility
  - 5th year EET senior with proficient course completion in the following:
    - Computer Architecture
    - Digital System I – III
    - Circuit Analyses I – III
    - Embedded Systems
    - Assembly Language
    - C++ Programming

Problem:
In today’s society advertising is a multi-billion dollar industry and with so many signs, billboards, and gimmicks we thought we could find a way to stand out and draw more attraction than the rest. The solution that this project will provide is a means of attracting attention where other forms of advertising might fail. It will engage curiosity and provide a competitive edge in the advertising market.
Solution/Methodology:
The Infinity Mirror Display uses two 2-way mirrors that surround an inner frame of RGB LEDs. Once these LEDs are illuminated, there light is constantly reflected between the two mirrors. It is because of this repetitive reflection that gives the device an illusion of depth. To display the textual message in the voided space of the display, an array of three 16x32 pixel RGB LED matrixes is used. Because these LED matrixes are positioned behind the second 2-way mirror, only the light they produce can penetrate through the mirror, concealing the hardware and making the text to appear to be hovering in space. All of this hardware is then controlled by an onboard micro-controller that interfaces to twitter over a WiFi hotspot. The micro-controller reads the Twitter accounts latest tweet and displays it on the device while also using the Twitter hash-tag convention for both text and ring colors.

We used Arduino micro controller to control the inside ring of LEDs, the Arduino micro controller also was hooked to an Ethernet cable that was sending pulling messages from a twitter account to display the color and effect of the LED ring. The Twitter account it pulled information was set up as a server and we had to hard code into the micro controller to look for tweets from that account only in order to gather the correct information to change the color and or effect of the LEDs. The micro controller was set to search the Twitter account every 30 seconds to see if new information had been provided and it would update to the LEDs accordingly.

The Parallax Propeller micro controller was chosen to run the three 16 x 32 RGB LED matrixes. The Parallax Propeller is a 32 bit 8 core microcontrollers fast enough to drive the matrixes without problem. This micro controller was chosen because of the speed it operates at, needed to handle the multiplexing of the matrixes. Essentially the information which was programed as a static string was displayed on the matrixes and shifted over one LED at a time to create a scrolling effect. This effect allowed the used to enter messages to be displayed at 140 characters, the limit on a single Twitter tweet.

Construction diagram shown below:
**Implementation:**
The schedule had changed a little from the original plan because of multiple unseen factors. The order and receiving times varied because a little more research was needed to determine correct parts needed. A majority of the time was spent on programming the micro; this was the most difficult part of the process. We did receive some help from varied online sources about issues with code we ran into. We did run into time constraints as to having the project be able to wirelessly pull information from a social networking site and display text as planned, but with more time we plan to improve and implement that.

At various stages of planning, building, and coding we ran into minor issues. In the planning stage we had originally planned to have the project working wirelessly in order to pull information from the Twitter account it was associated with. This became a slight issue when we saw that our time was limited and it was more important to have a working demo/project by the deadline.

In the building phase most of the building went as planned, we ran into issues with the display box being able to hold all of the components and hide the wires and electronics correctly in order to give it the best optical illusion possible. This was solved by some carefully planned wood working, building wire ways, mirror grooves, and back panel to house the micro controllers. The back panel was made of UHM Polypropylene, which provided us enough strength to hold the display together and safely house the components.

Coding issues happened more often than not due to the fact that neither one of us had ever programmed in Spin. This problem was easily overcome from asking multiple questions and searching online data bases to gain the information need to proceed to the next step in code. Spin was the language used for the Parallax Propeller, which is a proprietary language used mainly with the Parallax products and free to modify. This was helpful because posting on blogs asking for information on commands we were not familiar with we were able to finish the code in time and have a working project.

See Gantt Chart in Appendices
**Electronic Design:**
16x32x3=1536 LEDs that are controlled by 12 MB15026 LED controller chips. The MB15026 has 16-bit registers, one bit per pixel. This turns 1 of 16 LEDs on or off. The on level current is set by an external resistor. Controlling the 1536 LEDs with 192 outputs used the 74HC138. The 74HC138 is a 3 to 8 demultiplexer that uses three inputs A, B, and C to select 1 of 8 groups of pixels at a time. This means that 1536/8=192 LEDs can be turned on at a time. Because of this we have to show each group of 192 LEDs quickly to avoid flicker. This processes is called 1/8 scanning.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel pitch</td>
<td>6mm</td>
<td>Driving Mode</td>
<td>Constant current</td>
</tr>
<tr>
<td>LED spec</td>
<td>3528/ SMD 3-in-1</td>
<td>LED Wavelength</td>
<td>R:λ(625±5nm)</td>
</tr>
<tr>
<td>Encapsulation form</td>
<td></td>
<td></td>
<td>G:λ(520±5nm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B:λ(470±5nm)</td>
</tr>
<tr>
<td>Pixel configuration</td>
<td>1R1G1B/Real pixel</td>
<td>Working temperature</td>
<td>-10°C – 40°C</td>
</tr>
<tr>
<td>Module size</td>
<td>192mm*96mm</td>
<td>MTBF</td>
<td>≥10000Hours</td>
</tr>
<tr>
<td>Module resolution</td>
<td>32*16</td>
<td>Lifespan</td>
<td>100000Hours</td>
</tr>
<tr>
<td>Weight</td>
<td>0.18Kg</td>
<td>Smoothness</td>
<td>≤1mm/≤0.5mm</td>
</tr>
<tr>
<td>Scanning Mode</td>
<td>1/8scanning</td>
<td>Hub type</td>
<td>LINSN-HUB75</td>
</tr>
<tr>
<td>Brightness</td>
<td>1800cd/m2</td>
<td>Refresh frequency</td>
<td>≥400Hz</td>
</tr>
<tr>
<td>Power consumption</td>
<td>19.2W</td>
<td>Working Voltage</td>
<td>DC 5V</td>
</tr>
<tr>
<td>Viewing angle</td>
<td>H:160° / V:120°</td>
<td>Best view distance</td>
<td>≥6-33m</td>
</tr>
</tbody>
</table>

**Power:**
The 74HC138 demultiplexer cannot supply enough current to control 192 LEDs. Each of the 8 outputs goes to a SSF4953 dual p-channel mosfet to provide the needed current.

**Data Control:**
There are 6 data inputs: R1, G1, B1, R2, G2, B2, R3, G3, B3.
Data Shifting: Since there are 6 data inputs controlling 12, 16-Bits LED Drivers, the first group of the 16-Bits are sent to the first chip and shifted to the second allowing the next group of 16-Bits to be sent. The data shifted in is NOT applied until we toggle the “latch” signal input. The shift out on the second chip can output to the input of another board. So for three panels 96-Bits would need to be
shifted out.

**Interfacing:**
there are two 74HC245 octal bus transceivers that buffer all inputs. This allows us to connect directly to the Parallax Propeller’s 3.3V outputs. There is also an OE “Output Enable” input pin. This signal fans out to all LED controllers. If using more than 1 panel, this can be used to share the same data pins. To do this you need an OE output from the micro for each panel.

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Pin number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9</td>
<td>The lowest bit of row address</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>The second lowest bit of row address</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>The highest bit of row address</td>
</tr>
<tr>
<td>LE</td>
<td>14</td>
<td>Latch of row</td>
</tr>
<tr>
<td>CLK</td>
<td>13</td>
<td>Clock</td>
</tr>
<tr>
<td>EN</td>
<td>15</td>
<td>Enable</td>
</tr>
<tr>
<td>R1</td>
<td>1</td>
<td>Red data 1</td>
</tr>
<tr>
<td>R2</td>
<td>5</td>
<td>Red data 2</td>
</tr>
<tr>
<td>G1</td>
<td>2</td>
<td>Green data 1</td>
</tr>
<tr>
<td>G2</td>
<td>6</td>
<td>Green data 2</td>
</tr>
<tr>
<td>B1</td>
<td>3</td>
<td>Blue data 1</td>
</tr>
<tr>
<td>B2</td>
<td>7</td>
<td>Blue data 2</td>
</tr>
<tr>
<td>GND</td>
<td>4,8,12,16</td>
<td>Grounding</td>
</tr>
</tbody>
</table>

Please note schematics in appendices
**Code Design:**

Code had to be developed to drive the 16x32 RGB LED Matrix displays as well as the interior RGB LED Light Strip and interfacing to Twitter. Two microcontrollers were used for these tasks. A parallax propellor was chosen for the task of driving the LED matrix displays due to its high speed. Below is a full documentation of the code used for the parallax propellor microcontroller, written in Parallax’s language called Spin. The code is commented as best possible to describe what steps are being performed.

16x32 Driver Code –

VAR 'Variables to pass to assembly drivers
   long balance 'variable to scale input RGB values for color balance
   long Intensity 'variable to reduce brightness by modulating the enable pin (0..31)
   long BasePin1 'starting pin of 12 pins required for first panel
   long BasePin2 'starting pin of 6 pins required for second panel (or -1 to disable)
   long BasePin3 'starting pin of 6 pins required for third panel (or -1 to disable)
   long EnablePin456 'reserved (to implement 6-panel support)
   long pOutputArray 'pointer to precalculated array of outputs

PUB Start(pSettings)|i, j, k, section, bit,bits, c0, Pin_A, Pin_EN 'Show a 1bpp bitmap
   'retrieve settings
   long movel(@balance,pSettings,7)

   'Configure assembly driver pins before starting
   pinmask:=%1111_1111_1111<<BasePin1 '12 pins starting at BasePin1
   datamask:=%11_1111<<BasePin1
   clkmask:=(1<<9)<<BasePin1
   LeMask:=(1<<10)<<BasePin1
   enmask:=(1<<11)<<BasePin1

   if (BasePin2>1)
      'Enable second panel pins
      pinmask|=%11_1111<<BasePin2 '6 pins starting at BasePin2
      datamask:=%11_1111<<BasePin2

   if (BasePin3>1)
      'Enable second panel pins
      pinmask|=%11_1111<<BasePin3 '6 pins starting at BasePin3
      datamask:=%11_1111<<BasePin3

   'Launch assembly driver cog
   CogNew(@AsmEntry,pOutputArray)
return

DAT 'Matrix Assembly driver
    org 0
AsmEntry 'Start of driver
    mov dira.pimask 'enable pin outputs
MainLoop 'Start of the main display loop
    mov t1,par 'get address to pixel data (passed to assembly via cognew as parameter)
    mov t2,#8 'doing 8 sections
SectionLoop 'Start of section loop
    mov t5,#8 'there are 8 bits of color info, 3 colors for 24bpp color
    mov t7,#8
    sub t7,t5
    mov t8,#1
    shl t8,t7
    'if t5 is 8 (LSB), then t8 will be 1
    'if t5 is 1 (MSB), then t8 will be 128
    LevelLoop 'Start of loop to output to display (doing this t8 times)
    'This 32 rdlong, or, add loop is unrolled to make it as fast as possible
    ' #0
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
    or outa,clkmask 'Toggle the clock
    add t1,#4
    rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
'#$8
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
or outa,clkmask 'Toggle the clock
add t1,#4
rdlong outa,t1
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
'#$24
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
add    t1,#4
rdlong outa,t1
or     outa,clkmask 'Toggle the clock
sub     t1,#32*4
djnz    t8,#LevelLoop

'If we're done with this bit, point to next bit's data
add    t1,#32*4
djnz    t5,#Bitloop 'Do the next bit

djnz    t2,#SectionLoop 'Do the next section

'Start Over
jmp     #MainLoop
```plaintext
{
#******************* Defined data *******************
}

zero        long    0       'constants
d0          long    $200
pinmask     long    0       'This setting must be defined before starting cog
datamask    long    0       'This setting must be defined before starting cog
clkmask     long    0       'This setting must be defined before starting cog
LeMask      long    0       'This setting must be defined before starting cog
enmask      long    0       'This setting must be defined before starting cog

{
#******************* Undefined data *******************
}

'temp variables
t1           res    1       ' Used for DataPin mask and COG shutdown
t2           res    1       ' Used for CLockPin mask and COG shutdown
t3           res    1       ' Used to hold DataValue SHIFTIN/SHIFTOUT
t4           res    1       ' Used to hold # of Bits
t5           res    1
t6           res    1
t7           res    1
t8           res    1
```
16x32 Graphics Code –

VAR 'Configuration variables passed here from main routine
   long balance 'variable to scale input RGB values for color balance
   long Intensity 'variable to reduce brightness by modulating the enable pin
   (0..31)
   long BasePin[3] 'starting pin of three panels
   long EnablePin456 'reserved
   long pOutputArray 'pointer to precalculated array of outputs
   long Arrangement 'organization of panels
'Extra variables for assembly driver
   long pGammaR 'address of red gamma table
   long pGammaG 'address of green gamma table
   long pGammaB 'address of blue gamma table
   long pFont5x8 'address of 5x8 font

VAR 'Variables for assembly drivers
   long cog, command

VAR 'Balanced Gamma look up tables
   byte GammaR[256]
   byte GammaG[256]
   byte GammaB[256]

VAR 'Variables to support the standard text output commands, bin, hex, dec,
and out
   long row, col, rows, cols, flag, forecolor, backcolor
   long colors[8 * 2] '8 possible sets of forecolor and backcolor
   long nPanels

CON 'Some standard colors in regular 24-bit RGB format
   red= 255<<16 + 0<<8 + 0
   blue= 0<<16 + 0<<8 +255
   green= 0<<16 +255<<8 + 0
   black= 0<<16 + 0<<8 + 0
   white= 255<<16 +255<<8 +255
   dk_blue= 0<<16 + 0<<8 +64
   yellow= 255<<16 +255<<8 + 0

CON
   'Enumerate assembly driver commands
   #1, UpdateConfig, AsmSetPixel, DrawLimitedChar, AsmShowBitmap

PUB Start(pSettings):okay|i 'Initialize graphics and start assembly cog
   'retrieve settings from caller
   long move(@balance,pSettings,8)
'initialize output
InitializeOutputArray

'Calculate Balanced Gamma look up tables
UpdateGammas

'Point assembly to font table
pFont5x8:=@Font5x8

'start graphics support cog2
stop 'Stop any running cogs
command:=UpdateConfig<<16+@balance 'copy settings when starting cog
okay := cog := cognew(@init, @command) + 1

'Position text cursor to upper left corner
nPanels:=3 'you may want to change this to 2 or 1 if you have less panels
colors[0]:=yellow
colors[1]:=dk_blue
forecolor:=colors[0]
backcolor:=colors[1]
row:=col:=0
out(0) 'clear screen
nPanels:=1 'calculate number of panels from assigned pin numbers (negative if not present)
if BasePin[1]>0
    nPanels:=2
if BasePin[2]>0
    nPanels:=3
case arrangement
0: 'side-by-side landscape
   cols:=32*nPanels/5
   rows:=16/8
return

PUB Stop   '' Stop assembly support driver - frees a cog
if cog
    cogstop(cog~ - 1)

PRI UpdateGammas|i,r,g,b 'calculate the balanced gamma tables for each color
pGammaR:=@GammaR
pGammaG:=@GammaG
pGammaB:=@GammaB
r:=GetRValue(balance)
b:=GetBValue(balance)
g:=GetGValue(balance)
repeat i from 0 to 255
  GammaR[i]:=Gamma[i]*r/255
  GammaG[i]:=Gamma[i]*g/255
  GammaB[i]:=Gamma[i]*b/255

PRI InitializeOutputArray|i, j, section, bit, bits, c0, Pin_A, Pin_EN
  'Fill in the address and enable bits into the precalculated output array
  Pin_A:=BasePin[0]+6
  Pin_EN:=BasePin[0]+11

  'precalculate initial outputs
  repeat section from 0 to 7
    repeat bits from 0 to 7
      repeat bit from 0 to 31
        i:=section*32*8+bits*32+bit
        long[pOutputArray][i]:=0  'init to zero (probably not required)
        if bits==0
          k:=section-1
        else
          k:=section
        if k<0
          k:=7

        long[pOutputArray][i]|=k<<Pin_A  'set the section bits
        if bit=>Intensity  'disable display on last bit, before latch
          long[pOutputArray][i]|=1<<Pin_EN

PUB ShowBitmap(x0,y0,pBmp)|p,x,y,w,h,i  'Show a bitmap
  'This needs to be sped up with assembly code!
  'Read bitmap header
  p:=ReadBitmapHeader(pBmp)
  x:=x0
  y:=y0
  w:=biWidth
  h:=biHeight

  SetCommand(AsmShowBitmap,@p)

PUB SetAllPixels(c)|x,y,z
  z:=nPanels*32-1
  case Arrangement
0: 'Panels are side-by-side with total height=16
repeat y from 0 to 15
  repeat x from 0 to z
  SetPixel(x,y,c)

PUB RGB(r,g,b)|level 'create a long color from 3 color bytes
  'Blue byte first because that's how stored in bitmap...
  return ((r<<8+g)<<8)+b

PUB GetRValue(c)
  return (c>>16)&255

PUB GetGValue(c)
  return (c>>8)&255

PUB GetBValue(c)
  return c&255

PUB SetPixel(x,y,c) 'Set a pixel with assembly
  SetCommand(AsmSetPixel,@x)

PUB DrawChar5x8(x,y,c,fore,back)|i,j,k 'Draw a 5x8 font character,c, with upper
left corner giving by x and y
  'Will draw with specified foreground and background colors (unless set to -1)
  DrawLimitedC(x,y,c,fore,back,0,95)

PUB DrawText5x8(x,y,pString,fore,back)|i,j,k 'Draw a 5x7 string with upper left
corner giving by x and y
  repeat i from 1 to StrSize(pString)
    DrawLimitedC(x+(i-1)*6,y,byte[pString+i-1],fore,back,xmin,x)
  if ms<1 'need to wait at least 1 ms
    ms:=1

PUB ScrollText5x8(x,xmin,y,pString,fore,back,ms)|i,j,k,i2,j2,z,c 'Scroll text left
  from x back to xmin
  'Doing it the dumb way here and just drawing the whole string every time
  repeat k from 0 to StrSize(pString)*6
    'loop over each column in message
    repeat i from 1 to StrSize(pString)
      c:=byte[pString+i-1]
      DrawLimitedC(x+(i-1)*6-k+1,y,c,fore,back,xmin,x)
    if ms<1 'need to wait at least 1 ms
      ms:=1
waitcnt(cnt+ms*(clkfreq/1000))

PUB Dec(value) | i " Print a decimal number with 5x7 font using current row, col, forecolor and backcolor

if value < 0
    -value
    out("-"
    i := 1_000_000_000

repeat 10
    if value => i
        out(value / i + "0")
        value //= i
        result~~
    elseif result or i == 1
        out("0")
        i /= 10

PUB hex(value, digits) " Print a hexadecimal number with 5x7 font using current row, col, forecolor and backcolor

value <<= (8 - digits) << 2
repeat digits
    out(lookupz((value <= 4) & $F : "0"..'9', "A"..'F"))

PUB bin(value, digits) " Print a binary number with 5x7 font using current row, col, forecolor and backcolor

value <<= 32 - digits
repeat digits
    out((value <= 1) & 1 + "0")

PUB out(c) | i, k " Output a character or move cursor with 5x7 font using current row, col, forecolor and backcolor

"$00 = clear screen
 "$01 = home
 "$08 = backspace
 "$09 = tab (8 spaces per)
 "$0A = set X position (X follows)
 "$0B = set Y position (Y follows)
case flag
$00: case c
  $00: SetAllPixels(backcolor)
  col := row := 0
  $01: col := row := 0
  $08: if col
      col--
      $09: repeat
          print(" ")
        while col & 7
  $0A..$0C: flag := c
  return
  $0D: newline
  other: print(c)
$0A: col := c // cols
$0B: row := c // rows
$0C: forecolor := 2<<(c & 7)
    backcolor := 2<<(c & 7)+1
flag := 0

PRI Print(c) 'Output a character with 5x7 font using current row, col, forecolor and backcolor

DrawChar5x8(col*6,row*8,c,forecolor,backcolor)
if ++col == cols
  newline

PRI newline | i 'this needs work...

  col := 0
  row:=rows-1
  return

if ++row == rows
  row--
  ScrollScreen(0,8)
  repeat col from cols-1 to 0
    DrawChar5x8(col*6,row*8," ",forecolor,backcolor)

PUB str(stringptr)

" Print a zero-terminated string

repeat strsize(stringptr)
out(byte[stringptr++])

PUB ScrollScreen(x,y) 'Shift contents of screen by given x and y amounts 'Not yet implemented

PUB DrawLimitedC(x,y,c,fore,back,xmin,xmax)|z,i,j
SetCommand(DrawLimitedChar,@x)

PRI setcommand(cmd, argptr)

    command := cmd << 16 + argptr                         'write command and pointer
    repeat while command                                     'wait for command to be cleared,
        signifying receipt

DAT 'Assembly graphics support
    org     0
AsmEntry             'Start of driver
init
loop

    'Check for command
    rdlong  t1,par       wz
    if_z  jmp     #loop  'Wait for command

    mov address,t1                   'preserve address location for passing
                                         'variables back to Spin language.

    ror t1,#16+2                     'lookup command address
    add t1,#jumps
    movs :table,t1
    rol t1,#2
    shl t1,#3
:table        mov     t2,0
    shr t2,t1
    and t2,#$FF
    jmp t2                            'jump to command

jumps       byte    0                                 '0
            byte    UpdateConfig_                     '1
            byte    AsmSetPixel_                      '2
            byte    DrawLimitedChar_                  '3
            byte    AsmShowBitmap_                    '4
LoopEnd       wrlong  zero,par  'zero command to signify command complete
NotUsed_      jmp     #loop

DAT UpdateConfig_ 'Update operating parameters
  mov       t3,#12
  movd      GetArg,#AsmBalance
  call      #GetSettingsSub
  jmp       #Loop

DAT GetArgumentsSub 'retrieves t3 number of arguments
  movd      GetArg,#arg0  'get up to 7 arguments ; arg0 to arg6
GetSettingsSub 'enter here if you want to set destination to somewhere besides arg0
  mov       t2,address   'mov   t3,#7                       '?---+
  GetArg    rdlong    arg0,t2  'function is in the upper word of t1,t2, but this is ignored...
     add       GetArg,d0
     add       t2,#4
     djnz      t3,#GetArg

    'going to say we're done now, because we have the aruguments already
wrlong  zero,par  'zero command to signify command complete
GetArgumentsSub_RET
GetSettingsSub_RET
ret

DAT DrawLimitedChar_  'draw a character at given x and y but limit to xmin, xmax
  mov       t3,#7
  call      #GetArgumentsSub
'arg0=x0
'arg1=y0
'arg2=char
'arg3=fore color
'arg4=back color
'arg5=xmin
'arg6=xmax

  mov       t9,#6
  mov       x1,arg0
  mov       t12,AsmPFont5x8
  add       t12,arg2
  rol       arg2,#2
  add       t12,arg2
DLC_xLoop
cmp      x1,arg5 wz,wc
if_b jmp  #DLC_xLoopEnd
cmp      x1,arg6 wz,wc
if_a jmp  #DLC_xLoopEnd
mov      t11,#8
mov      y1,arg1

rdbyte   t10,t12
DLC_yLoop
    ror      t10,#1 wc
    mov      pixelc,arg4
if_c mov  pixelc,arg3
    cmp      t9,#1 wz,wc 'last column?
if_e mov  pixelc,arg4
    mov      pixelx,x1
    mov      pixely,y1
call     #SetPixelSub
    add      y1,#1
djnz      t11,#DLC_yLoop
DLC_xLoopEnd
    add      x1,#1
    add      t12,#1
djnz      t9,#DLC_xLoop
    'all done
    jmp      #loop
DLC_LastLine

DAT AsmShowBitmap_  'Display a 24bpp pixel at given x and y
    mov      t3,#5
call     #GetArgumentsSub

'arg0=address of bitmap bits
'arg1=x0
'arg2=y0
'arg3=Width
'arg4=Height

    mov      y1,arg2
AsmBmpLoopY
    mov      x1,arg1
    mov      t9,arg3
AsmBmpLoopX
    rdbyte    pixelc,arg0
    sub       arg0,#1
    rdbyte    t1,arg0
    sub       arg0,#1
    rdbyte    t2,arg0
    sub       arg0,#1
    rol       pixelc,#16

    rol       t1,#8
    add       pixelc,t1
    add       pixelc,t2
    mov       pixelx,x1
    mov       pixely,y1
    call      #SetPixelSub
    add       x1,#1
    djnz      t9,#AsmBmpLoopX
    add       y1,#1
    djnz      arg4,#AsmBmpLoopY

' all done
    jmp       #loop

DAT AsmSetPixel_  'Set Pixel
' Note: Setting a pixel is a bit complex because pixel information is stored in the
        precalculated output array
    ' We need to set 3 bits in each of 8 longs in this array
    ' We also need to figure out which panel each pixel is in
    ' We are also going to balance and gamma correct the color
    mov       t3,#3
    call      #GetArgumentsSub
    mov       pixelx,arg0  ' arg0:=x
    mov       pixely,arg1'  arg1:=y
    mov       pixelc,arg2'  arg2:=c

    call      #SetPixelSub
' all done
    jmp       #loop

DAT SetPixelSub 'subroutine to set pixel at pixelx, pixely with color pixelc

    rol       pixelc,#1 wc  'exit if color is negative
    if_c  jmp       #SetPixelSub_RET
    ror       pixelc,#1

'calculate t2:=offset and check that x and y are in bounds
    movs      MovBasePin,#AsmBasePin1 'reset initial source
'Look at x
    cmp   pixelx,#0 wz,wc
    if_b  jmp   #SetPixelSub_RET  'x too small
    cmp   pixelx,#32 wz,wc
    if_ae sub    pixelx,#32
    if_ae add   MovBasePin,#1
    cmp   pixelx,#32 wz,wc
    if_ae sub    pixelx,#32
    if_ae add   MovBasePin,#1
    cmp   pixelx,#32 wz,wc
    if_ae jmp   #SetPixelSub_RET       'x too big
MovBasePin  mov    t2,AsmBasePin1

'Make sure basepin is valid
    cmp   t2,#32 wz,wc
    if_ae jmp   #SetPixelSub_RET  'pixel is on a panel that is not present

'Now, look at y
    cmp   pixely,#0 wz,wc
    if_b  jmp   #SetPixelSub_RET  'y too small
    cmp   pixely,#8 wz,wc
    if_ae sub    pixely,#8
    if_ae add   t2,#3
    cmp   pixely,#8 wz,wc
    if_ae jmp   #SetPixelSub_RET       'y too big

'Calculate gamma corrected rgb
    mov    t1,pixlc
    ror    t1,#16
    and    t1,#255
    add    t1,AsmPGammaR
    rdbyte   red_.t1
    mov    t1,pixlc
    ror    t1,#8
    and    t1,#255
    add    t1,AsmPGammaG
    rdbyte   green_.t1
    mov    t1,pixlc
    and    t1,#255
    add    t1,AsmPGammaB
    rdbyte   blue_.t1

'Shift colors all the way left so we can shift into carry later
    rol    red_.#24
    rol    blue_.#24
    rol    green_.#24
'calculate base address of 8 longs (need to add in the i*32<<2 part with t4 later)
  mov  t5,pixely 't5=y
  shl  t5,#8
  add  t5,pixelx 't5=(y*8)*32+x
  shl  t5,#2
  add  t5,AsmPOutputArray 't5=[pOutputArray][(y*8)*32+x]

't7 will be mask for 3 bits
  mov  t7,#%111
  shl  t7,t2 't7= %111<<offset

'Now, write three bits of RGB to 8 longs in output array
'Starting with MSBit
  mov  t1,#8  't1=i+1
  ASP_loop
    'calculate address of long to alter, t4
    mov  t4,t1   'calculate offset for this bit, t4
    sub  t4,#1
    shl  t4,#(5+2)
    add  t4,t5
    rdlong  t6,t4  't6=long[pOutputArray][(y*8+i)*32+x]
    andn  t6,t7

    'shift left-most color bits into carry and then shift carry to form BGR bits (blue first)
    rol  blue_,#1 wc
    rcl  t8,#1
    rol  green_,#1 wc
    rcl  t8,#1
    rol  red_,#1 wc
    rcl  t8,#1
    and  t8,#%111

    'save result
    shl  t8,t2
    or   t6,t8
    wrlong  t6,t4
  djnz  t1,#ASP_Loop
SetPixelSub_RET
  RET

DAT {########################## Defined data
###########################}
zero long 0 'constants
d0 long $200
pVariables long 0

{
#----------------------------------- Undefined data
#-----------------------------------
}
address res 1
arg0 res 1
arg1 res 1
arg2 res 1
arg3 res 1
arg4 res 1
arg5 res 1
arg6 res 1
arg7 res 1
arg8 res 1

x1 res 1
y1 res 1
x2 res 1
y2 res 1
c1 res 1
c2 res 1

t1 res 1
'temp variables
shutdown

t2 res 1
'temp variables
shutdown

t3 res 1
'temp variables
shutdown

t4 res 1
'temp variables
shutdown

t5 res 1
'temp variables
shutdown

t6 res 1
'temp variables
shutdown

t7 res 1
'temp variables
shutdown

t8 res 1
'temp variables
shutdown

t9 res 1
'temp variables
shutdown

t10 res 1
'temp variables
shutdown

t11 res 1
'temp variables
shutdown

t12 res 1
'temp variables
shutdown

t13 res 1
'temp variables
shutdown

'12 parameters to be updated
AsmBalance res 1
AsmIntensity res 1
AsmBasePin1 res 1
PRI ReadBitmapHeader(pB):bmpAddress|p,i
 'analyze bitmap image
 p:=pB
 bytemove(@bfType,p,2) 'read bmp header
 p+=2
 bytemove(@bfSize,p,4) 'read bmp header
 p+=4
 bytemove(@bfReserved1,p,4) 'read bmp header
 p+=4
 bytemove(@bfOffBits,p,16) 'read bmp header
 p+=16
 bytemove(@biPlanes,p,4)
 p+=4
 bytemove(@biCompression,p,24)
 p+=24
 'calculate actual bytes in palette
 i:=bfOffBits-54

 p+=i
 return p+(biHeight*biWidth*3)-1 'return pointer to last byte in bitmap

DAT 'Data space for reading bitmap files

BMPHeader byte 'Mostly using info from here:
bfType byte "B","M"
bfSize long 0
bfReserved1 word 0
bfReserved2 word 0
bfOffBits long 54
biSize long 40
biWidth long 0
biHeight long 10
biPlanes word 1
biBitCount word 24
biCompression long 0
biSizeImage long 0
biXPelsPerMeter long 0
biYPelsPerMeter long 0
biClrUsed long 0
biClrImportant long 0

DAT 'Gamma curve
gamma byte 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
byte 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5
byte 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 11, 11, 12, 12, 13, 13, 14
byte 14, 14, 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 23, 24, 25
byte 25, 26, 27, 27, 28, 29, 29, 30, 31, 31, 32, 32, 33, 34, 34, 35, 35, 36, 37, 37, 38, 39, 40
byte 41, 42, 42, 43, 44, 44, 45, 46, 47, 48, 49, 50, 51, 52, 52, 53, 54, 55, 56, 57, 59, 60
byte 61, 62, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 75, 77, 78, 79, 80, 82, 83, 84
byte 85, 87, 88, 89, 91, 92, 93, 95, 96, 98, 99, 100, 102, 103, 105, 106, 108, 109, 111
byte 112, 114, 115, 117, 119, 120, 122, 123, 125, 127, 128, 130, 132, 133, 135, 137, 138
byte 140, 142, 144, 145, 147, 149, 151, 153, 155, 156, 158, 160, 162, 164, 166, 168, 170
byte 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 197, 199, 201, 203, 205
byte 247, 250, 252, 255

DAT '// standard ascii 5x8 (really 6x8 because need a padding column) font from Adafruit example code

font5x8
byte $00, $00, $00, $00, $00
byte $3E, $5B, $4F, $5B, $3E
byte $3E, $6B, $4F, $6B, $3E
byte $32, $49, $49, $49, $32
byte $32, $48, $48, $48, $32
byte $32, $4A, $48, $48, $30
byte $3A, $41, $41, $21, $7A
byte $3A, $42, $40, $20, $78
byte $00, $9D, $A0, $A0, $7D
byte $39, $44, $44, $44, $39
byte $3D, $40, $40, $40, $3D
byte $3C, $24, $FF, $24, $24
byte $2B, $2F, $FC, $2F, $2B
byte $FF, $09, $29, $F6, $20
byte $60, $7E, $7E, $09, $03
byte $20, $54, $54, $79, $41
byte $00, $00, $44, $7D, $41
byte $30, $48, $48, $4A, $32
byte $38, $40, $40, $22, $7A
byte $00, $7A, $0A, $0A, $72
byte $7D, $0D, $19, $31, $7D
byte $26, $29, $29, $2F, $28
byte $26, $29, $29, $29, $26
byte $30, $48, $4D, $40, $20
byte $38, $08, $08, $08, $08
byte $08, $08, $08, $08, $38
byte $2F, $10, $C8, $AC, $BA
byte $2F, $10, $28, $34, $FA
byte $00, $00, $7B, $00, $00
byte $08, $14, $2A, $14, $22
byte $22, $14, $2A, $14, $08
byte $AA, $00, $55, $00, $AA
byte $AA, $55, $AA, $55, $AA
byte $00, $00, $00, $FF, $00
byte $10, $10, $10, $FF, $00
byte $14, $14, $14, $FF, $00
byte $10, $10, $FF, $00, $FF
byte $10, $10, $FO, $10, $FO
byte $14, $14, $14, $FC, $00
byte $14, $14, $FF, $00, $FF
byte $14, $14, $F4, $04, $FF
byte $14, $14, $17, $10, $1F
byte $10, $10, $1F, $10, $1F
byte $14, $14, $14, $1F, $00
byte $10, $10, $10, $FO, $00
byte $00, $00, $00, $1F, $10
byte $10, $10, $10, $1F, $10
byte $10, $10, $10, $FO, $10
byte $00, $00, $00, $FF, $10
byte $00, $00, $FF, $01, $03
byte $E0, $80, $FF, $00, $00
byte $08, $08, $6B, $6B, $08
byte $36, $12, $36, $24, $36
byte $06, $0F, $09, $0F, $06
byte $00, $00, $18, $18, $00
byte $00, $00, $10, $10, $00
byte $30, $40, $FF, $01, $01
byte $00, $1F, $01, $01, $1E
byte $00, $19, $1D, $17, $12
byte $00, $3C, $3C, $3C, $3C
byte $00, $00, $00, $00, $00
The following code is for the Arduino microcontroller, which handles the tasks of making a TCP connection, requesting data from Twitter, processing said data and controlling the RGB LED Light Strip. Its code is as follows with comments to explain each step.

Twitter and LED Code –

//--Include-Libraries-------------------------------------------------------------------------------------------
#include <SPI.h>
#include <Ethernet.h>
#include <TextFinder.h>

//--Ethernet-Constants---------------------------------------------------------------------------------------
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0xD3, 0xE3 };
IPAddress ip(192,168,137, 2);

//--Variables----------------------------------------------------------------------------------------------------
int counter = 1;

EthernetClient client;
String TwitterName ="EETSeniorDesign";
char tweet[140];
char hashtag[140];
String SearchString ="<title>";
byte charsize;
char serverName[] = "api.twitter.com";

char effectcode;
char colorcode;

int rLEDpin = 3;
int gLEDpin = 5;
int bLEDpin = 6;

char color;
int fadeValue;

//--Set-Up Function-------------------------------------------------------------------------------------------
void setup()
{

}
Serial.begin(9600); 
Ethernet.begin(mac, ip);

analogWrite(rLEDpin, 255); // turn on all LEDs for white
analogWrite(gLEDpin, 255);
analogWrite(bLEDpin, 255);

delay(3000);
}

void loop()
{
    Serial.print("Query: ");
    Serial.println(counter);
    Serial.println("Connecting to Twitter");

    TextFinder finder( client,2 );
    client.print("GET /1/statuses/user_timeline.rss?screen_name=");
    client.print(TwitterName);
    client.println("&count=1 HTTP/1.1");
    client.println("HOST: api.twitter.com");
    client.println();

    while (client.connected())
    {
        if (client.available())
        {
            Serial.println("Searching for Tweet");
            if ((finder.find("<item>")&&finder.getString("EETSeniorDesign:","</title>",tweet,140))&&finder.getString("#","</title>",hashtag,140))
                Serial.print("Found Tweet: ");
                Serial.println(tweet);
                Serial.print("Found Hashtag: ");
                Serial.println(hashtag);
                effectcode = hashtag[0];
                colorcode = hashtag[1];
                break;
}
delay(1);
Serial.flush();
client.flush();
client.stop();

//---Use-Hashtag-to-determine-LED-color-------------------------------------------------------------------

if (colorcode == 82) // hashtag = #?R : set color = red
    color = rLEDpin;
else if (colorcode == 71) // hashtag = #?G : set color = green
    color = gLEDpin;
else if (colorcode == 66) // hashtag = #?B : set color = blue
    color = bLEDpin;
else // hashtag color error
{
    Serial.print(colorcode);
    Serial.println(" = Invalid Color Code");
}

//---Use-Hashtag-to-determine-LED-effect-------------------------------------------------------------------

if (effectcode == 70) // hashtag = #F? : fade LEDs
{
    analogWrite(rLEDpin, 0); // turn off LEDs for update
    analogWrite(gLEDpin, 0);
    analogWrite(bLEDpin, 0);
    for(int x = 0; x < 10; x++) // ~30 second delay loop
    {
        for(fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) // fade max to min
        {
            analogWrite(color, fadeValue);
            delay(30);
        }
        for(fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) // fade min to max
        {
            analogWrite(color, fadeValue);
            delay(30);
        }
    }
    analogWrite(color, 255); // hold max value for update
}
else if (effectcode == 83) //hashtag = #S? : make LEDs solid color
{
    analogWrite(rLEDpin, 0); // turn off LEDs for update
}
analogWrite(gLEDpin, 0);
analogWrite(bLEDpin, 0);
analogWrite(color, 255); // solid max color
delay (30000); // 30 second delay
}
else //hashtag effect error
{
    Serial.print(effectcode);
    Serial.println(" = Invalid Effect Code");
    delay (30000); // 30 second delay
}
Serial.println();

counter++; //increment counter
} //End

Hardware Design
The frame of the Advertising Infinity Mirror Display was designed in Google Sketchup 8. This 3D modeling CAD software offered from Google free of charge allowed us too quickly and precisely design a structurally sound frame to mount all the optical and electrical components. Detailed views of each component are shown below.
Special consideration was given to wiring. The above image shows a hidden path in the material that allows the wire to be ran though the frame and out an adjoining member.

**Left Vertical Member**

**Isometric View**

**Top View**
Right Vertical Member

Isometric View

Top View

Bottom View
Left Horizontal Member

Isometric View

Top View

Bottom View
Corner Support with Wiring Hole

Isometric View

Bottom View
Corner Support without Wiring Hole

Isometric View

Bottom View
Budget:

<table>
<thead>
<tr>
<th>Description</th>
<th>Estimated Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mirrors (2)</td>
<td>$274.00</td>
</tr>
<tr>
<td>LED Strip</td>
<td>$50.00</td>
</tr>
<tr>
<td>LED Display (3)</td>
<td>$210.00</td>
</tr>
<tr>
<td>Arduino Microcontroller</td>
<td>$45.00</td>
</tr>
<tr>
<td>Ethernet Module</td>
<td>$47.00</td>
</tr>
<tr>
<td>Parallax Propeller Microcontroller</td>
<td>$75.00</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>$30.00</td>
</tr>
<tr>
<td>Hardware (Frame Materials)</td>
<td>Donated</td>
</tr>
<tr>
<td>Total Estimate</td>
<td>$731.00</td>
</tr>
</tbody>
</table>

The cost of the project may seem high at first glance but we do believe we could cut the cost down dramatically in the future. The mirrors were the most expensive components but if we were to select a different type or even a different size to be able to mass produce and sell at a reasonable price.

Conclusion:
During our demonstration at the Technology Exposition we were able to verify our theory that the Advertising Infinity Mirror Display would quickly and effectively attract attention to itself and in turn, the message that was being displayed. Going forward we plan on refining the device to make it more affordable so that it will be able to be installed in a variety of businesses.
Appendix:

Hardware Schematics:

Arduino to RGB Light Strip:
Parallax Propeller:
Arduino:

Reference Designs are provided "AS IS" and "WITH ALL FAULTS." ARDUINO DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, RELATING TO PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. ARDUINO may make changes to specifications and product descriptions at any time without notice. The customer must not rely on the absence of characteristics or any features or instructions marked "reserved" or "undefined." Arduino reserves the right to make changes to its products and services, including discontinuing or altering any future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information. Arduino is a registered trademark. Use of the ARDUINO name must be compliant with http://www.arduino.cc/en/Main/Policy.