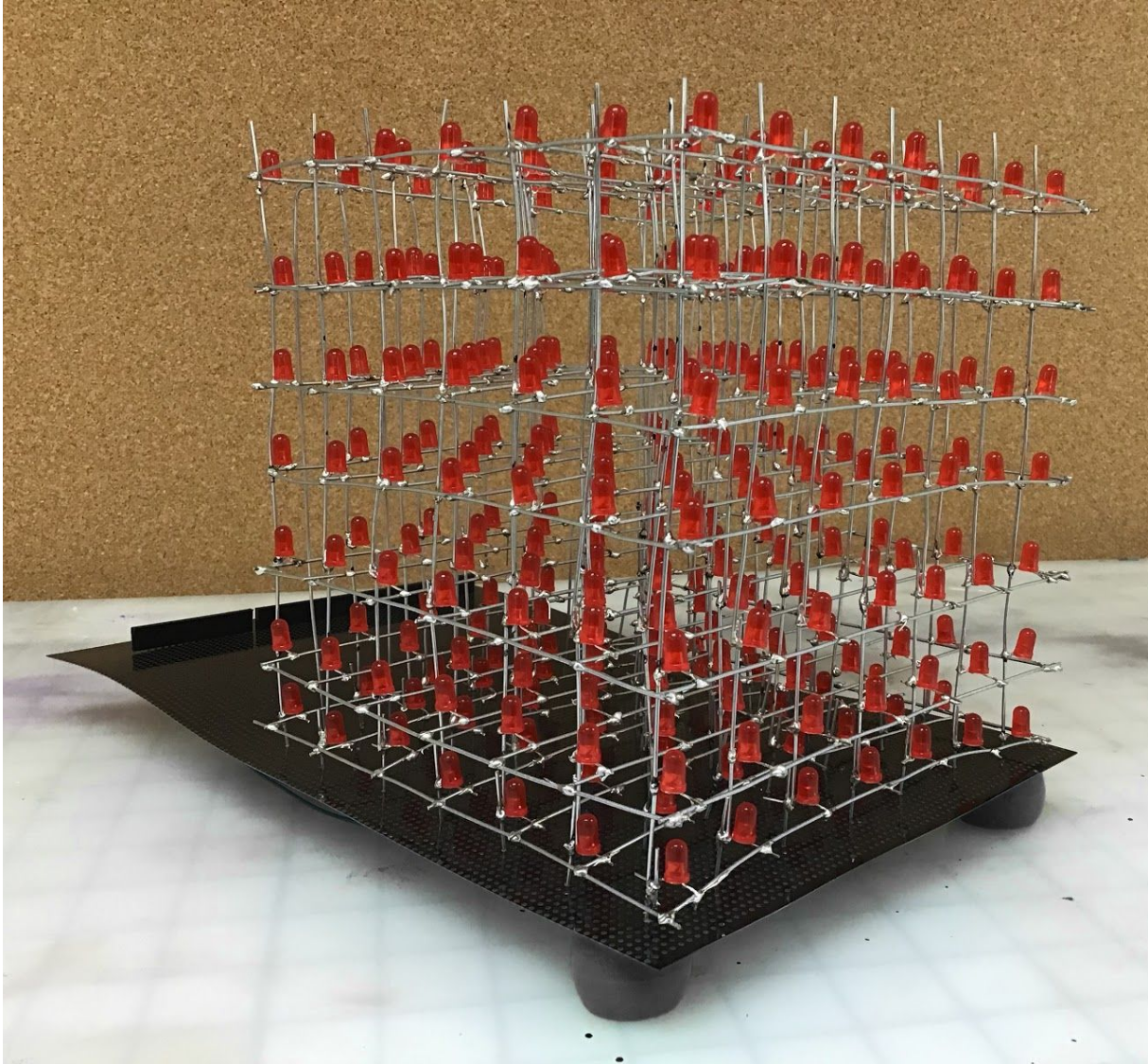


# 3D Etch-a-Sketch - Final Report



Alex Smith

Sean Hoerger

## **Abstract**

One of the most common DIY projects done with LEDs is the creation of an three-dimensional LED matrix. Typically these projects use the matrix for the purpose of displaying various animations, however we were more interested in allowing a user to interact with the matrix in some way. To that end, we wanted to create a three-dimensional LED Matrix which a user could use as an Etch-a-Sketch. To do this we use the Cyclone IV FPGA to turn on LEDs in the matrix via a time multiplexing scheme, and the ATSAM4S4B Microcontroller to read user inputs from two joysticks and a reset button. The FPGA and microcontroller are then connected via an SPI link where the Microcontroller reads user inputs, updates the desired LEDs to be turned on by updating a char array, and then sends the signal to the FPGA which updates the matrix appropriately.

## **Introduction**

This goal of this project was to create a three-dimensional LED matrix which functioned as a three-dimensional Etch-a-Sketch. We thought the project would be a fun use of LEDs and would allow for a wide variety of possible designs to be created on the LED matrix we created. Additionally, a functioning LED Matrix which is controlled by a separate unit (in this case the microcontroller) and is able to have any LED turn on at any time has many applications for various games and activities other than an Etch-a-Sketch. A snake game, a game to avoid falling obstacles by controlling a ‘player’ LED, or any number of others are implementable through the same scheme as the Etch-a-Sketch through some non-trivial but not exceptionally difficult changes in the software on the microcontroller.

The project as a whole consists of a Cyclone IV FPGA which is responsible for sending the signals to the LED matrix turning on the appropriate LEDs based on a 343 bit one hot encoded signal stored in RAM. To turn on these LEDs with the limited number of output pins available to the FPGA the LEDs are lit using a time multiplexing scheme. To do this the FPGA sends the same signal seven-bit signal to seven flops and then inverts each signal after the flop, and only activates one flop at a time, continually cycling through these flops and also through seven pins which power the layers of the LED matrix (more on this in the section on matrix design and FPGA design). The other major component of the system is the ATSAM4S4B microcontroller which is responsible for reading user inputs from the two joysticks and reset pushbutton, and then for determining which new LED should be turned on. That is then encoded in a char array and sent to the FPGA over an SPI link (the microcontroller being the master and the FPGA the slave). In this way the system is able to take a user input and change the LEDs which are lit on the LED matrix, allowing for the desired functionality. A block diagram of the system can be seen below.

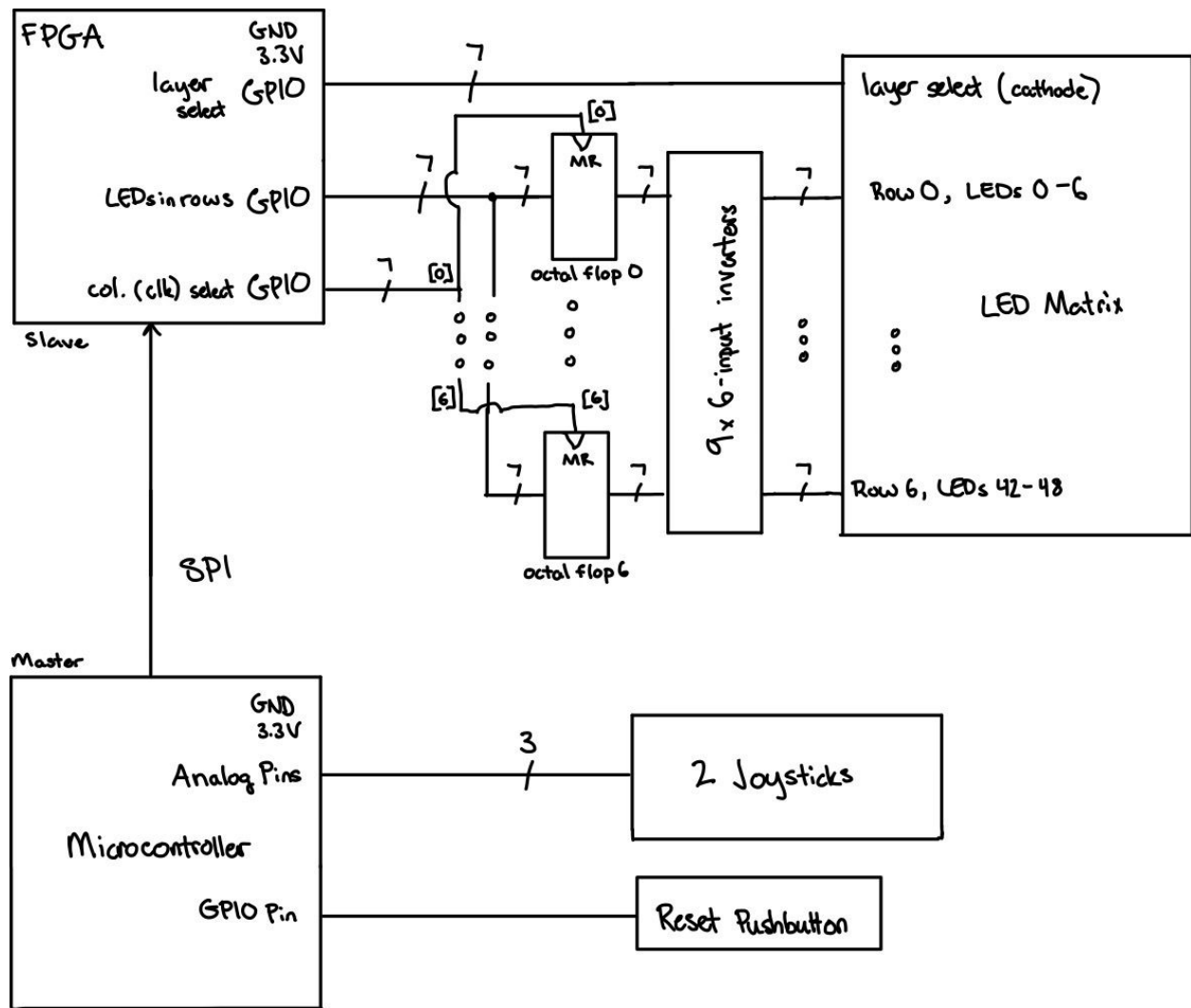
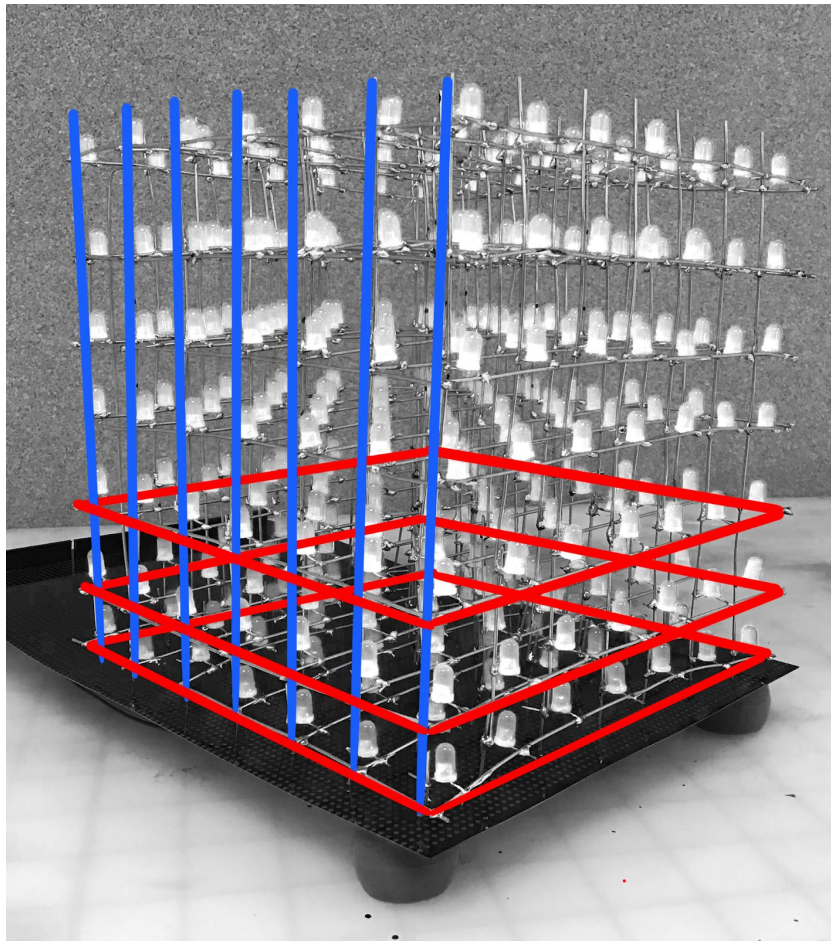


Figure 1: System Block Diagram

## New Hardware

This project entailed two new pieces of hardware. The simpler hardware is the two joysticks which allow the user to input any of six possible directions (forward, backward, left, right, up, down). The joysticks function by producing an analog signal proportional to the location of the joystick with respect to a two axis system. The joystick is also able to be pressed, acting as a switch although we did not utilize this functionality as the joystick buttons are low quality and often fail. To determine the direction of the joystick it is necessary to read the two analog output pins corresponding to the x and y axis, the signal ranges from the supply voltage to ground at the extreme of each axis and sits at half the supply voltage when at rest in the center of the pad. In this way the position of the joystick can be determined by reading the voltage of each axis at any given time.

The second new piece of hardware was the 7x7x7 LED Matrix. One of the largest challenges of our project was simply building our LED Matrix. Our design is a 7x7x7 LED matrix. Essentially, the matrix consists of seven layers, powered individually. Each layer is a 7x7 grid of LEDs, each sharing a common cathode. To connect the layers, forty-nine vertical wires provide common anodes to the vertical columns of LEDs. Thus, to turn on an LED, the desired layer is powered, and the desired column is pushed low. The remaining columns are pushed high. Thus, current only flows through the desired LED, creating light. An image of this can be seen below, seven of the vertical wires are highlighted in blue and three layers are highlighted in red.



*Figure 2: Constructed LED Matrix*

Building the LED matrix proved to be quite a challenge; soldering 343 LEDs together is no quick task. To expedite the assembly process, we built two molds out of wood. The first mold holds an LED while we bend its anode and cathode wires into the correct shape. The anode was bent into a loop to fit around the vertical wire. The second mold held forty-nine LEDs in the correct spacing for a layer. We then soldered seven LEDs in a row to a row wire, and repeated

six more times. Once we had all the rows, we could then add two end wires to complete the layer.

Once we had completed all seven layers, we could begin connecting the layers. We started by soldering the forty-nine vertical wires to the bottom layer. Then, the idea was slide each layer over the top, using the loops on each LED. However, the loops were exactly the size of the wire, thus there was no room for error. Adding to the difficulty, all forty-nine LEDs had to be lined up at once. Ultimately, we elected not to use the loops as originally designed, and we had to piece each vertical column together.

After the cube had been fully assembled, we soldered it to the protoboard (which also turned out to be quite tricky to line up all the holes). We then added the additional wires to power each layer and column and the header pins to make the wiring easier. Underneath the protoboard, wires were added to connect the layer select pins to the layer select wire and to connect the column select pins to the column wires.

### **Schematics**

The schematics are organized into modules for ease of understanding as seen in the following pages.

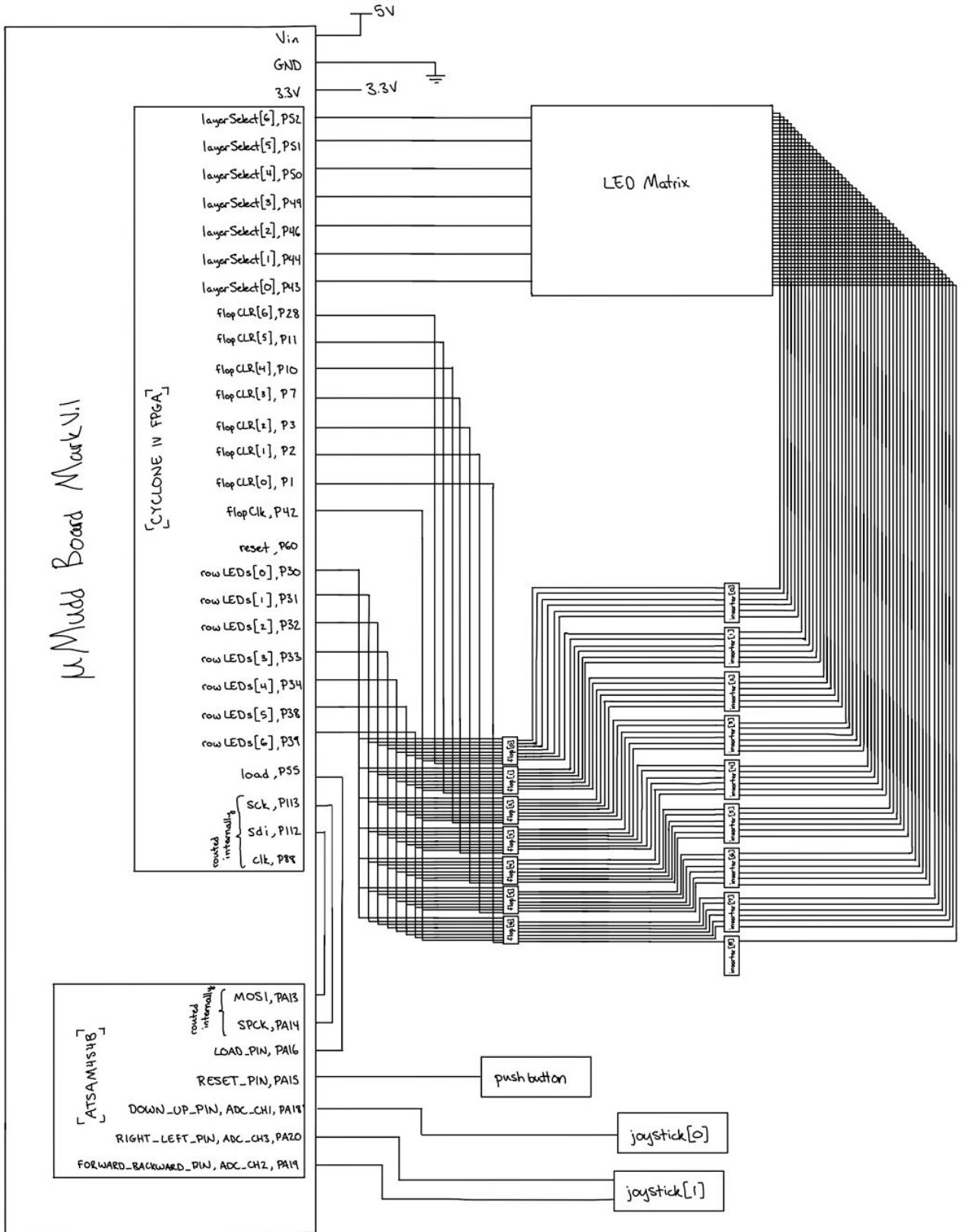


Figure 3: Overall System Schematic

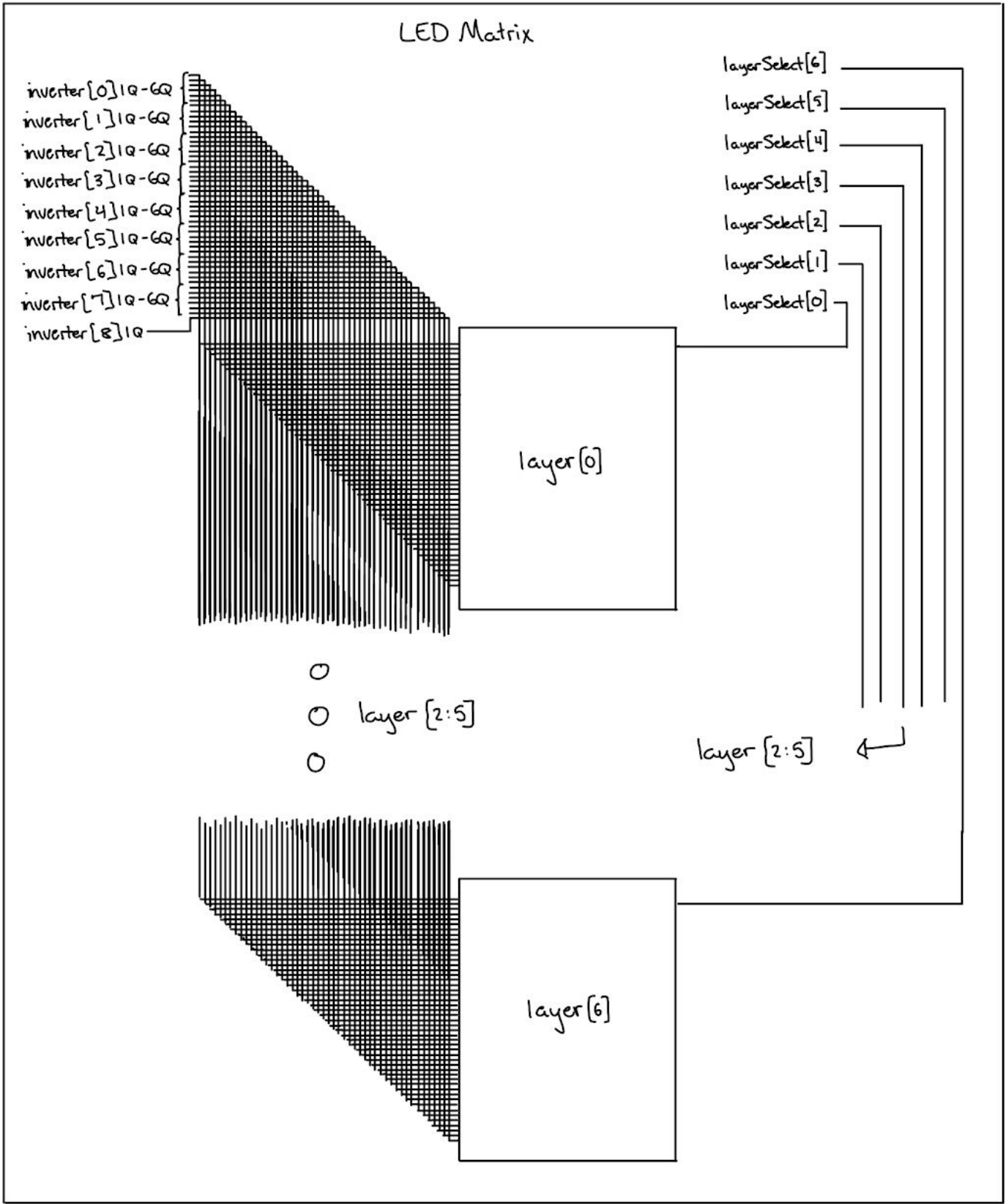


Figure 4: LED Matrix Module

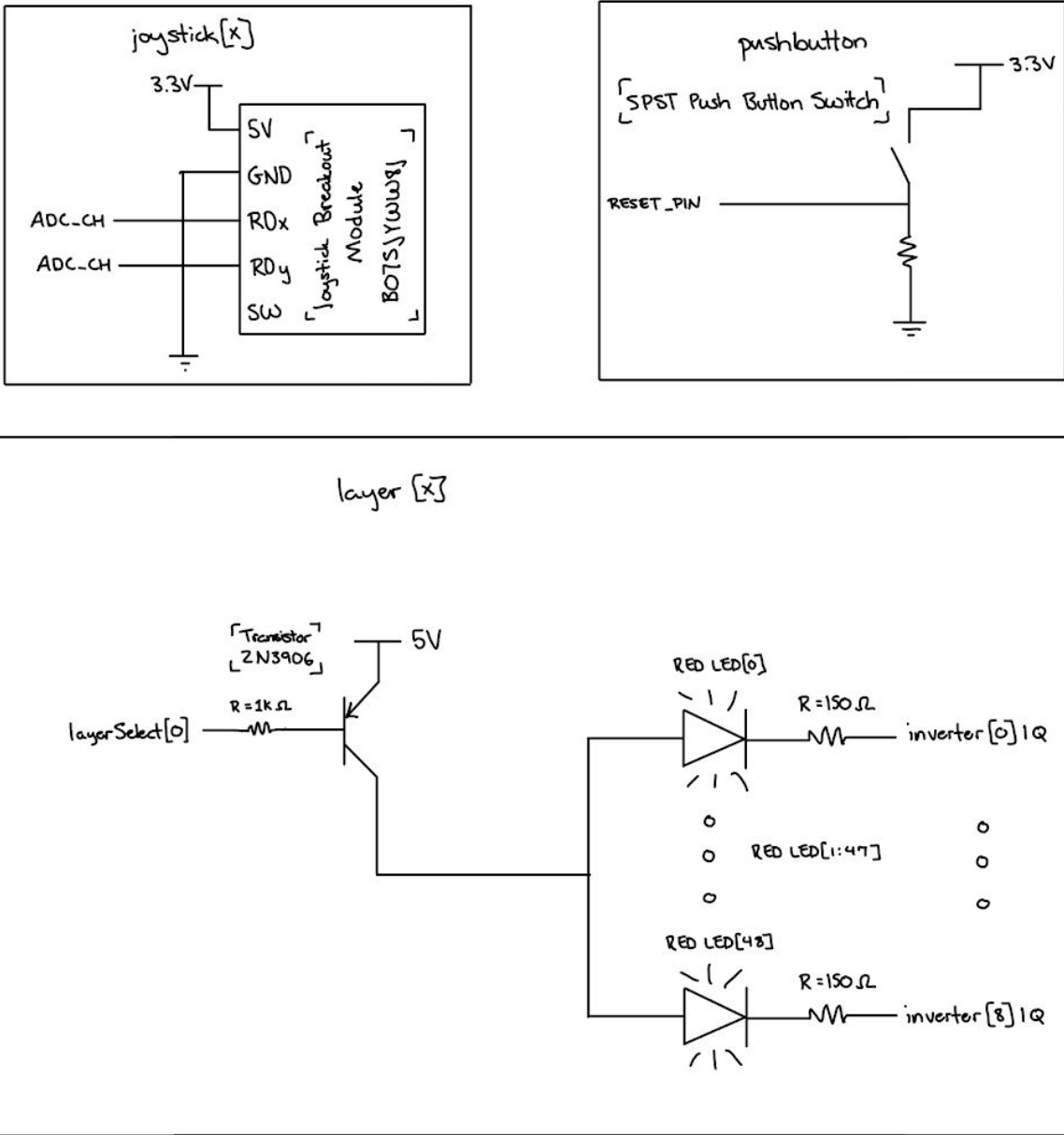


Figure 5: Joystick, Pushbutton and layer Modules

### Microcontroller Design

For our design, the microcontroller was tasked with reading user inputs, managing the board state (which LEDs are on and off), and sending the board state to the FPGA through SPI. The microcontroller keeps track of three important pieces of information: the board state, the location of the cursor (the location of the user), and whether or not the reset button has been pressed. The board state is a character array of size 49, one character for each row. The seven least significant bits of each character correspond to the seven LEDs in a given row, a binary one indicating a lit LED and a binary zero indicating an off LED. The cursor information is stored as



three integers representing the x, y, and z coordinates of the user. The reset trigger is saved as a boolean integer.

Users provide input through two joysticks. These joysticks provide an analog signal to the microcontroller that correspond to the position of the joystick. The microcontroller, reads the input using its analog peripheral. Depending on the reading, the location of the cursor is appropriately incremented or decremented in the along x, y, or z axis. There are software checks to make sure the cursor does not leave the domain of the matrix; if the user hits the edge of the matrix, the cursor cannot continue in that direction.

The etch-a-sketch utilized a few key functions to run properly:

- `getMeasurement()` - reads user input and updates the cursor location, signals if the reset button is pressed
- `turnOn(int x, int y, int z)` - turns on the LED at the given x, y, and z coordinates
- `turnOff(int x, int y, int z)` - turns off the LED at the given x, y, and z coordinates
- `spiSendBoard()` - sends the board state over SPI

The microcontroller is also tasked with controlling animations. Using the `turnOn` and `turnOff` methods in conjunction with a variety of loops, delays, hard coding strategies, creating fun animations is relatively simple. We created three animations to go along with our etch-a-sketch.

- Rainforest - random LEDs at the top of the display turn on one-by-one. Upon an LED turning on, the rest of the LEDs in the column turn on in descending order. Once a critical number (21) of columns have been lit, the animation alternates between a new column being lit and already lit column being turned off. Columns are turned off in the same fashion as they are turned on, starting at the top and descending to the bottom.
- Space Mountain - seven random LEDs on the right face of the matrices display are turned on at once. They streak quickly across the matrix in a line from right to left, and this process is repeated with different, random LEDs in each cycle. The final effect is similar to looking at moving lights out of a subway window or taking a ride on Disney's Space Mountain roller coaster.
- Wave Peak - consists of an oscillating cone. The appearance is similar to that of a speaker playing a low bass note.

## **FPGA Design**

The FPGA design mainly involved setting up a system which is able to time multiplex through the forty-nine rows of seven LEDs, in addition to storing a 343 bit signal which encodes the LEDs that should be on in the matrix in RAM so that the microcontroller can only send a SPI

communication packet when a joystick input is received rather than sending the signal at the desired multiplexing speed. A block diagram of the FPGA modules can be seen below.

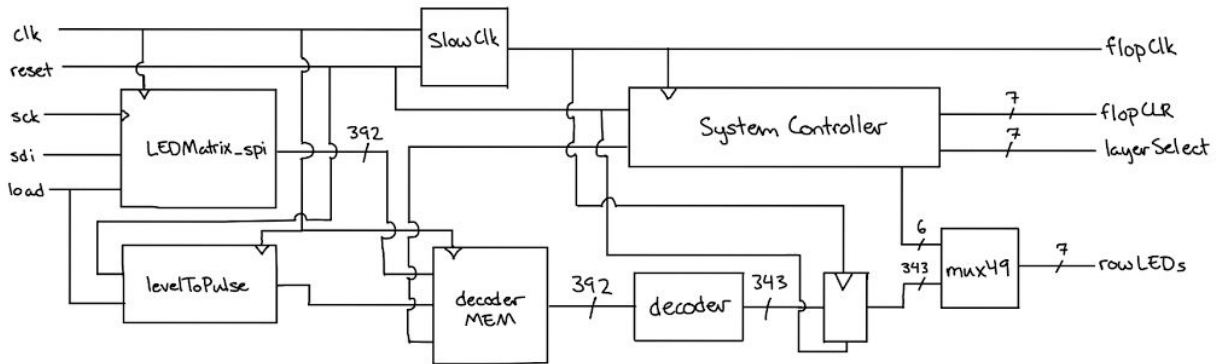


Figure 6: Block Diagram of FPGA System

An outline of each module is included below:

- **SlowClk**: this module takes the 40 MHz FPGA clock signal and turns it into a 6,666 Hz clock signal.
- **LEDMatrix\_spi**: this module implements a simple SPI communication system as the slave where it is only able to receive data from the master.
- **levelToPulse**: this module implements a Level to Pulse converter for use in determining when the SPI packet being read from the microcontroller is ready to be pushed into RAM.
- **decoderMem**: this module takes the 392 bit signal encoded in the microcontroller as a char array with 49 elements and stores it in the RAM available to the FPGA.
- **decoder**: this module takes in the 392 bit signal encoded in RAM and pulls out the bits which correspond to the 343 LEDs in the matrix (by a one-hot encoding).
- **mux49**: this module is a 49 input mux which parses the 343 bit bus encoding the LEDs which should be turned on and selects the one to be pushed to the 7 flops at any given time based on commands from the System Controller.
- **systemController**: this module implements the multiplexing necessary to turn on all LEDs. It does this by having each row in the layers correspond to a single state and by sending out the appropriate flopCLR and layerSelect signal while also sending the appropriate command to the mux to select the 7 bits to be pushed to the flops.

## Results

We are quite proud of our final project. We were able to fully implement both the etch-a-sketch and multiple animations in addition to building the full 7x7x7 LED Matrix and controller with joysticks. Playing with the etch-a-sketch proves to be very fun and satisfying. The animations are equally enjoyable and mesmerizing.

However, our project was not free of bugs nor learning opportunities. Building the LED matrix proved to be a massive undertaking; with every completed step, we realized there was more to do than we initially planned. Furthermore, we struggled with phantom lights turning on

in the bottom layer of LEDs and mysterious timing discrepancies with the communication between the microcontroller and the FPGA. With infinite time these bugs could likely be squashed however we are extremely satisfied with the result of the project as we were able to create a functional Etch-a-Sketch and some interesting animations as well.

## References

HMC Student Sarp Misoglu completed a similar project involving an LED matrix in 2017. We referenced Misoglu’s final project report for inspiration on solving various technical challenges with regards to the matrix design and software.

We also referenced the YouTube video, “How to Make an LED Cube at Home” by the channel Tapendra Mandal. It gave us some very useful tips for assembling our LED matrix. The video can be found at this link: <https://www.youtube.com/watch?v=2BIVUKW5hL0&t=238s>

## Data Sheets:

- 2N3906 PNP Transistor: <http://pages.hmc.edu/harris/class/e155/2N3906.pdf>
- 7404 Hex Inverter: <http://www.ti.com/lit/ds/symlink/sn74ls04.pdf>
- SN74LS273N Octal Flop: <http://www.ti.com/lit/ds/symlink/sn74ls273.pdf>

## Bill of Materials

Item	Price	Link
343+ LEDs	\$15.96	<a href="https://www.amazon.com/stores/page/015D8455-927B-43B2-81ED-2514F8D2C0F6?ingress=2&amp;visitId=129eb292-229d-47e8-a31b-ff578aa3d362&amp;ref_=bl_dp_s_web_18685144011">https://www.amazon.com/stores/page/015D8455-927B-43B2-81ED-2514F8D2C0F6?ingress=2&amp;visitId=129eb292-229d-47e8-a31b-ff578aa3d362&amp;ref_=bl_dp_s_web_18685144011</a>
Galvanized Steel Wire	\$5.99	Available at Lowes
Protoboard Base	\$11.96	<a href="https://www.amazon.com/gp/product/B07DH7HWFT/ref=ox_sc_act_title_1?smid=A1THAZDOWP300U&amp;psc=1">https://www.amazon.com/gp/product/B07DH7HWFT/ref=ox_sc_act_title_1?smid=A1THAZDOWP300U&amp;psc=1</a>
Joysticks	\$9.99	<a href="https://www.amazon.com/gp/product/B07SJYWW8J/ref=ox_sc_act_title_2?smid=A2760MNFACRLA8&amp;psc=1">https://www.amazon.com/gp/product/B07SJYWW8J/ref=ox_sc_act_title_2?smid=A2760MNFACRLA8&amp;psc=1</a>
Octal Flops	\$0.85 x 7 = \$5.95	<a href="https://www.jameco.com/Jameco/Products/ProdDS/47386.pdf">https://www.jameco.com/Jameco/Products/ProdDS/47386.pdf</a>
<b>Total</b>	<b>\$49.85</b>	

## Software & Verilog

Attached as follows is the software used to control the ATSAM4S4B and the verilog used to imply logic on the Cyclone IV FPGA.

## Software

```
// joystick.c by Sean Hoerger
// shoerger@hmc.edu

#include <stdint.h>
#include <time.h>
#include "SAM4S4B.h"
#include "SAM4S4B_sys.h"
#include "SAM4S4B_pio.h"
#include "SAM4S4B_spi.h"

// dimensions of the cube
#define ROW_SIZE 7
#define BOARD_SIZE 49

// SAM pins
#define LOAD_PIN PIO_PA16
#define RESET_PIN PIO_PA15
#define DOWN_UP_PIN ADC_CH1
#define RIGHT_LEFT_PIN ADC_CH3
#define FORWARD_BACKWARD_PIN ADC_CH2
#define MODE_PIN PIO_PA10

// stores board state
// each row stored as the least
// significant 7 bits of a char
char board[BOARD_SIZE];

// the coordinates of the cursor
int xPos;
int yPos;
int zPos;

// reset trigger
int reset;

// gets user input
// - 0 if no input
// - 1 if directional input
// - 2 if reset
int getMeasurement() {
    // test for reset
    if(pioDigitalRead(RESET_PIN) == PIO_HIGH) {
        return 2; // reset trigger
    }

    // get directional input from user
    if(adcRead(RIGHT_LEFT_PIN) > 2.7) // move right
        --xPos;
    else if(adcRead(RIGHT_LEFT_PIN) < .3) // move left
        ++xPos;
    else if(adcRead(FORWARD_BACKWARD_PIN) < .3) // move back
        --yPos;
    else if(adcRead(FORWARD_BACKWARD_PIN) > 2.7) // move forward
```

```

        ++yPos;
    else if(adcRead(DOWN_UP_PIN) > 2.7) // move up
        --zPos;
    else if(adcRead(DOWN_UP_PIN) < .3) // move down
        ++zPos;
        else return 0; // return 0 if no user input

    // check boundary cases
    if(xPos < 0) xPos = 0;
    if(xPos > 6) xPos = 6;
    if(yPos < 0) yPos = 0;
    if(yPos > 6) yPos = 6;
    if(zPos < 0) zPos = 0;
    if(zPos > 5) zPos = 5; // keep bottom layer from being accessed

    return 1; // return 1 if directional input

}

// turns off all LEDs on the board
void clearBoard() {
    // set all chars in board state to 0000_0000, turning off all
lights
    for(int i = 0; i < BOARD_SIZE; ++i) {
        board[i] = 0x00;
    }
}

// turns on all LEDs on the board
void fillBoard() {
    // set all chars in board state to 0111_1111, turning on all lights
    for(int i = 0; i < BOARD_SIZE; ++i) {
        board[i] = 0x7F;
    }
}

// turns on the LED at the specified location
void turnOn(int x, int y, int z) {
    // account for board shift
    ++z;
    if(z == BOARD_SIZE) z = 0;
    // turn on the specified bit
    board[7*y + z] |= 1 << x;
}

// turns off the LED at the specified location
void turnOff(int x, int y, int z) {
    // account for the board shift
    ++z;
    if(z == BOARD_SIZE) z = 0;
    // turn off the specified bit
    board[7*y + z] &= ~(1 << x);
}

```

```

// sends the board state over spi
void spiSendBoard() {
    pioDigitalWrite(LOAD_PIN, 1);
    // send each character over spi
    for (int i = 0; i < BOARD_SIZE; ++i){
        spiSendReceive(board[i]);
    }
    pioDigitalWrite(LOAD_PIN, 0);
}

int main(void) {

    // set up SAM
    samInit();
    pioInit();
    tcDelayInit();
    adcInit(ADC_MR_LOWRES_BITS_10);
    adcChannelInit(FORWARD_BACKWARD_PIN, ADC_CGR_GAIN_X2, ADC_COR_OFFSET_ON);
// the x direction of joystick 1 --> forward/back
    adcChannelInit(RIGHT_LEFT_PIN, ADC_CGR_GAIN_X2, ADC_COR_OFFSET_ON); // the
y direction of joystick 1 --> right/left
    adcChannelInit(DOWN_UP_PIN, ADC_CGR_GAIN_X2, ADC_COR_OFFSET_ON); // the x
direction of joystick 2 --> up/down
    spiInit(MCK_FREQ/244000, 0, 1);

    // set up pins
    pioPinMode(LOAD_PIN, PIO_OUTPUT);

    int input;

    // main etch-a-sketch loop
    while(1) {
        // clear the board and reset the cursor to the starting position
        clearBoard();
        spiSendBoard();
        tcDelayMillis(100);
        xPos = 3;
        yPos = 3;
        zPos = 3;
        turnOn(xPos, yPos, zPos);
        spiSendBoard();
        tcDelayMillis(100);
        reset = 0;

        while(!reset) {
            input = getMeasurement();
            // get measurement
            if(input == 2) {
                // reset
                reset = 1;
            }
            else {
                // update board state to reflect user input

```

```

        turnOn(xPos, yPos, zPos);
        spiSendBoard();
        tcDelayMillis(200);
    }
}
}
}

```

## Animations:

```

// animations.c by Sean Hoerger
// shoerger@hmc.edu

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "SAM4S4B.h"
#include "SAM4S4B_sys.h"
#include "SAM4S4B_pio.h"
#include "SAM4S4B_spi.h"

#define ROW_SIZE 7
#define BOARD_SIZE 49
#define LOAD_PIN PIO_PA16
#define RESET_PIN PIO_PA15

char board[BOARD_SIZE];

// turns of all LEDs on the board
void clearBoard() {
    for(int i = 0; i < BOARD_SIZE; ++i) {
        board[i] = 0x00;
    }
}

// turns on all LEDs on the board
void fillBoard() {
    for(int i = 0; i < BOARD_SIZE; ++i) {
        board[i] = 0x7F;
    }
}

// turns on the LED at the specified location
void turnOn(int x, int y, int z) {
    ++z;
    if(z == BOARD_SIZE) z = 0;
    board[7*y + z] |= 1 << x;
}

```

```

// turns off the LED at the specified location
void turnOff(int x, int y, int z) {
    ++z;
    if(z == BOARD_SIZE) z = 0;
    board[7*y + z] &= ~(1 << x);
}

// sends the board state over spi
void spiSendBoard() {
    pioDigitalWrite(LOAD_PIN, 1);
    for (int i = 0; i < BOARD_SIZE; ++i){
        spiSendReceive(board[i]);
    }
    pioDigitalWrite(LOAD_PIN, 0);
}

int main(void) {

    // set up SAM
    samInit();
    pioInit();
    tcDelayInit();
    spiInit(MCK_FREQ/244000, 0, 1);

    // set up pins
    pioPinMode(LOAD_PIN, PIO_OUTPUT);

    int counter = 0;

    while(1) {
        //////////////////////////////////////
        // RAINFOREST
        //////////////////////////////////////
        // turns on and off random columns
        for (int y = 0; y < ROW_SIZE; ++y) {
            int x = rand() % 7;
            for (int z = 0; z < 6; ++z){
                turnOn(x, y, z);
                spiSendBoard();
                tcDelayMillis(100);
            }
            ++counter;
        }
        if (counter > 21) {
            while(1) {
                for (int y = 0; y < ROW_SIZE; ++y) {
                    int x = rand() % 7;
                    for (int z = 0; z < 6; ++z){
                        turnOn(x, y, z);
                        spiSendBoard();
                        tcDelayMillis(100);
                    }
                    x = rand() % 7;
                }
            }
        }
    }
}

```



```

        for (int z = 0; z < 6; ++z){
            turnOff(x, y, z);
            spiSendBoard();
            tcDelayMillis(100);
        }
    }

}

////////////////////////////////////////////////////
// SPACE MOUNTAIN
////////////////////////////////////////////////////
// lights go zoom zoom!
//
int lights[ROW_SIZE-1];
//
//
// for (int i = 0; i < ROW_SIZE - 1; ++i) {
//     int y = rand() % 7;
//     lights[i] = y;
// }
//
// spiSendBoard();
// tcDelayMillis(100);
// for (int x = 0; x < ROW_SIZE; ++x) {
//     for (int z = 0; z < ROW_SIZE -1; ++z) {
//         turnOn(x, lights[z], z);
//     }
//     spiSendBoard();
//     tcDelayMillis(35);
// }
// for (int x = 0; x < ROW_SIZE; ++x) {
//     for (int z = 0; z < ROW_SIZE -1; ++z) {
//         turnOff(x, lights[z], z);
//     }
//     spiSendBoard();
//     tcDelayMillis(35);
// }
//
////////////////////////////////////////////////////
// Peak Wave
////////////////////////////////////////////////////
// central wave oscillation action

//
int delayTime = 38;
//
//
// // step 0
// for (int x = 0; x < ROW_SIZE; ++x) {
//     for (int y = 0; y < ROW_SIZE; ++y) {
//         turnOn(x, y, 3);
//     }
// }
// spiSendBoard();
// tcDelayMillis(delayTime);
//
//
// // step 1

```

```

//      turnOn(3, 3, 2);
//      turnOff(3, 3, 3);
//      spiSendBoard();
//      tcDelayMillis(delayTime);
//
//      // step 1
//      turnOn(3, 3, 1);
//      for (int x = 2; x < 5; x++) {
//          for(int y = 2; y < 5; ++y) {
//              if(((x == 2) || (x == 4)) || ((y == 2) || (y == 4))) {
//                  turnOn(x,y,2);
//                  turnOff(x,y,3);
//              }
//          }
//      }
//      turnOff(3, 3, 2);
//      spiSendBoard();
//      tcDelayMillis(delayTime);
//
//      // step 2
//      turnOn(3, 3, 0);
//      for (int x = 2; x < 5; x++) {
//          for(int y = 2; y < 5; ++y) {
//              if(((x == 2) || (x == 4)) || ((y == 2) || (y == 4))) {
//                  turnOn(x,y,1);
//                  turnOff(x,y,2);
//              }
//          }
//      }
//      for (int x = 1; x < 6; x++) {
//          for(int y = 1; y < 6; ++y) {
//              if(((x == 1) || (x == 5)) || ((y == 1) || (y == 5))) {
//                  turnOn(x,y,2);
//                  turnOff(x,y,3);
//              }
//          }
//      }
//      turnOff(3, 3, 1);
//      spiSendBoard();
//      tcDelayMillis(delayTime);
//
//      while (1) {
//          // step 3
//          turnOn(3, 3, 1);
//          turnOff(3, 3, 0);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//
//          // step 4
//          turnOn(3, 3, 2);
//          turnOff(3, 3, 1);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//

```

```

//          // step 5
//          turnOn(3, 3, 3);
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if(((x == 2) || (x == 4)) || ((y == 2) || (y ==
4)))) {
//                      turnOn(x,y,2);
//                      turnOff(x,y,1);
//                  }
//              }
//          }
//          turnOff(3, 3, 2);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//          // step 6
//          turnOn(3, 3, 4);
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if(((x == 2) || (x == 4)) || ((y == 2) ||
(y == 4)))) {
//                      turnOn(x,y,3);
//                      turnOff(x,y,2);
//                  }
//              }
//          }
//          turnOff(3, 3, 3);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//          // step 7
//          turnOn(3, 3, 5);
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if(((x == 2) || (x == 4)) || ((y == 2) ||
(y == 4)))) {
//                      turnOn(x,y,4);
//                      turnOff(x,y,3);
//                  }
//              }
//          }
//          for (int x = 1; x < 6; x++) {
//              for(int y = 1; y < 6; ++y) {
//                  if(((x == 1) || (x == 5)) || ((y == 1) ||
(y == 5)))) {
//                      turnOn(x,y,3);
//                      turnOff(x,y,2);
//                  }
//              }
//          }
//          turnOff(3, 3, 4);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//          //

```

```

//          // step 8
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if((x == 2) || (x == 4)) || ((y == 2) ||
(y == 4))) {
//                      turnOn(x,y,5);
//                      turnOff(x,y,4);
//                  }
//              }
//          }
//          for (int x = 1; x < 6; x++) {
//              for(int y = 1; y < 6; ++y) {
//                  if((x == 1) || (x == 5)) || ((y == 1) ||
(y == 5))) {
//                      turnOn(x,y,4);
//                      turnOff(x,y,3);
//                  }
//              }
//          }
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//
//          // step 8
//          turnOn(3, 3, 4);
//          turnOff(3, 3, 5);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//
//          // step 9
//          turnOn(3, 3, 3);
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if((x == 2) || (x == 4)) || ((y == 2) ||
(y == 4))) {
//                      turnOn(x,y,4);
//                      turnOff(x,y,5);
//                  }
//              }
//          }
//          turnOff(3, 3, 4);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//
//          // step 10
//          turnOn(3, 3, 2);
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if((x == 2) || (x == 4)) || ((y == 2) ||
(y == 4))) {
//                      turnOn(x,y,3);
//                      turnOff(x,y,4);
//                  }
//              }
//          }
//          }

```

```

//          turnOff(3, 3, 3);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//
//          // step 11
//          turnOn(3, 3, 1);
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if(((x == 2) || (x == 4)) || ((y == 2) ||
(y == 4))) {
//                      turnOn(x,y,2);
//                      turnOff(x,y,3);
//                  }
//              }
//          }
//          for (int x = 1; x < 6; x++) {
//              for(int y = 1; y < 6; ++y) {
//                  if(((x == 1) || (x == 5)) || ((y == 1) ||
(y == 5))) {
//                      turnOn(x,y,3);
//                      turnOff(x,y,4);
//                  }
//              }
//          }
//          turnOff(3, 3, 2);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//
//          // step 12
//          turnOn(3, 3, 0);
//          for (int x = 2; x < 5; x++) {
//              for(int y = 2; y < 5; ++y) {
//                  if(((x == 2) || (x == 4)) || ((y == 2) ||
(y == 4))) {
//                      turnOn(x,y,1);
//                      turnOff(x,y,2);
//                  }
//              }
//          }
//          for (int x = 1; x < 6; x++) {
//              for(int y = 1; y < 6; ++y) {
//                  if(((x == 1) || (x == 5)) || ((y == 1) ||
(y == 5))) {
//                      turnOn(x,y,2);
//                      turnOff(x,y,3);
//                  }
//              }
//          }
//          turnOff(3, 3, 1);
//          spiSendBoard();
//          tcDelayMillis(delayTime);
//      }
}

```

## Verilog

```
// E155 Final Project
// LED Matrix Etch-a-Sketch
// Sean Hoerger, Alex Smith
// shoerger@g.hmc.edu, acsmith@g.hmc.edu

// testbench
module testbench();
    logic clk, reset, sck, sdi, load, flopClk;
    logic [6:0] layerSelect, rowLEDs, flopCLR;

    LEDMatrix ledmtx(clk, reset, sck, sdi, load, flopClk, layerSelect,
rowLEDs, flopCLR);

    initial
        forever begin
            clk = 1'b0; #2;
            clk = 1'b1; #2;
        end

    initial begin
        reset = 1'b1; #20;
        reset = 1'b0;
    end

endmodule

// top level module controlling the LED matrix and implementing SPI
communication with
// the microcontroller
module LEDMatrix(input logic clk, reset,
                 input logic sck,
                 input logic sdi,
                 input logic load,
                 output logic flopClk,
                 output logic [6:0] layerSelect, rowLEDs,
flopCLR);
    // instantiate internal logic signals
    logic [342:0] interLEDdata, LEDdata;
    logic slowClkSig, writeData, addr;
    logic [391:0] spiLEDdata, memLEDdata;
    logic [5:0] rowSelect;

    // connect the modules together
    LEDMatrix_spi          spi(clk, sck, sdi, load, spiLEDdata);
    levelToPulse           l2p(slowClkSig, reset, load, writeData);
    slowClk                sclk(clk, reset, slowClkSig);
endmodule
```

```

    regClk                regclk(slowClkSig, reset, interLEDData, LEDdata);
    systemController      sc(slowClkSig, reset, rowSelect, layerSelect, flopCLR,
addr);
    mux49                 muxer(LEDdata, rowSelect, rowLEDs);
    decoderMEM            decMEM(clk, writeData, addr, spiLEDdata,
memLEDdata);
    decoder               dec(memLEDdata, interLEDData);
    // send the slow clock signal to the flops
    assign flopClk = slowClkSig;

endmodule

// module for SPI communication with microcontroller
module LEDMatrix_spi(input logic clk,
                    input logic sck,
                    input logic sdi,
                    input logic load,
                    output logic [391:0] spiLEDdata);

    logic [391:0] spiLEDdata_captured;
    // capture input when load is asserted
    always_ff@(posedge sck)
        if(load) spiLEDdata_captured = {spiLEDdata_captured[390:0],sdi};
    always_ff@(posedge clk)
        if(!load) spiLEDdata = spiLEDdata_captured;
endmodule

// level to pulse converter to send write data signal
module levelToPulse(input logic clk, reset, load,
                    output logic write);

    typedef enum logic [1:0] {s0, s1, s2} statetype;
    statetype state, nextstate;

    always_ff@(posedge clk, posedge reset)
        if(reset) state <= s0;
        else state <= nextstate;

    always_comb
        case(state)
            s0: if(!load) nextstate = s1;
                else nextstate = s0;
            s1: if(!load) nextstate = s2;
                else nextstate = s0;
            s2: if(!load) nextstate = s2;
                else nextstate = s0;
        endcase
endmodule

```

```

        assign write = (state == s1);

endmodule

// the decoder module which stores the LED data in memory for access
module decoderMEM(input logic clk,
                 input logic we,
                 input logic a,
                 input logic [391:0] spiLEDData,
                 output logic [391:0] newLEDData);

    logic [391:0] RAM[1:0];
    initial
    $readmemh("C:/Users/acsmith/Desktop/Final_SHAS/memfile.dat.txt",RAM);
    // [note] move this into always_ff to imply block RAM
    assign newLEDData = RAM[a];

    always_ff@(posedge clk)
        if(we) RAM[a] <= spiLEDData;
endmodule

// the module which takes the 391 bit memory and pulls out the LED data
module decoder(input logic [391:0] memLEDData,
              output logic [342:0] LEDdata);

    // pull out the necessary parts of memLEDData
    assign LEDdata[6:0] = memLEDData[6:0];
    assign LEDdata[13:7] = memLEDData[14:8];
    assign LEDdata[20:14] = memLEDData[22:16];
    assign LEDdata[27:21] = memLEDData[30:24];
    assign LEDdata[34:28] = memLEDData[38:32];
    assign LEDdata[41:35] = memLEDData[46:40];
    assign LEDdata[48:42] = memLEDData[54:48];
    assign LEDdata[55:49] = memLEDData[62:56];
    assign LEDdata[62:56] = memLEDData[70:64];
    assign LEDdata[69:63] = memLEDData[78:72];
    assign LEDdata[76:70] = memLEDData[86:80];
    assign LEDdata[83:77] = memLEDData[94:88];
    assign LEDdata[90:84] = memLEDData[102:96];
    assign LEDdata[97:91] = memLEDData[110:104];
    assign LEDdata[104:98] = memLEDData[118:112];
    assign LEDdata[111:105] = memLEDData[126:120];
    assign LEDdata[118:112] = memLEDData[134:128];
    assign LEDdata[125:119] = memLEDData[142:136];
    assign LEDdata[132:126] = memLEDData[150:144];
    assign LEDdata[139:133] = memLEDData[158:152];
    assign LEDdata[146:140] = memLEDData[166:160];

```



```

assign LEDdata[153:147] = memLEDdata[174:168];
assign LEDdata[160:154] = memLEDdata[182:176];
assign LEDdata[167:161] = memLEDdata[190:184];
assign LEDdata[174:168] = memLEDdata[198:192];
assign LEDdata[181:175] = memLEDdata[206:200];
assign LEDdata[188:182] = memLEDdata[214:208];
assign LEDdata[195:189] = memLEDdata[222:216];
assign LEDdata[202:196] = memLEDdata[230:224];
assign LEDdata[209:203] = memLEDdata[238:232];
assign LEDdata[216:210] = memLEDdata[246:240];
assign LEDdata[223:217] = memLEDdata[254:248];
assign LEDdata[230:224] = memLEDdata[262:256];
assign LEDdata[237:231] = memLEDdata[270:264];
assign LEDdata[244:238] = memLEDdata[278:272];
assign LEDdata[251:245] = memLEDdata[286:280];
assign LEDdata[258:252] = memLEDdata[294:288];
assign LEDdata[265:259] = memLEDdata[302:296];
assign LEDdata[272:266] = memLEDdata[310:304];
assign LEDdata[279:273] = memLEDdata[318:312];
assign LEDdata[286:280] = memLEDdata[326:320];
assign LEDdata[293:287] = memLEDdata[334:328];
assign LEDdata[300:294] = memLEDdata[342:336];
assign LEDdata[307:301] = memLEDdata[350:344];
assign LEDdata[314:308] = memLEDdata[358:352];
assign LEDdata[321:315] = memLEDdata[366:360];
assign LEDdata[328:322] = memLEDdata[374:368];
assign LEDdata[335:329] = memLEDdata[382:376];
assign LEDdata[342:336] = memLEDdata[390:384];
endmodule

// the controller for the system instantiated as an FSM
module systemController(input logic clk, reset,
                        output logic [5:0] rowSelect,
                        output logic [6:0]
layerSelect, flopCLR,
                        output logic memAddr);

    // internal logic for FSM
    logic [19:0] controlSig;
    typedef enum logic [5:0] {s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10,
s11, s12, s13, s14, s15, s16, s17, s18, s19, s20, s21, s22, s23, s24, s25, s26,
s27, s28, s29, s30, s31, s32, s33, s34, s35, s36, s37, s38, s39, s40, s41, s42,
s43, s44, s45, s46, s47, s48, s49} statetype;
    statetype state, nextstate;

    // set up the FSM clock
    always_ff@(posedge clk, posedge reset)
        if(reset) state <= s0;
        else      state <= nextstate;

```

```
// next state logic
always_comb
    case(state)
        s0: nextstate = s1;
        s1: nextstate = s2;
        s2: nextstate = s3;
        s3: nextstate = s4;
        s4: nextstate = s5;
        s5: nextstate = s6;
        s6: nextstate = s7;
        s7: nextstate = s8;
        s8: nextstate = s9;
        s9: nextstate = s10;
        s10: nextstate = s11;
        s11: nextstate = s12;
        s12: nextstate = s13;
        s13: nextstate = s14;
        s14: nextstate = s15;
        s15: nextstate = s16;
        s16: nextstate = s17;
        s17: nextstate = s18;
        s18: nextstate = s19;
        s19: nextstate = s20;
        s20: nextstate = s21;
        s21: nextstate = s22;
        s22: nextstate = s23;
        s23: nextstate = s24;
        s24: nextstate = s25;
        s25: nextstate = s26;
        s26: nextstate = s27;
        s27: nextstate = s28;
        s28: nextstate = s29;
        s29: nextstate = s30;
        s30: nextstate = s31;
        s31: nextstate = s32;
        s32: nextstate = s33;
        s33: nextstate = s34;
        s34: nextstate = s35;
        s35: nextstate = s36;
        s36: nextstate = s37;
        s37: nextstate = s38;
        s38: nextstate = s39;
        s39: nextstate = s40;
        s40: nextstate = s41;
        s41: nextstate = s42;
        s42: nextstate = s43;
        s43: nextstate = s44;
```

```

s44: nextstate = s45;
s45: nextstate = s46;
s46: nextstate = s47;
s47: nextstate = s48;
s48: nextstate = s49;
s49: nextstate = s1;
default: nextstate = s0;
endcase

// output logic
always_comb
case(state)
s0: controlSig = 20'b00000000000000000000;
s1: controlSig = 20'b0000000000000100000001;
s2: controlSig = 20'b0000001000000010000010;
s3: controlSig = 20'b0000010000000010000100;
s4: controlSig = 20'b0000011000000010001000;
s5: controlSig = 20'b0000100000000010010000;
s6: controlSig = 20'b0000101000000010100000;
s7: controlSig = 20'b0000110000000011000000;
s8: controlSig = 20'b00001110000001000000001;
s9: controlSig = 20'b0010000000001000000010;
s10: controlSig = 20'b0010010000001000000100;
s11: controlSig = 20'b001010000000100001000;
s12: controlSig = 20'b001011000000100010000;
s13: controlSig = 20'b001100000000100100000;
s14: controlSig = 20'b001101000000101000000;
s15: controlSig = 20'b0011100000010000000001;
s16: controlSig = 20'b0011110000010000000010;
s17: controlSig = 20'b0100000000010000000100;
s18: controlSig = 20'b0100010000010000001000;
s19: controlSig = 20'b010010000001000010000;
s20: controlSig = 20'b010011000001000100000;
s21: controlSig = 20'b010100000001001000000;
s22: controlSig = 20'b01010100000100000000001;
s23: controlSig = 20'b01011000000100000000010;
s24: controlSig = 20'b01011100000100000000100;
s25: controlSig = 20'b01100000000100000001000;
s26: controlSig = 20'b01100100000100000010000;
s27: controlSig = 20'b0110100000010000100000;
s28: controlSig = 20'b0110110000010001000000;
s29: controlSig = 20'b011100000001000000000001;
s30: controlSig = 20'b0111010000000000000010;
s31: controlSig = 20'b011110000001000000000100;
s32: controlSig = 20'b0111110000000000001000;
s33: controlSig = 20'b1000000001000000010000;
s34: controlSig = 20'b1000010001000000100000;
s35: controlSig = 20'b100010001000001000000;

```

```

s36: controlSig = 20'b10001101000000000001;
s37: controlSig = 20'b10010001000000000010;
s38: controlSig = 20'b10010101000000000100;
s39: controlSig = 20'b10011001000000001000;
s40: controlSig = 20'b10011101000000010000;
s41: controlSig = 20'b10100001000000100000;
s42: controlSig = 20'b10100101000001000000;
s43: controlSig = 20'b10101010000000000001;
s44: controlSig = 20'b10101110000000000010;
s45: controlSig = 20'b10110010000000000100;
s46: controlSig = 20'b101101100000000001000;
s47: controlSig = 20'b10111010000000010000;
s48: controlSig = 20'b10111110000000100000;
s49: controlSig = 20'b11000010000001000000;
default: controlSig = 20'b00000000000000000000;
endcase

logic [6:0] preLayer;
assign {rowSelect, flopCLR, preLayer} = controlSig;
// invert the bits so the desired transistor drives a single layer
invertBits invBit(preLayer, layerSelect);
assign memAddr = 1'b0; // always use the same memory location

endmodule

// a module to invert all the bits of a particular signal
module invertBits(input logic [6:0] select,
                 output logic [6:0] newSelect);

    assign newSelect[0] = !select[0];
    assign newSelect[1] = !select[1];
    assign newSelect[2] = !select[2];
    assign newSelect[3] = !select[3];
    assign newSelect[4] = !select[4];
    assign newSelect[5] = !select[5];
    assign newSelect[6] = !select[6];
endmodule

// a regular speed clock storing and passing the LED array signal
module regClk(input logic clk, reset,
             input logic [342:0] d,
             output logic [342:0] q);

    always_ff@(posedge clk, posedge reset)
        if(reset) q <= 343'b0;
        else      q <= d;
endmodule

// a slowClock module producing a slow clock for multiplexing

```

```

module slowClk(input logic clk, reset,
               output logic slowClk);

    // the following code is via a previous microps project Author: Sarp
    Misoglu Date: 11/20/2017
    logic display_clk;
    logic[31:0] counter;

    always_ff @(posedge clk, posedge reset)
        begin
            if(reset)
                begin
                    counter = 0;
                    display_clk = 0;
                end
            else if(counter == 6_000) // good for minimal
flickering/maximal brightness with all lights on
                //else if(counter == 20_000_000) // good for watching each
cycle turn on LEDs
                begin
                    counter = 0;
                    display_clk = ~display_clk;
                end
            else
                begin
                    counter = counter + 1;
                end
        end

    assign slowClk = display_clk;
endmodule

// 49 input MUX used to select the desired 7 bits corresponding to one row of
LEDs
module mux49(input logic [342:0] LEDdata,
             input logic [5:0] rowSelect,
             output logic [6:0] rowLEDs);

    // to begin we parse the LEDdata into 49 x 7 bit parcels

    logic [6:0] l0r6, l0r5, l0r4, l0r3, l0r2, l0r1, l0r0;
    assign {l0r6, l0r5, l0r4, l0r3, l0r2, l0r1, l0r0} = LEDdata[48:0];
    logic [6:0] l1r6, l1r5, l1r4, l1r3, l1r2, l1r1, l1r0;
    assign {l1r6, l1r5, l1r4, l1r3, l1r2, l1r1, l1r0} = LEDdata[97:49];
    logic [6:0] l2r6, l2r5, l2r4, l2r3, l2r2, l2r1, l2r0;
    assign {l2r6, l2r5, l2r4, l2r3, l2r2, l2r1, l2r0} = LEDdata[146:98];
    logic [6:0] l3r6, l3r5, l3r4, l3r3, l3r2, l3r1, l3r0;
    assign {l3r6, l3r5, l3r4, l3r3, l3r2, l3r1, l3r0} = LEDdata[195:147];
    logic [6:0] l4r6, l4r5, l4r4, l4r3, l4r2, l4r1, l4r0;

```

```

assign {l4r6, l4r5, l4r4, l4r3, l4r2, l4r1, l4r0} = LEDdata[244:196];
logic [6:0] l5r6, l5r5, l5r4, l5r3, l5r2, l5r1, l5r0;
assign {l5r6, l5r5, l5r4, l5r3, l5r2, l5r1, l5r0} = LEDdata[293:245];
logic [6:0] l6r6, l6r5, l6r4, l6r3, l6r2, l6r1, l6r0;
assign {l6r6, l6r5, l6r4, l6r3, l6r2, l6r1, l6r0} = LEDdata[342:294];

// a case statement which determines which parcel to select
always_comb
    case(rowSelect)
        0: rowLEDs = l0r0;
        1: rowLEDs = l0r1;
        2: rowLEDs = l0r2;
        3: rowLEDs = l0r3;
        4: rowLEDs = l0r4;
        5: rowLEDs = l0r5;
        6: rowLEDs = l0r6;
        7: rowLEDs = l1r0;
        8: rowLEDs = l1r1;
        9: rowLEDs = l1r2;
        10: rowLEDs = l1r3;
        11: rowLEDs = l1r4;
        12: rowLEDs = l1r5;
        13: rowLEDs = l1r6;
        14: rowLEDs = l2r0;
        15: rowLEDs = l2r1;
        16: rowLEDs = l2r2;
        17: rowLEDs = l2r3;
        18: rowLEDs = l2r4;
        19: rowLEDs = l2r5;
        20: rowLEDs = l2r6;
        21: rowLEDs = l3r0;
        22: rowLEDs = l3r1;
        23: rowLEDs = l3r2;
        24: rowLEDs = l3r3;
        25: rowLEDs = l3r4;
        26: rowLEDs = l3r5;
        27: rowLEDs = l3r6;
        28: rowLEDs = l4r0;
        29: rowLEDs = l4r1;
        30: rowLEDs = l4r2;
        31: rowLEDs = l4r3;
        32: rowLEDs = l4r4;
        33: rowLEDs = l4r5;
        34: rowLEDs = l4r6;
        35: rowLEDs = l5r0;
        36: rowLEDs = l5r1;
        37: rowLEDs = l5r2;
        38: rowLEDs = l5r3;
    endcase

```

```
39: rowLEDs = 15r4;
40: rowLEDs = 15r5;
41: rowLEDs = 15r6;
42: rowLEDs = 16r0;
43: rowLEDs = 16r1;
44: rowLEDs = 16r2;
45: rowLEDs = 16r3;
46: rowLEDs = 16r4;
47: rowLEDs = 16r5;
48: rowLEDs = 16r6;
    default: rowLEDs = 7'b1111111; // default to all LEDs on to
make this case obvious
    endcase
endmodule
```