Transparent Concrete with Embedded Image Recognition LED Array

Le Grand, Raymond
Muldoon, James
Roslyakov, Stanislav

Theory (Conceptual Design):

This project consists of a an LED array that will be placed behind a translucent optical fiber embedded concrete in order to generate images. The prototype has 20 rows and 20 columns, and employs the concept of persistence of vision. This works like a common television but with a lower refresh rate. In order to actuate each row and column field effect P channel MOSFETs will be used. A microcontroller that has over 40 digital pins will be used. An image will be formed by iterative activation in each row which provides power to the LEDs in that row, and then activating each column which grounds the required LEDs. The microcontroller cycles through each row above 20Hz, which makes the flashing almost invisible to the human eye. A control system built in NI-LabVIEW was developed in order to control what will be displayed.

Mechanical Design and Assembly:

The mechanical design of the system consisted of a concrete block embedded with optical fibers, a board with dimensions of 8in x 8in x .5in to hold the LEDs, a cardboard box to enclose the LEDs, and a perforation board for extra circuitry. The assembly of these respective parts was accomplished using standard tools available in the developer's respective labs at NYU-Poly. The concrete block was created by a company called Lucan, based in Germany.

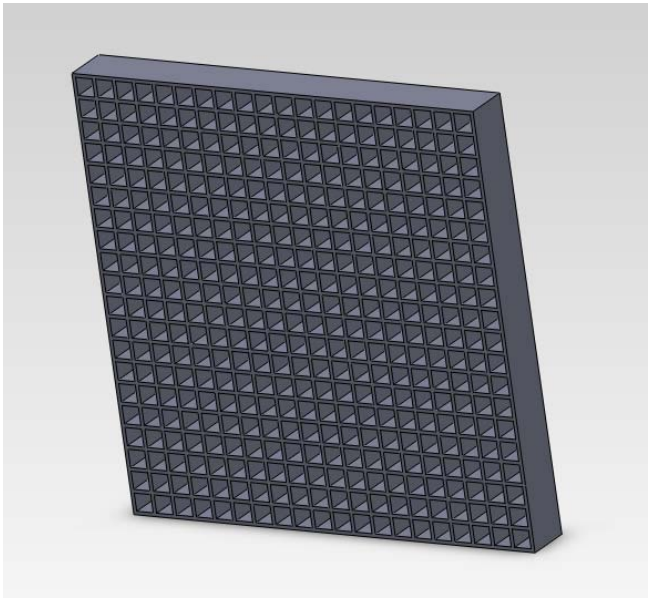

Figure 1 – Initial Solid works design

In the beginning of this project, a Solidworks design was created to house the LEDs, as shown in the Figure 1 on the right. Each hollow area would house 1 LED, and the entire frame would be mounted on the back of the concrete block. After further investigation it was concluded that the design would cost $100 and 48 hours to print. This was inadequate in terms of time and money costs, so a regular composite board was used instead.

The composite board was scored with a caliper in order to set up a grid system for the LEDs. Then, a drill press with laser guides was used to drill 400 holes in the board, as shown in Figure 2. These holes were 5/16in diameter. Next, the resistors were attached to

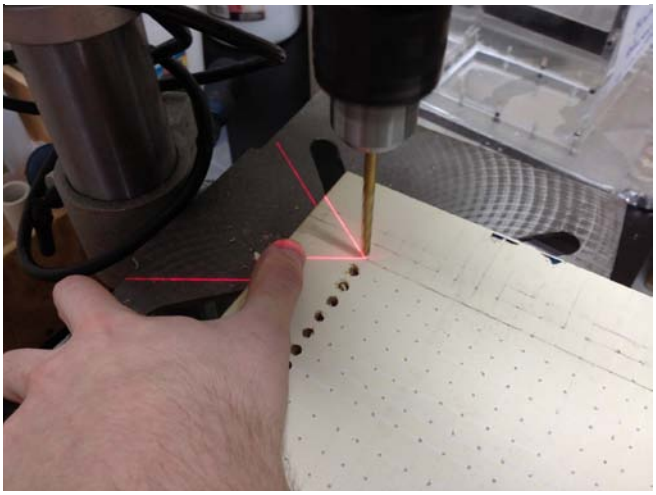

Figure 2 – Drilling the LED housing holes in composite board

the LEDs and then placed into their respective holes. All 400 LEDs were then secured with hot glue. Following that, each positive side of the LED was connected to a wire running along the rows of the board that would provide power to the LED. Then, the negative side of the LED, through the resistor, was connected to a wire running along the columns of the board, thus providing a ground. Figure 3 shows the results of this stage of assembly.

In addition to creating the board of LEDs, part of the construction process involved setting up a perforation board to house the main Arduino Mega microcontroller, the 40 Mosfets driving the LED display, and the other peripheral components. These peripheral components included an emergency power shutoff switch, two reed relays which serve as actuators for the LED display, and a photoresistor to allow for sensing the lightness or darkness of the surroundings. This board also houses the USB connection for the Arduino, as well as the terminals (shown in green) to connect the external 7v power supply. The resulting board is displayed above in Figure 4.

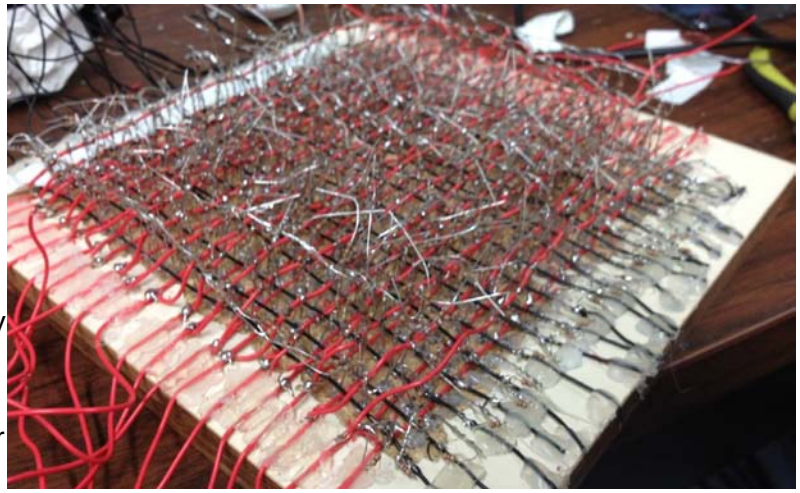

Figure 3 – Half Wired Board

Once the above components were assembled, a cardboard cover was placed over the back of the LED display in order to protect it from wires being bent, shorted, or disconnected. Then, the perforated board was attached to the cardboard box, and the wires from the LED board were attached. The results of this operation are shown in Figure 5. The final step was to secure the LED display to the concrete block. Once this was done, the setup was complete, as shown in Figure 6

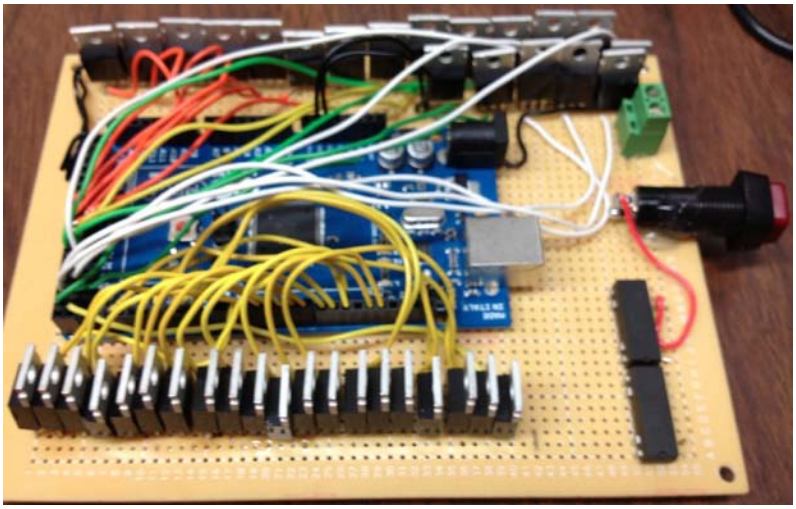

Figure 4 – Control Board

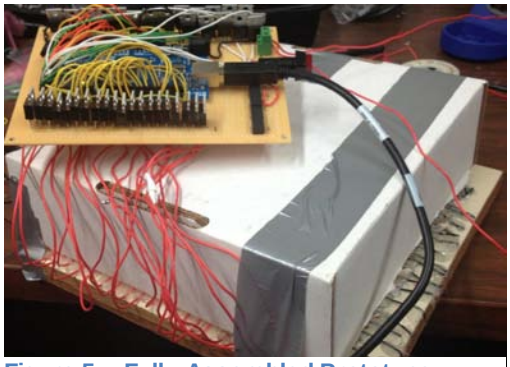

Figure 5 – Fully Assembled Prototype

Figure 6- LED Display under the concrete

Electronic Circuits:

A simple double reed relay (Figure 8) circuit with a simple switch was used to provide and cut power to the LED display. If the switch is closed, the current flows through both of the Reed Relays, with a coil current drain of 40mA for each relay. The reed relays were rated for 12V coil max voltage therefore it was safe to use 7V to actuate them. The reason for using 2 reed relays was that the power supply used to provide the 7V, can supply around 2A while each relay is rated for less, therefore in a case of a short the double relay will be safer to operate.

The MOSFETS used were FQP3P50-ND field effect P-channel. They are voltage driven, therefore a resistor was not provided at the gate. The voltage drop acrss the MOSFET was 1V. The voltage drop across the LED was 2.7V. Using this information the resistor was sized. A portion of the LED driving circuit can be seen in Figure 7..

$$7V - 2 * 1V - 2V = 3V$$
$$V = IR$$
$$3V = 40mA(R) \quad \rightarrow \text{Peak continuous current was rated at 50mA.}$$
$$R = 75Ohm \quad \rightarrow \text{therefore based on vendor a 82 Ohm resistor was chosen.}$$

This allowed us to calculate the power requirements of the entire LED display.
The MOSFET are field effect, therefore draw zero current. If the display is implemented correctly, only one row can be on at a time.

LED Current Draw: $40mA * 20 = 800mA$
Reed Relays: 80mA
Arduino: Draws power from USB; 100mA should be provided if running from main power

Power Supply Requirements: 1A

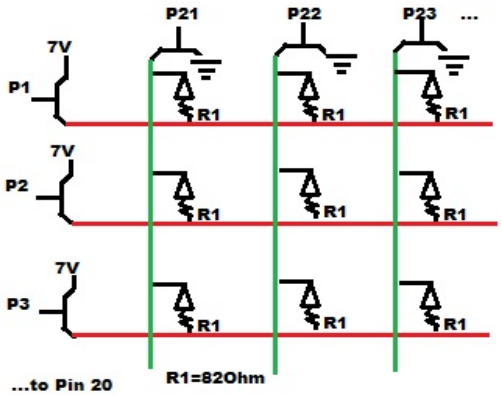

Figure 7 – Main LED Circuit with 6 Mosfets

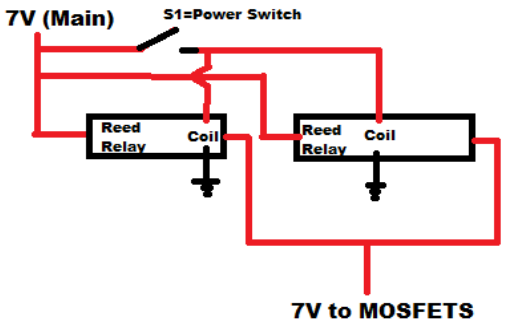

Figure 8 – Reed Relay Power

The arduino had a voltage divider with a photo resistor as an analogue sensor to measure light in the room. Once the light values were low and a jumper pin was grounded, the display started. If the jumper pin was high, the display started regardless of light readings. This sensor is shown in Figure 9.

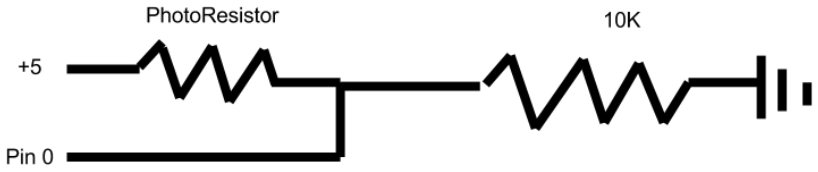

Figure 9 – Photo resistor Circuit for activating light sensitivity mode.

Bill of Material and Prototype Cost

The LED Display prototype contained the materials listed below.

Part	Vendor	Part Number	Quantity	Price per [USD]	Total [USD]
Orange LEDs	DigiKey	365-1190-ND	400	0.11	44.00
Perf-Board	DigiKey	3396K-ND	1	2.45	2.45
MOSFETS	Digikey	FQP3P50-ND	40	0.93	37.20

Reed Relay	DigiKey	HE102-ND*	2	1.50	3.00
Arduino Mega	Sparkfun	DEV-11061	1	58.95	58.95
82ohm Resistor	DigiKey		400		7.00
Total:					152.60

*Read Below

The use of around 25ft of soft core or braided wire was required. Hot glue and electrical tape is necessary to insulate junctions of row and column connections for each LED. The Reed relay can be any Hamlin series, whose coil nominal voltage is 5V.

Cost analysis for mass production:

To mass produce this layout, a PCB board can be manufactured to house surface mount LEDs in their own pockets. This PCB can be produced in modular squares with i2c or SPI communication in order to arrange the byte arrays in order amongst all the panels. PCB will reduce the short circuit issues that you can have with open wires, and the resolution can be improved since the holes that were drilled were limited to composite board integrity. For mass production a custom made microcontroller can be developed using the same ATmega chip, but with less functionality but enough to complete the tasks. This would reduce cost drastically, and allow for slicker design, since the Arduino board is bulky. The mosfets can be manufactured into an array chip, to reduce the use of 40 mosfets individual ICS.

Detailed analysis:

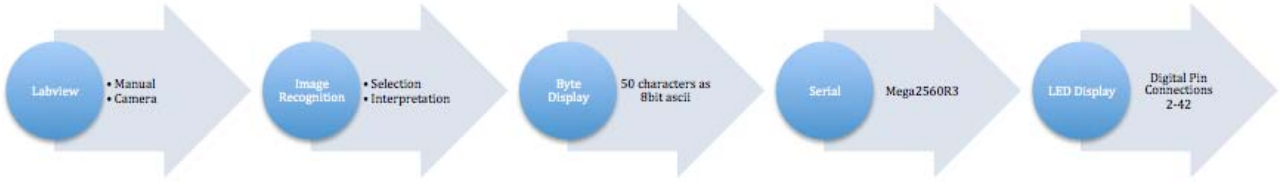

Figure 10 – Flow Chart

Labview is used in conjunction to the Arduino implementation. In Labview one selects the mode by which to requisition the LED data input. If image is chosen then an image will be processed through a matlab script built into the LabVIEW VI and display a pattern to show a silhouette. If manual mode is chosen, then each LED can be selected individually to draw a picture (Figure 11). When the program is used in image mode (Figure 12), there is a threshold knob that will allow one to select the amount of data being filtered out of the image to obtain a desirable result.

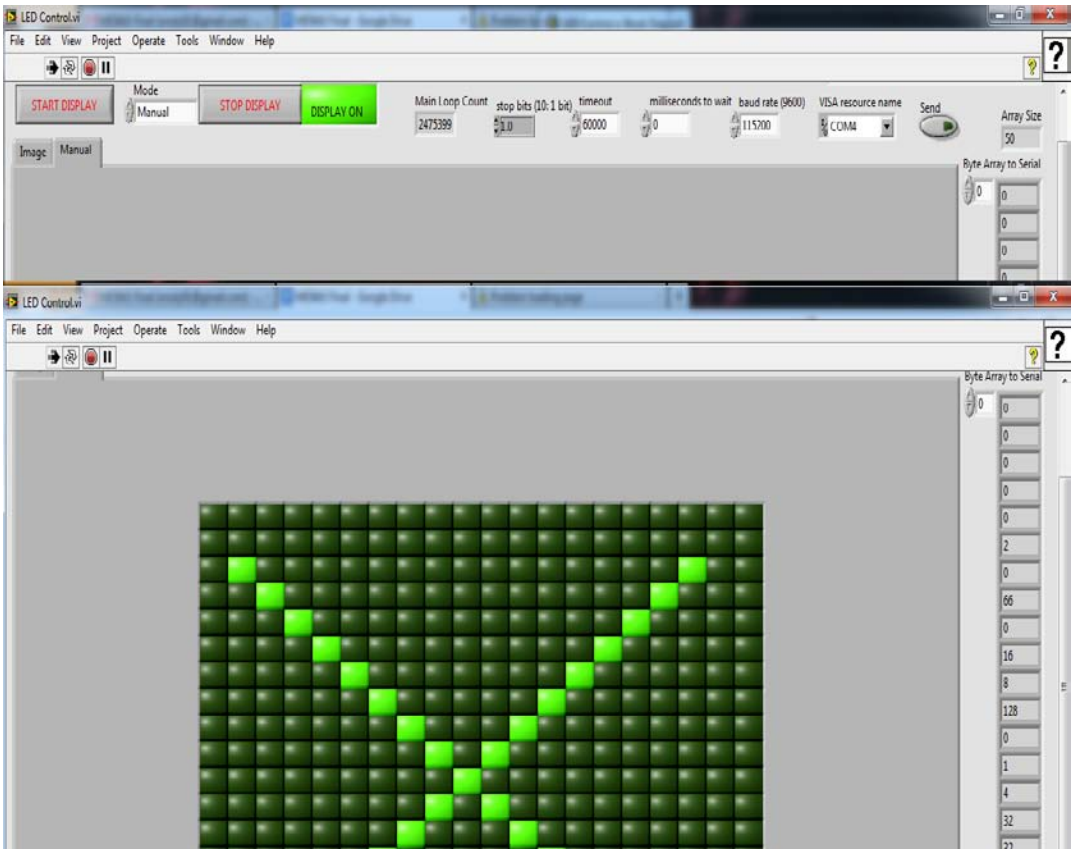

Figure 11 – Manual Mode drawing operation

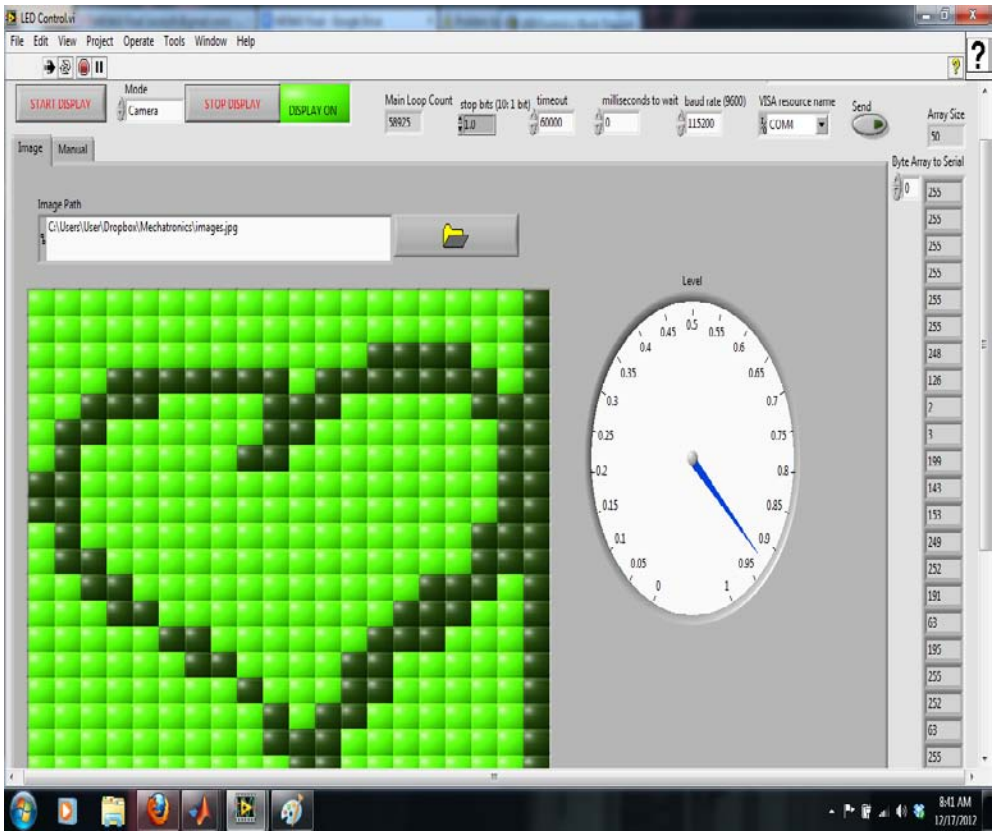

Figure 12 – Image Mode recognition.

Results from the mode will be the following 400 LEDs in the matrix being displayed as single bits arranged into 50 bytes. These 50 bytes are in order to meet the limitations of the Arduinos Serial buffer of 64 bytes [1]. An alternative to this method would be to use the Serial.flush() functionality to sanitize the buffer, but synchronization issues may become an issue and data could be lost in the process. Figure 13 shows the boolean array building for manual mode. Append array is used to combine 20 boolean values for each row and then another append array is used to combine the 20 arrays into one 400 bit array. The values are then combined into 50 bytes using boolean array to byte array conversion. Figure 14 shows the use of a matlab script which uses black and white functions to turn a JPEG image into a 20x20 pixel image. We then append the black and white values of 400 for each pixel into a byte wise array of 50 in which each byte represents 8 pixels/LEDs.

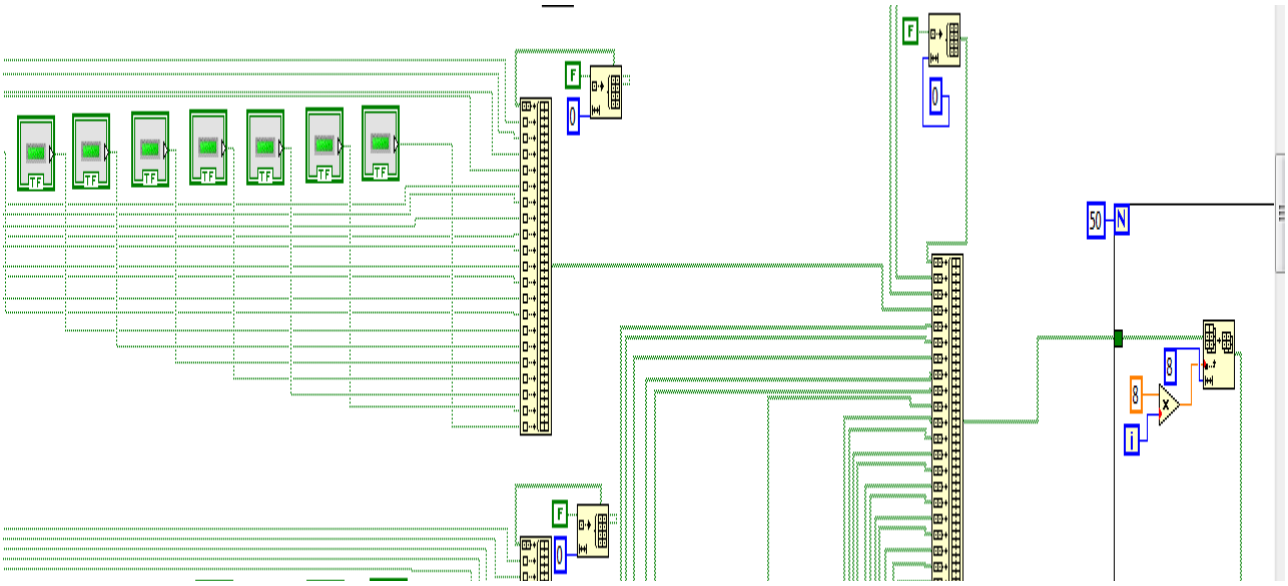

Figure 13– Manual Mode Array Building

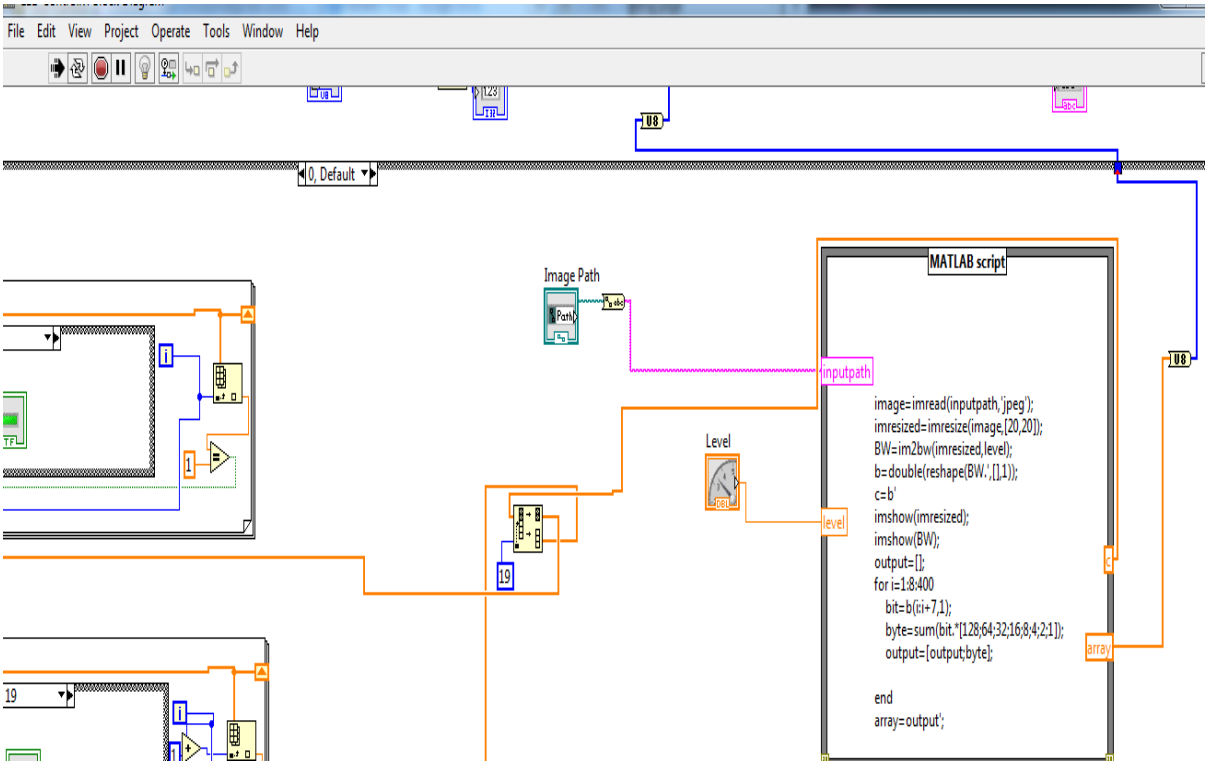

Figure 14 – Image Mode Array Building though Matlab script and interpreting it using the manual mode LEDs as indicators.

The 50 byte string is then read in serially by the Arduino Mega 2560R3 and processed by the state machine shown below in the Figure X. The state machine was setup for two primary reasons. The first was in order to ensure that the data flow would accurately reflect what was being transferred to be displayed from Labview. The state system is simple and increments forward upon completion of its task, however it will not go back. The second reason that the

state machine was implemented was in order to be able to expand in future implementations the dynamic switching of images. This would force the state machine to go from its final state back to the beginning (which is currently not the case for our current implementation)

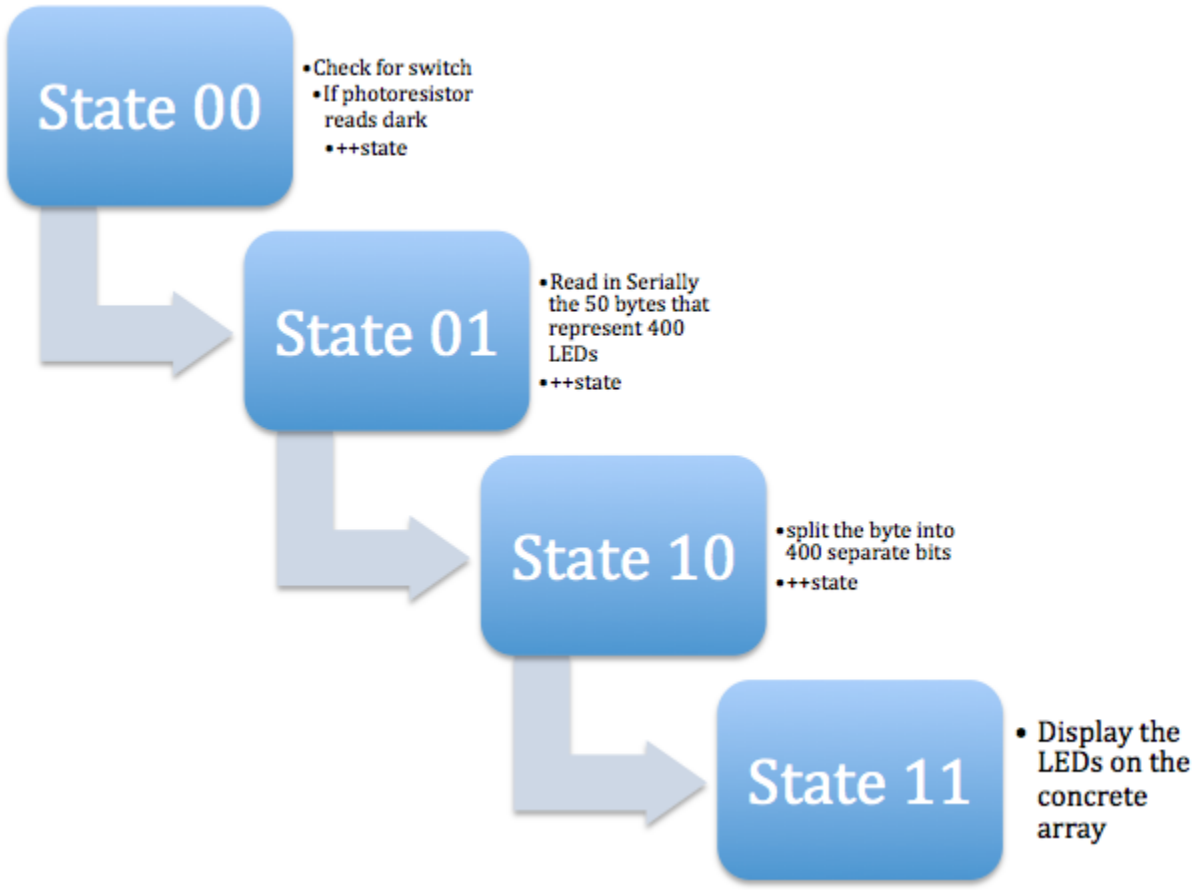

Figure 15 – State Source Machine

State 00:

- This state is the first initial state once the system has made it to the arduino's section. The state will hang at 00 if the switch is toggled "High" until the photoresistor has an analog read in of below the threshold which would indicate that it is dark. Else if the switch is toggled "low" the state will increment and the next time the main loop is called, it will enter State 01.

State 01:

- This state will read in serially the 50 bytes that are being transferred. Each bit in the 50 bytes represents a single LED in the 20x20 matrix. Once this stage is complete the state will increment to the next.

State 10:

- The 50 bytes will be split into each bit singly and stored in an array of 400. The state increments once this process is concluded.

State 11:

- Once in the final state the LED array will actually be powered using the digital io pins 2-44. As was mentioned earlier, the image is going to be displayed by making use of

persistence of vision. Only one row will be lit at any time but the entire column that contains pins that need to be lit up shall be. This will be moving at a high enough refresh rate or frequency above 20Hz that the human eye will not be able to detect the individual aliasing.

Justification in Advantages and Disadvantages:

One of the bigger advantages to using the state machine is the the program flows cleanly so that it can be readily edited later on for future expansion. Another reason the state machine was implemented was because it excludes portions of the program from running unless it is in the state that is necessary to run it. This ensures that the program will operate correctly with relative ease of coding. This is common practice in the making and implementation of controllers in high level and hardware description languages such as VHDL/Verilog.

We chose to implement our design using a 20x20 instead of a more simplistic 4x4 in order to truly show a decent resolution in the picture that is being represented. The advantage here from a prototyping perspective is that in the case of a prototype that needs to be marketed to a client, the 20x20 is considerably more impressive.

Our choice in using the Arduino Mega 2560R3 over that of the BS2 was because of the ease in implementation of the code as well as the amount of digital and analog pins. The large majority of the digital pins are being used currently. Additionally, the ability for the Mega to serially communicate with up to four separate serial channels allowed us to debug issues that we ran into much more efficiently.

We chose to go with a persistence of vision design due to the time frame that we had in order to fabricate the design. In order to do we used Mosfets due to familiarity with them. Had we gone with the alternative route we would have needed instead of a common ground, we would have needed to have a unique ground and a third transistor in order to ensure that the design was operational to turn on each individual LED separately in any combination. The original proposal for this was to be built using BJTs instead of Mosfets, though the design was later thought to be too costly on time.

The use of a digital switch over another kind was partially for simplicity of implementation, but also mainly because there would be no reason to overcomplicate the switch if it was purely for an on/off toggle that determined automatic or manual operation.

A photoresistor was chosen specifically because we could obtain an analog reading of the amount of light that was currently in the room. This is important because the LED array works best when there is lower light. Consequently the code was set up to run based on the photoresistor determining that it was dark enough assuming the operational mode was in automatic (the digital switch was turned on).

We chose reed relays because they were able to actuate on a lower voltage. Additionally, they produce a lower kickback.

We chose Mosfets over BJTs because they have high impedance and draw negligible current. This is advantageous because the Mega 2560R3 can only source a limited amount, and this would be the most conservative approach.

The reason we chose the use of transparent concrete was because it would allow us to readily see the LEDs through it in order to display our image. Another advantage to this use is for the commercial applications that this design brings to the architectural field. It is a purely aesthetic application; however, it has a great monetary value in industry.

Future Work:

Printed Circuit Board (PCB) implementation is the most immediate route that we plan on taking. This would be done through the use of Eagle, PCBexpress, or OrCad. The reason this would be great is because the entire design becomes much easier board to prototype on, as well as the presentation factor is higher.

The PCB in conjunction with a 3D printed design to fabricate the LED array on will be the cleanest implementation for prototyping in the long run. Additionally, if the PCB is designed to be a full testing device, there will be room for expansion into other test areas that will end up saving money in the long run on time spent to implement.

Further work past the prototyping of the PCB would to design the Application Specific Integrated Circuit (ASIC) that would be able to implement the design as well as write the VHDL that would model the architecture.

Source code:

*Matlab Script***:**

Below is the main script to do image decryption for byte wise operation. Variable input path is taken from the Labview VIEW path controller, this variable signified the location of the image. The variable level is taken from the VI as a number from 0-1.00 which corresponds to the black and white cutoff level, this variable is used with the im2bw command which turns an image to black and white. The variable b is an output from the script which is an array of 400 bits corresponding to each LED row wise. The variable output is the same array converted into 50 bytes representing pairs of 8 LEDs.

Note:

VI-Input/output dictates that this variable is designated as input or output within labview

```
inputpath=VI-Input
level=VI-Input
image=imread(inputpath,'jpeg');
```

```
imresized=imresize(image,[20,20]);
BW=im2bw(imresized,level);
b=double(reshape(BW.',[],1));  %%VI-Output
imshow(imresized);
imshow(BW);
output=[];
for i=1:8:400
        bit=b(i:i+7,1);
        byte=sum(bit.*[128;64;32;16;8;4;2;1]);
        output=[output;byte];

end
output   %%VI-Output
```

Arduino Code:

```
#include <EEPROM.h>                    // include libraries that are necessary

byte pin_arr[41];                      // define transmit pin array
byte inStream;                         // for incoming serial data
int index;                             // index into array; where we store the char val.
byte inVal[53];                        // array of input values.
int led_arr[401];                      // LED high or low
int state;                             // state counter

byte mask;                             // mask variable that will be used for shift-reg comparison
byte bitDelay;                         // delay for cycle on shift register etc to complete tasks
int mask_cnt;                          // mask counter that allows for us to fully populate the LED_arr
int photo_resistor_pin;                // analog_pin is the pin that will recieve the analog data from
the photoresistor
int digital_switch_pin;                // analog_switch that if high means we are in automatic mode
and use the photo-resistor

void setup() {
  Serial.begin(115200);   // opens serial port, sets data rate to 115200 bps

  index = 0;                           // initialize to 0
  inStream = 0;                        // initialization val is by default low
  state = 0;                           // initialization of the data_valid flag for the eeprom to low
  mask = 1;                            // mask is set = 00000001, binary
  bitDelay = 0.6;                      // 100 microsecond delay
  mask_cnt = 0;                        // mask counter that indexes the 400 sized led_arr
```

```
  photo_resistor_pin = 0;               // analog_pin is assigned 0.
  digital_switch_pin = 52;              // digital_switch_pin is assignged analog pin 1

  // setup pin value for transmit in the pin_arr
  for(int i = 1; i < 40; i++){
     //skip pins 18,19
     if(i+1<18)
       pin_arr[i] = i+1;
     else
       pin_arr[i] = i+3;
  }
  pin_arr[0]=50; //necessary for miswired mosfet


  //Print out pins used, for debug purposes
  for(int i = 0; i < 40; i++){
     Serial.print("Pin ");
     Serial.println(pin_arr[i]);
  }
  // setup the pin_arr to all be output pins
  for(int i = 0; i < 40; i++){
     pinMode(pin_arr[i], OUTPUT);
     digitalWrite(pin_arr[i],HIGH); //Turn off all LEDs
  }
  Serial1.begin(9600);                  // opens Serial port, sets data rate to 115200 bps
  //eeprom_clear();
}


// main loop for getting stuff done.
void loop() {
    state_machine();                    // goes to statemachine to control the program.
}

// controller function that drives the operational state machine
void state_machine(){

  switch(state){
    case 0:
      if(is_switch_toggled() && !is_automatic_photo_resistor()){
        // nop
        Serial.println(read_photo_resistor(), DEC);
        Serial.println(digitalRead(digital_switch_pin), DEC);
        delay(1000);
```

```
    }
    else
      ++state;
    break;
  case 1:
    //Serial1.println("Waiting for input");
    // we take in the first 52 bytes. 2 stop bytes + 50 valid bytes of data.
    if (Serial.available() > 0) {
      inStream = Serial.read();
      if(index>=2) //ignore bad bits
        inVal[index-2] = inStream;

      Serial1.print("index: ");
      Serial1.print(++index, DEC);
      Serial1.print("; ");
      Serial1.println(inStream, BIN);
      if(index == 52)
        ++state;                        // move to next state.
    }
    break;
  case 2:
    // split the byte data into bits
    for(int i = 0; i<50; i++){
      splitByte(inVal[i]);
    }

    Serial1.println("Display:");
    for(int i = 0, j=0; i<400; i++, j++){

      //Serial1.print("LED: ");
      //Serial1.print(i, DEC);
      //Serial1.print("; ");
      Serial1.print(led_arr[i], DEC);
      if(i%19==0 && i>0)
        Serial1.println();
//      eeprom_write(led_arr[i], j);
    }
    mask_cnt = 0;                       // reset counter so that we can have it auto update.
    index    = 0;                       // reset index so we can accept new data.
    ++state;                            // advance to next state
    break;
  case 3:
//    eeprom_display();
    led_display();
```

```
      break;
    default:                        // nop
      break;
  }
}


// check to see if the analog switch is pressed. if it is high return 1 and we run the photo-resistor
// if low, then we return 0 and run without checking photo-resistor
int is_switch_toggled(){
  if(digitalRead(digital_switch_pin))
    return 1;                       // switch is high
  else
    return 0;                       // switch is low
}

// Threshold value 560 is a pretty dark shadow over the resistor
// 800++ is a bright amount of light
int is_automatic_photo_resistor(){
  if (read_photo_resistor()<500)
    return 1;                       // it is dark and should display the LED array
  else
    return 0;                       // it is too bright to run the LED display
}

// read in the anaolog value of the photoresistor
int read_photo_resistor(){
  return analogRead(photo_resistor_pin);
}

// LED display activation of pins for row column display.
// The Mosfets use inverted logic, so driving the pin high turns it off.
void led_display(){
  int row = 0, col = 0;
  for(int cnt = 0; cnt < 400; cnt++){
    row = (int)(cnt/20);
    col = (int)(cnt%20);

    delay(bitDelay);            //delay

    if(led_arr[cnt]){
      //Turn on the LED
      digitalWrite(pin_arr[row],LOW);
      digitalWrite(pin_arr[col+20],LOW);
```

```
      // DELAY before going "high" again (inverter logic)
      delay(bitDelay);           //delay
      digitalWrite(pin_arr[row],HIGH);
      digitalWrite(pin_arr[col+20],HIGH);

    }
    else{
      //Turn off the LED
      digitalWrite(pin_arr[row],HIGH);
      digitalWrite(pin_arr[col+20],HIGH);
    }
  }
}

void splitByte(byte in){
  for(mask = 00000001; mask>0; mask<<=1){
    if (in & mask){                    // if bitwise AND resolves to true
      led_arr[mask_cnt++] = 1;             // save 1
    }
    else{                              //if bitwise and resolves to false
      led_arr[mask_cnt++] = 0;             // save 0
    }
  }
}

// clears the eeprom down to 0
void eeprom_clear(){
  // write a 0 to all 512 bytes of the EEPROM
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, 0);
}

// returns 1 when the data write is complete
int eeprom_write(int data, int ind){
    if(ind < 400)
      EEPROM.write(ind, data);

    return 1;
}

// read from eeprom based on address parameter.
int eeprom_read(int address){
    return EEPROM.read(address);
```

```
}

void eeprom_display(){
  for(int i = 0; i < 512; i++){
      Serial1.println(eeprom_read(i));
      if(i == 400)
        Serial1.println("Quit Looking 400 is ^^");
  }
  Serial1.println();
}
```

Citations:

[1] " Serial Available()," *Arduino*, Vol. , no. , pp. , .[]. :http://arduino.cc/en/Serial/Available. [Accessed 17 December 2012]