LED Snake with Tilt Controls

Final Project Report
December 13, 2019
ENGR155

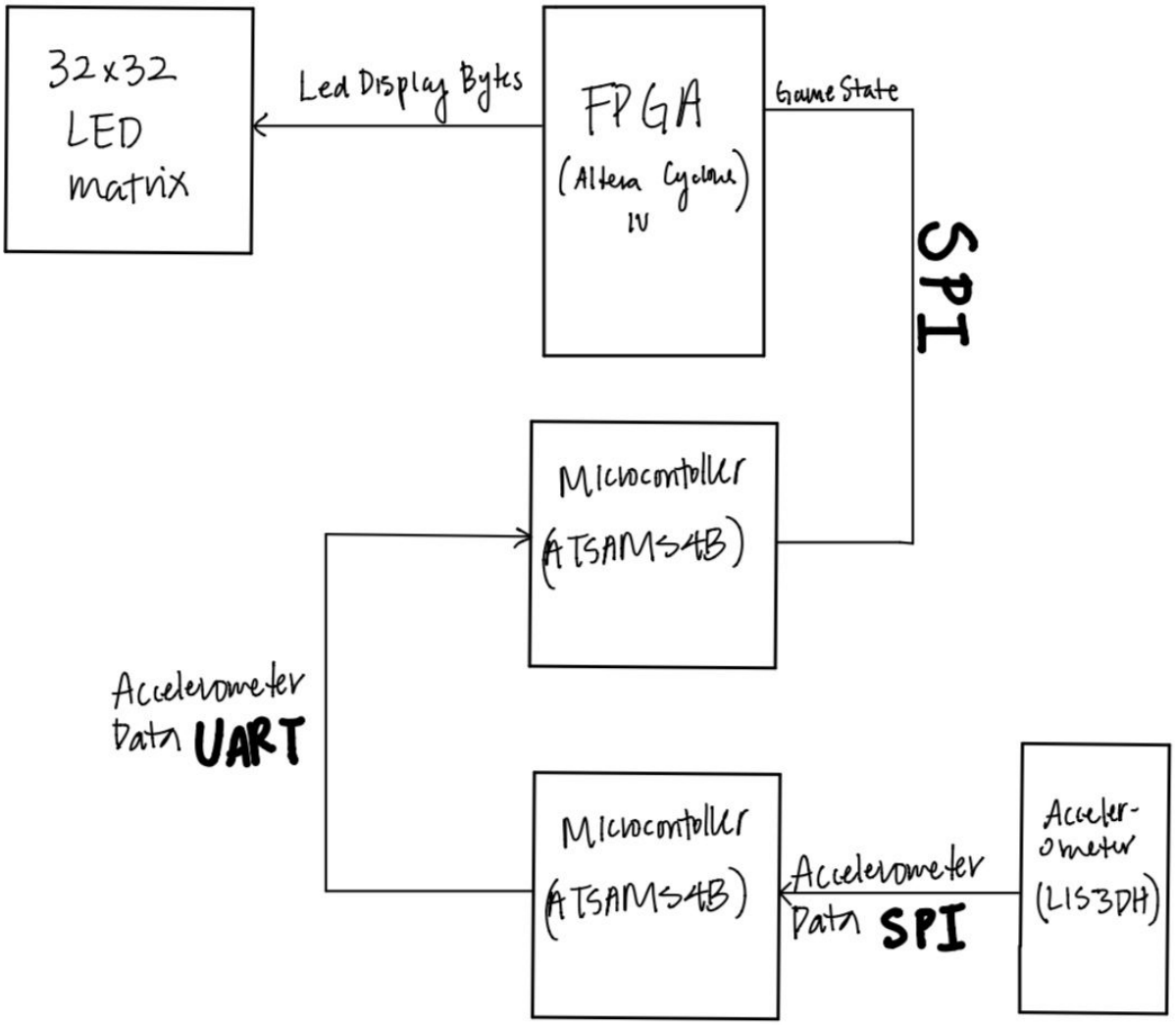
Shivam Malpani and Monica Yao

Abstract:

Snake is a popular game that has become a classic over time. The objective of the game is to eat as many foods that appear in random locations as possible, and become as long as possible. If your head eats another part of the snake or the wall, you lose. This project would implement the classic game, but with a twist that the user would play the game with an accelerometer. Our project consists of two microcontrollers, and the Adafruit 32x32 LED Matrix which functions as our display. We implemented the game by writing the majority of its logic on one microcontroller, which controls the state of the game, and outputs the pixels onto the LED Matrix. The second microcontroller retrieves data from the accelerometer so it knows the direction the player wants to move in, and sends it to the other microcontroller which controls the game state.

Introduction

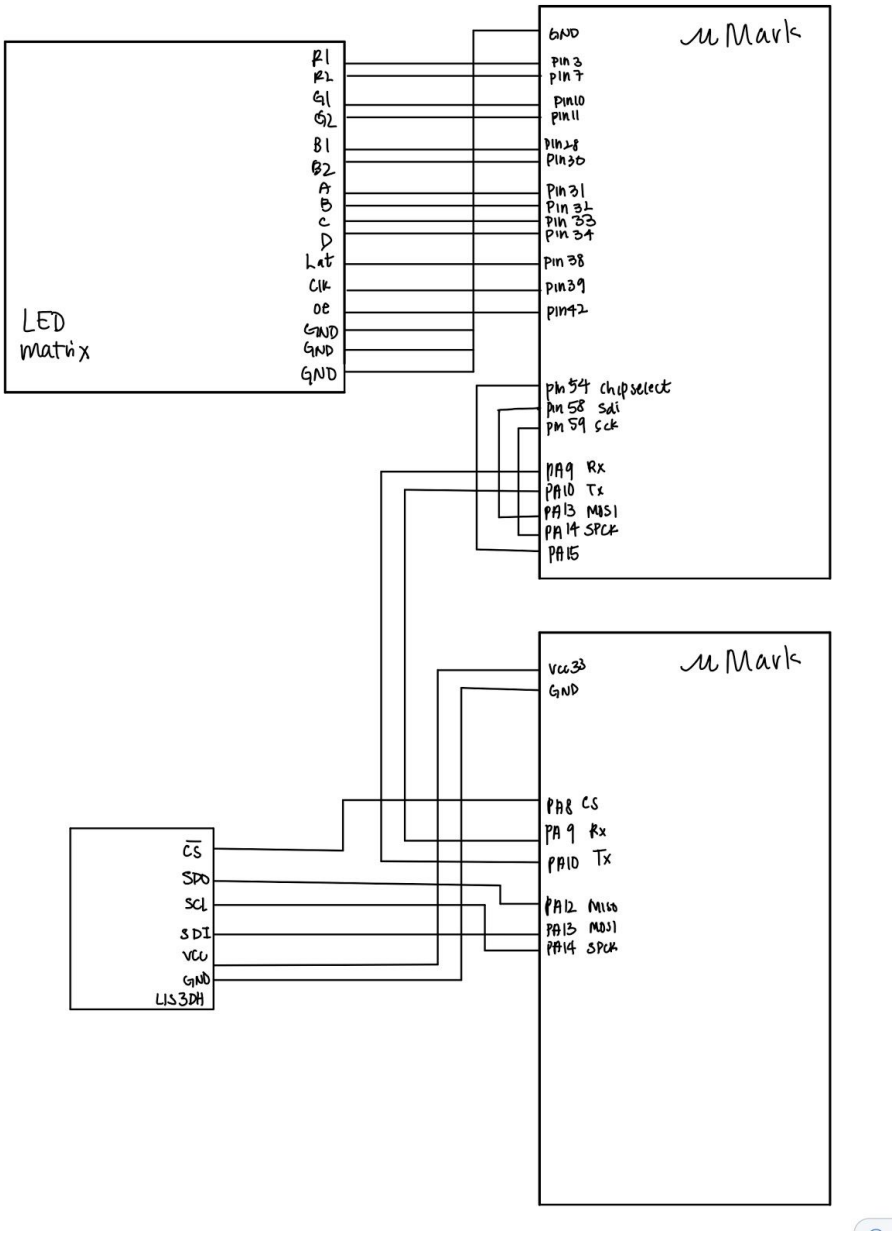
For the Final Project, we want to implement the popular snake video game that is controlled by an accelerometer instead of the traditional keyboard or joystick input. The motivation for this game is to give the users a simpler way to control the classic game. We will be using two ATSAMS4B Microcontrollers and the Altera Cyclone IV FPGA in addition to the 32x32 Adafruit LED Matrix. One of the two microcontrollers controls the game state and communicates with the FPGA. The FPGA takes the game state, and displays the board for users to see. The other microcontroller reads the x, y, and z inputs from the accelerometer, and interprets the readings to output a direction that the user wants to go. The direction is sent back to the other microcontroller that controls the game state as a character through UART.



New Hardware

The new hardware we utilized in this project is the Adafruit 32 x 32 LED Matrix. The FPGA drove the LED display based on the input it received from the microcontroller. We had never interfaced with an LED display before, especially using an FPGA. There was no official documentation for the LED matrix itself. Ray's Logic's website was quite useful, although some of his math isn't fully correct depending on the size of the display you are using. The datasheet of the registers at the back of the display can also be used to get more information of the required format of the SPI waveform.

Schematics



As aforementioned, our project included two microcontrollers, which are connected by and communicate through UART. One of the microcontrollers (top) controls the game state, and sends the appropriate pixels to the FPGA, which then renders it on the LED matrix. The other microcontroller (bottom) reads user input from the accelerometer, and sends it to the game state through UART.

Microcontroller Design

We had successfully set up SPI communication with the accelerometer to read the x,y,z orientation of the sensor. Setting up the communication was difficult as the default communication required 16 cycles of the clock where for a read, 8 bits would be sent by the master and 8 bits would be sent by the slave. We made the mistake of assuming the number of clock cycles depended on the length of the data placed into the transfer registers. After reaching out to a grutor, We found the mistake We were making, and changed the clock to create 16 cycles for every communication. Once we were able to communicate with the accelerometer and get accurate readings, we proceeded to think about ways we can record useful movements. Since we wanted uses to play the snake game, we created a 'reset' position that a user must revert to before their next move would be recorded. This would mean that if the user tilted the sensor right, they would have to move it to the reset position before tilting it again. This allowed us to sense unique movements and avoid repeated detections of movements. The C code reads data from the accelerometer and converts it into a character that is used as user input for updating the game. We use a variable 'resetPos' that checks whether the user has been to the reset position before, and only processes the readings of the accelerometer if the variable it set to true. Once the reading is taken, the variable is set to false. This logic can be seen in the acclerometer.c file attached in the appendix.

We also successfully set up SPI communication with the LED display. The SPI was set up to communicate on 16 clock cycles. The display would take in 2 bytes of data for each pixel. There were 32 pixels in a row and the LED display required 6 bytes of data to latch and blank. Thus, we sent a total of 32*2 + 4= 70 bytes of data for each row of the LED and use 35 SPI send commands for lighting up every row. This logic can be seen in the LEDToByte method in game.c file attached in the appendix.

We implemented the game logic to that would update the snake movements, food positions and update the board accordingly. The snake's next movements were based on user input from the accelerometer. If the head of the snake ate a fruit, the snake would grow in size, and the food would be replaced by another fruit placed randomly on the board. If the snake collided with a wall or by itself, it would terminate the game. Timing of the game was also an important element as we wanted the game to be played at a comfortable speed. We controlled the speed of the game by using a timer counter, but we also didn't want to miss out on sampling the accelerometer so we would sample the accelerometer multiple times for every time we updated the game.

FPGA Design

Since the majority of our game logic remains in the microcontroller, the FPGA has the purpose of receiving data from the microcontroller, and using the revived data to drive the LED matrix. Due to the 16 bit restraint from the microcontroller, as 16 bits are required for accelerometer communication, we stayed consistent and read 16 bit data from the microcontroller through SPI. The FPGA is broken down into two modules, one called led_matrix and the other called spi_slave_read. The first module takes a 16 bit data input, a slow clock generated by the other module, and a chip select which functions as a reset. The second module shifts the SPI slave data in, and generates a done signal, which function as a clock for the led_matrix module. As a reminder, the clock that LED matrix functions on marks the end of shifting in an input.

In more details, the led_matrix takes in a slow clock, 16 bits of led data, and a chip select, and outputs the bits necessary for driving the LED matrix, including R1, G1, B1, R2, G2, B2, A, B, C, D, output enable, latch, and an output clock signal. The led_matrix module uses a 35 bit counter, since a single row has 32 columns, and we need additional clock cycles to output OE and latch while not giving the LED matrix an output clock. At each negative edge of the input clock, we update the column and row. At each positive edge of the input clock, we update the R1G1B1 values for the current row, and R2G2B2 for current row + 16. When we reach cycle 33, we assert LAT for the data to latch on for that row, and then on cycle 34, make output enable low, which will make sure to not erase the data we have shifted in. During cycles 33-35, we do not output a clock to the LED matrix. We then update the row, and then start scanning the different columns and writing RGB values again.

The spi_slave_read module takes in a clock signal from the microcontroller, and shifts in a bit from the slave data in at every clock signal. We also use a 16 counter to keep track of when the 16 bits are done shifting in, then this done signal functions as a clock for led_matrix, effectively telling the other modules to start when the 16 bits are shifted in. We synchronize the counter and data shifted in with chip select. At negative edge of clock, the counter is updated, and ar the positive edge of clock, the data is shifted in.

Results

We were successful in meeting our minimal requirements. We were able to create a game of snake that was played with an accelerometer input. Our different interfaces successfully communicated through the different serial protocols. However, in our initial proposal, we wanted to implement the game in which the user would be able to wirelessly control the snake, and ideally attached to a glove. There were multiple roadblocks encountered that impeded our progress.

First of all, we were unable to output the desired display with the LED matrix, it was a lot more difficult than expected. Initially, we had thought that because Adafuit provided a tutorial for turning on the LED Matrix, the matrix would not be a great time sink. We thought we could spend the majority of the time setting up the bluetooth connection BlueSMiRF, and debug the wireless communication. However, the tutorial used a built in Arduino library that drove the board with little trouble, as the library handled timing for you. Thus, it proved to be harder than expected to understand when we should time the latch and output enable signals so that the board stays rendered. In addition, simulating in ModelSim proved to be different from the logic analyzer at times, so debugging hardware though it worked in simulation also increased the difficulties of driving the LED Matrix. With additional documentation, perhaps it would have been easier to implement the LED Matrix. The LED matrix also proves to be hard to work with because any loose wires would cause specific rows to be off. It was ambiguous if the incorrect display was due to loose wiring or to our code.

Another difficult aspect of the project was configuring the BlueSMiRF. Initially, we were able to connect to the bluetooth, and connect to the BlueSMiRF through a COM Port. We used the TeraTerm terminal to send set commands on the BlueSMiRF. But even after setting one of the BlueSMiRF to the master configuration and the other to the slave, and set the modules to connect to the addresses of the other, we had little success.

We were able to accomplish the configurations before break, so when we came back, we decided to focus on finishing the game while keeping UART communication between two microcontrollers. This made sure that a pivot towards using the BlueSMiRFs would only require setting up the correct configures for the bluetooth modules. However, when we finished the game, our USB to bluetooth adapter had several driver issues, and we could not connect to the BlueSMiRFs anymore to change the configurations.

References

Harris, David Money, and Sarah L Harris. Digital Design and Computer Architecture. Morgan Kaufmann, 2007.

Parts List

Part	Sources	Vendor Part #	Price
32x32 LED Matrix	Adafruit	607	$50

Appendices

accelerometer.c

```
//E155 Final Project
// Shivam Malpani and Monica Yao 12/13/19
// This file will take in readings from the accelerometer and interpret it send over a direction

//#ifndef ACCELEROMETER
//#define ACCELEROMETER

#include <stdio.h>
#include "SAM4S4B_lab7/SAM4S4B.h"
#define ACC_PIN 8

short resetPos= 0;
volatile unsigned char debug;
volatile short x,y,z, who;
char lastInp= 'u';
uint16_t readX[3]= {0xA9A9, 0xABAB, 0xADAD};
char readAcc;

char getAccelInp(){
	//acclerometer logic
	//spiInit16(5, 0, 1);

	debug = spiSendReceive16(0x202F); //WHO AM I
	who= spiSendReceive16(0x8F8F);
	x = spiSendReceive16(readX[0]);
	y = spiSendReceive16(readX[1]);
	z = spiSendReceive16(readX[2]);
	if(x>-9 && x<12 && y>-13 && y<12 && ((z>55 && z<75) || z==127)){
		resetPos=1;
		return lastInp;
	}

	//right y<-27
```

```
        else if(resetPos==1 && y<-20){
                resetPos= 0;
                lastInp= 'r';
                return 'r';
        }
        //left  y>30
        else if(resetPos==1 && y>20){
                resetPos= 0;
                lastInp= 'l';
                return 'l';
        }
        //up x> 25
        else if(resetPos==1 && x>25){
                resetPos= 0;
                lastInp= 'u';
                return 'u';
        }
        //down x<-25
        else if(resetPos==1 && x<-20){
                resetPos= 0;
                lastInp= 'd';
                return 'd';
        }

        return lastInp;
}

int main(void){
        samInit();
  pioInit();
        tcInit();
        tcDelayInit();
        uartInit(0,140);
        spiInit16(5,0,1);
        pioPinMode(PIO_PA8, PIO_OUTPUT);

        while(1){
                readAcc= getAccelInp();
                uartTx(readAcc);
```

```
        }
        return 1;
}
```

game.c

```
//E155 Final Project
//Shivam Malpani and Monica Yao 12/13/19
//This file is the main game state, which will output the LEDs to the Matrix
//It will also take in a direction from the other microcontroller

#include <stdio.h>
#include <stdlib.h>
#include "SAM4S4B_lab7/SAM4S4B.h"
#include "pixel.h"
#include "lose.h"

#define CS 15 //Chip select

#define SELECT_LED 19
#define SELECT_ACC 20

#define MS 129
#define MS_1 128


Pixel board [32][32];
int snake[20][2];
int lenSnake;
int food[2];
int eaten;
char inpAcc;
void boardInit(){
	for(int i=0; i<32; ++i){
		for (int j = 0; j<32; ++j){
			if (i == 0 || i == 1 || i == 30 || i == 31 || j==0 || j==1 || j==30||
j==31){
				setWall(&board[i][j]);
			}
			else{
				setGround(&board[i][j]);
			}
		}
```

```
	}
}

/*
This method puts the snanke on the board.
It sets the board pixels to a isSnake for every
unit of the snake.
*/
void putSnakeOnBoard(){
	for(int i=0; i<lenSnake; ++i){
		setSnake(&board[snake[i][1]][snake[i][0]]);
	}
}

/*
This method removes the snake off the board. It does this by
setting every pixel of the board that was previously a snake,
to the ground.
*/
void removeSnakeOnBoard(){
		for(int i=0; i<lenSnake; ++i){
		setGround(&board[snake[i][1]][snake[i][0]]);
	}
}
/*
Initilizes a snake of length 5 in the middle of the board.
This method also places the snake on the board.
*/
void snakeInit(){
	int initPos_X= 16;
	int initPos_Y= 16;
	lenSnake= 5;
	for(int i=0; i<lenSnake; ++i){
		snake[i][0]= initPos_X;
		snake[i][1]= initPos_Y+i;
	}
	putSnakeOnBoard();
}
```

```
/*
This method puts the food on the board by setting
a pixel of the board to isFood.
*/
void putFoodOnBoard(){
		setFood(&board[food[1]][food[0]]);
}

/* This method removes the food off the board by setting the pixel of
the board that was previously food to ground.*/
void removeFoodOnBoard(){
	setGround(&board[food[1]][food[0]]);
}

/* This method initilizes food to a location on the upper half of the board. */
void foodInit(){
	food[0]=16;
	food[1]=7;
	putFoodOnBoard();
}

// UPDATE FUNCTIONS

/* It checks if the food was eaten. If it is eaten
it updates the location of the food to a new random location
on the board. */
void updateFood(){
	if(eaten){
		removeFoodOnBoard();
		do{
			food[0]= rand()%29;
			food[1]= rand()%29;
		}while(board[food[1]][food[0]].isSnake || food[0]<4 || food[1]<4);
		putFoodOnBoard();
		eaten=0;
	}
}

/* This method updates the position of the snake as it moves. The direction the snake moves
```

```
   depends on user input which is passed in as a character. */
void updateSnake(char move){
	removeSnakeOnBoard();
	int nextPos_X= snake[0][0];
	int nextPos_Y= snake[0][1];

	if(move=='d'){
		++nextPos_Y;
	}
	else if(move=='r'){
		++nextPos_X;
	}
	else if(move=='l'){
		--nextPos_X;
	}
	else{ // 'u'
		--nextPos_Y;
	}
	if(nextPos_X==food[0] && nextPos_Y==food[1]){
		if(lenSnake!=20){
			++lenSnake;
		}
		eaten=1;
	}

	//shift
	for(int i= lenSnake-2; i>=0; --i){
		snake[i+1][0]= snake[i][0];
		snake[i+1][1]= snake[i][1];
	}
	snake[0][0]=nextPos_X;
	snake[0][1]=nextPos_Y;
	putSnakeOnBoard();
}

/* This method checks if the snake has collided with a wall or itself */
bool checkCollsion(){
	volatile int i=0;
	volatile int j=0;
```

```
	for(i=0; i<lenSnake; ++i){
		//checks for wall collisions
		if(snake[i][0]<3 || snake[i][1]<3 || snake[i][0]>29 || snake[i][1]>29){
			return 1;
		}
		for(j=0; j<i; ++j){
			//
			if(i!=j && snake[i][0] ==snake[j][0] && snake[i][1]==snake[j][1]){
				return 1;
			}
		}
	}
	return 0;
}

/* this method calls other update methods to update the entire game */
int updateGame(){
	//char inp= uartRx();
	updateSnake(inpAcc);
	updateFood();
	bool lose= checkCollsion();
	return lose;
}

/* This method sends the LED data one pixel at a time. It sends 35*2 bytes for every row since
32*2 bytes send pixel data
  and 3*2 bytes are for shifting the bytes in. */
void sendLED()
{
	pioDigitalWrite(CS, 1);
	pioDigitalWrite(CS, 0);
	int i,j;
	for (i = 0; i < 16; ++i){
		for (j = 0; j < 35; ++j){
			uint16_t result;
			if(j>=32){
				result= 0x00;
			}
			else{
```

```
                                char top, bot;
                                if(i==snake[0][1] && j==snake[0][0]){
                                        top= 0x01;
                                }
                                else{
                                 top = LEDToByte(&board[i][j]);
                                }
                                if(i+16==snake[0][1] && j==snake[0][0]){
                                        bot=0x01;
                                }
                                else{
                          bot = LEDToByte(&board[i+16][j]);
                                }
                         result = (top<<3) | bot;
                        }

                        spiSend(result);
                }
        }
}

/* Reads a char for user input from the acclerometer. This information is send either via wired
UART or bluetooth UART */
void readAcc(){
        inpAcc= uartRx();
        while(inpAcc!='u' && inpAcc!='d' && inpAcc!='l' && inpAcc!='r'){
                inpAcc= uartRx();
        }
}

/* The main gets all the differnent pieces together and uses a timer to play the game at a decent
pace. */
int main(void){
        samInit();
  pioInit();
        tcInit();
        tcDelayInit();
        boardInit();
        snakeInit();
```

```
        foodInit();

  spiInit16(5,0,1);
        uartInit(0,140);

        pioPinMode(CS, PIO_OUTPUT);
        pioPinMode(SELECT_LED, PIO_OUTPUT);
        pioPinMode(SELECT_ACC, PIO_OUTPUT);

        tcChannelInit(TC_CH0_ID,  TC_CLK2_ID, TC_MODE_UP); // mck/8 and counter just
goes up with no comparison to RC
        tcChannelInit(TC_CH1_ID,  TC_CLK2_ID, TC_MODE_UP);

        // start with making both SPIs NOT slaves
        pioDigitalWrite(SELECT_LED, 1);
        pioDigitalWrite(SELECT_ACC, 1);

        bool lose = 0;
        unsigned long end = MS*4000/8;
        unsigned long end1 = MS_1*4000/8;
        int temp=400;
        while(--temp){
                sendLED();
        } //To display the game for a few moments before the game starts.
        while(!lose){
                lose= updateGame();
                sendLED();
                for(int c=0; c<3; ++c){
                        tcResetChannel(TC_CH1_ID);
                        while (tcReadChannel(TC_CH1_ID) < end1){
                                tcResetChannel(TC_CH0_ID);
                                while (tcReadChannel(TC_CH0_ID) < end){
                                        sendLED();
                                        readAcc();

                                }
                }
        }
```

```
        }
        //The game ends if the player loses.
        while(1){
                sendLoseScreen();
        }
        return 1;
}
```

matric.sv

```
//E155 Final Project
//Shivam Malpani and Monica Yao 12/13/19
//This file will drive the LED matrix, using the SPI information from the microcontroller

// The top module for driving the matrix
module led_1(input logic clk, chip_select, // chip select is default high
                                           input logic sck, sdi,
                                           output logic r1, g1, b1,
                                           output logic r2, g2, b2,
                                           output logic a, b, c, d,
                                           output logic lat, oe, led_clk);

        logic [15:0] led_data;
        logic [3:0] row;
        logic [5:0] column;
        logic sclk;

        led_matrix matrix(sclk, chip_select, led_data, r1, g1, b1, r2, g2, b2, row, column, a, b,
c,d, lat, oe, led_clk);
        spi_slave_read spi(sck, chip_select, sdi, sclk, led_data);
endmodule


module led_matrix(input logic sclk, chip_select,
                                           input logic [15:0] led_data, // led_data changes
every cycle (it is readings from the SPI)
                                           output logic r1, g1, b1,
                                           output logic r2, g2, b2,
                                           output logic [3:0] row,
                                           output logic [5:0] column,
                                           output logic a, b, c, d,
                                           output logic lat, oe, led_clk);


        logic [2:0] rgb1, rgb2;

        assign rgb1 = led_data[5:3];
```

```
assign rgb2 = led_data[2:0];

//The values of column and row are updated on row and column.
always_ff@(negedge sclk, posedge chip_select)
	if(chip_select)
		begin
			column<= 34;
			row<=0;
		end
	else if (column >= 34)
		begin
			column <= 0;
		end
	else
		begin
			column <= column + 1;
			if(column==33)
				row<= row+1;
		end


//Input is sent to the LED on posedge.
always_ff @(posedge sclk, posedge chip_select)

	if (chip_select)
		begin
		{r1, g1, b1} <= 0;
			{r2, g2, b2} <= 0;
		end

	else if (column < 32)
		begin

			{r1, g1, b1} <= rgb1;
			{r2, g2, b2} <= rgb2;
		end

//combinational logic for latch and output enable/blank.
always_comb
```

```
                begin
                oe= ~(column==33);//~(column>31&& column<34);
                lat= (column==32);
                led_clk= sclk && column<32;
                end

        assign a = row[0];
        assign b = row[1];
        assign c = row[2];
        assign d = row[3];
endmodule

// Read in 35*2 bytes from the ATSAM.
module spi_slave_read(input  logic sck, chip_select,
            input  logic sdi,
            output  logic done,
            output logic [15:0] led_data);

   logic [3:0] byte_count;


        always_ff@(negedge sck, posedge chip_select)
                if (chip_select) byte_count = 4'b1111;
                else byte_count = byte_count + 4'b1;



        // shift register for the input
   always_ff @(posedge sck)
      led_data <= (byte_count == 0)? {15'bxxxxxxx, sdi} : {led_data[14:0], sdi};


   // it is done with the byte count hits 15 ( this is used as clk for leds )

   assign done = (byte_count == 4'd15);

endmodule
```

lose.h

```
//E155 Final Project
//Shivam Malpani and Monica Yao 12/13/19

#ifndef LOSE
#define LOSE

#define RESET_PIN 15

const char loseBoard[32][32] = {
 {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'},  // 0
 {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'},  // 1
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '#', '#'},  // 2
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '#', '#'},  // 3
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '#', '#'},  // 4
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '#', '#'},  // 5
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '#', '#'},  // 6
 {'#', '#', '-', '-', '-', '-', '+','-', '-', '-', '-', '-', '+', '-', '-', '-', '+', '-', '-', '-', '-','+', '-', '-', '-', '+', '-', '-', '-', '-', '#', '#'},  // 7
 {'#', '#', '-', '-', '-', '-', '-','+', '-', '-', '-', '+', '-', '-', '-', '+', '+', '+', '-', '-', '-','+', '-', '-', '-', '+', '-', '-', '-', '-', '#', '#'},  // 8
 {'#', '#', '-', '-', '-', '-', '-','-', '+', '-', '+', '-', '-', '-', '+', '+', '+', '+', '+', '-', '-','+', '-', '-', '-', '+', '-', '-', '-', '-', '#', '#'},  // 9
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '+', '-', '-', '-', '-', '+', '+', '+', '+', '+', '-', '-','+', '-', '-', '-', '+', '-', '-', '-', '-', '#', '#'},  // 10
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '+', '-', '-', '-', '-', '+', '+', '+', '+', '+', '-', '-','+', '-', '-', '-', '+', '-', '-', '-', '-', '#', '#'},  // 11
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '+', '-', '-', '-', '-', '-', '+', '+', '+', '-', '-', '-','-', '+', '-', '+', '-', '-', '-', '-', '-', '#', '#'},  // 12
```

```
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '+', '-', '-', '-', '-', '-', '-', '+', '-', '-', '-', '-','-', '-', '+', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 13
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 14
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 15
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 16
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 17
 {'#', '#', '-', '-', '-', '-', '-','+', '-', '-', '-', '-', '-', '-', '+', '+', '-', '-', '-', '-', '+','+', '-', '-', '+', '+', '+', '-', '-', '-',
'#', '#'},  // 18
 {'#', '#', '-', '-', '-', '-', '-','+', '-', '-', '-', '-', '-', '-', '+', '+', '+', '+', '-', '-', '+', '-','-', '+', '-', '+', '-', '-', '-', '-', '-',
'#', '#'},  // 19
 {'#', '#', '-', '-', '-', '-', '-','+', '-', '-', '-', '-', '-', '+', '+', '+', '+', '+', '+', '-', '-', '+','+', '-', '-', '+', '-', '-', '-', '-', '-',
'#', '#'},  // 20
 {'#', '#', '-', '-', '-', '-', '-','+', '-', '-', '-', '-', '-', '+', '+', '+', '+', '+', '+', '-', '-', '-','-', '+', '-', '+', '+', '-', '-', '-', '-',
'#', '#'},  // 21
 {'#', '#', '-', '-', '-', '-', '-','+', '-', '-', '-', '-', '-', '-', '+', '+', '+', '+', '-', '-', '-', '-','-', '+', '-', '+', '-', '-', '-', '-', '-',
'#', '#'},  // 22
 {'#', '#', '-', '-', '-', '-', '-','+', '+', '+', '+', '-', '-', '-', '+', '+', '-', '-', '-', '+', '-','-', '+', '-', '-', '+', '-', '-', '-', '-', '-',
'#', '#'},  // 23
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '+','+', '-', '-', '+', '+', '+', '-', '-', '-', '#',
'#'},  // 24
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 25
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 26
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 27
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 28
 {'#', '#', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-','-', '-', '-', '-', '-', '-', '-', '-', '#',
'#'},  // 29
 {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#'},  // 30
 {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',
'#', '#', '#', '#'},  // 31
};
```

```
/* Converts the string to a the corresponding character based on the color scheme decided. */
char stringToByte(char inputChar)
{
	char result;
	switch(inputChar) {
		case '#':
			result = 0x01; // WALL: 0000_0111
			break;
		case '-':
			result = 0x00; // IDK: 0000_0000
			break;
		case '+':
			result = 0x07; // TEXT: 0000_0100
			break;
	}
	return result;
}

/* Sends the lose screen to the LED. The logic is the same as the one in LEDToByte */
void sendLoseScreen()
{
	pioDigitalWrite(RESET_PIN, 1);
	pioDigitalWrite(RESET_PIN, 0);
	int i,j;
	for (i = 0; i < 16; ++i){
		for (j = 0; j < 35; ++j){

			char result;
			if(j>=32){
				result= 0x00;
			}
			else{
			 char top = stringToByte(loseBoard[i][j]);
			 char bot = stringToByte(loseBoard[i+16][j]);
			 result = (top<<3) | bot;
			}
			spiSend(result);
		}
```

```
        }
}

#endif
```

pixel.h

```
//E155 Final Project
//Shivam Malpani and Monica Yao 12/13/19

#ifndef PIXEL
#define PIXEL

#include <stdbool.h>

/* Definition of a pixel. At any a point a pixel can be the wall, snake or food. */
typedef struct {
	bool isWall;
	bool isGround;
	bool isSnake;
	bool isFood;
} Pixel;

/* --------- Setter methods ---------- */

void setWall(Pixel * inputP){
	inputP->isWall = true;
	inputP->isGround = false;
	inputP->isSnake = false;
	inputP->isFood = false;
}

void setGround(Pixel * inputP){
	inputP->isWall = false;
	inputP->isGround = true;
	inputP->isSnake = false;
	inputP->isFood = false;
}

void setSnake(Pixel * inputP){
	inputP->isWall = false;
	inputP->isGround = false;
	inputP->isSnake = true;
	inputP->isFood = false;
```

```
}

void setFood(Pixel * inputP){
	inputP->isWall = false;
	inputP->isGround = false;
	inputP->isSnake = false;
	inputP->isFood = true;
}


/* This method replies with a character that will indicate the right color for the pixel.
  Snake is Red, Food is Green, Wall is White. */
char LEDToByte(Pixel * inputP)
{
	//Rxx- 0100- 0x04
	//xGx- 0010- 0x02
	//xxB- 0001- 0x01
	char result;
	if (inputP->isWall){
		result = 0x07;
	}
	else if(inputP->isFood){
		result= 0x06;
	}
	else if(inputP->isSnake){
		result= 0x04;
	}
	else {
		result= 0x00;
	}

	return result;
}

#endif
```