Lab 5
Dr. Ryan Gerdes
ECE 3710
By: Jonathan Tousley
Partner: Aaron Kunz

Overview:

The microcontroller was configured to interface with an LCD touchscreen. The screen would display 3 boxes and upon being touched would turn each box would turn its respective LED on and off. For operation, this project requires:

- A 5V voltage source for the LCD touchscreen (3.3 volts gave inconsistent results)
- The LCD touchscreen
- Lots of wires

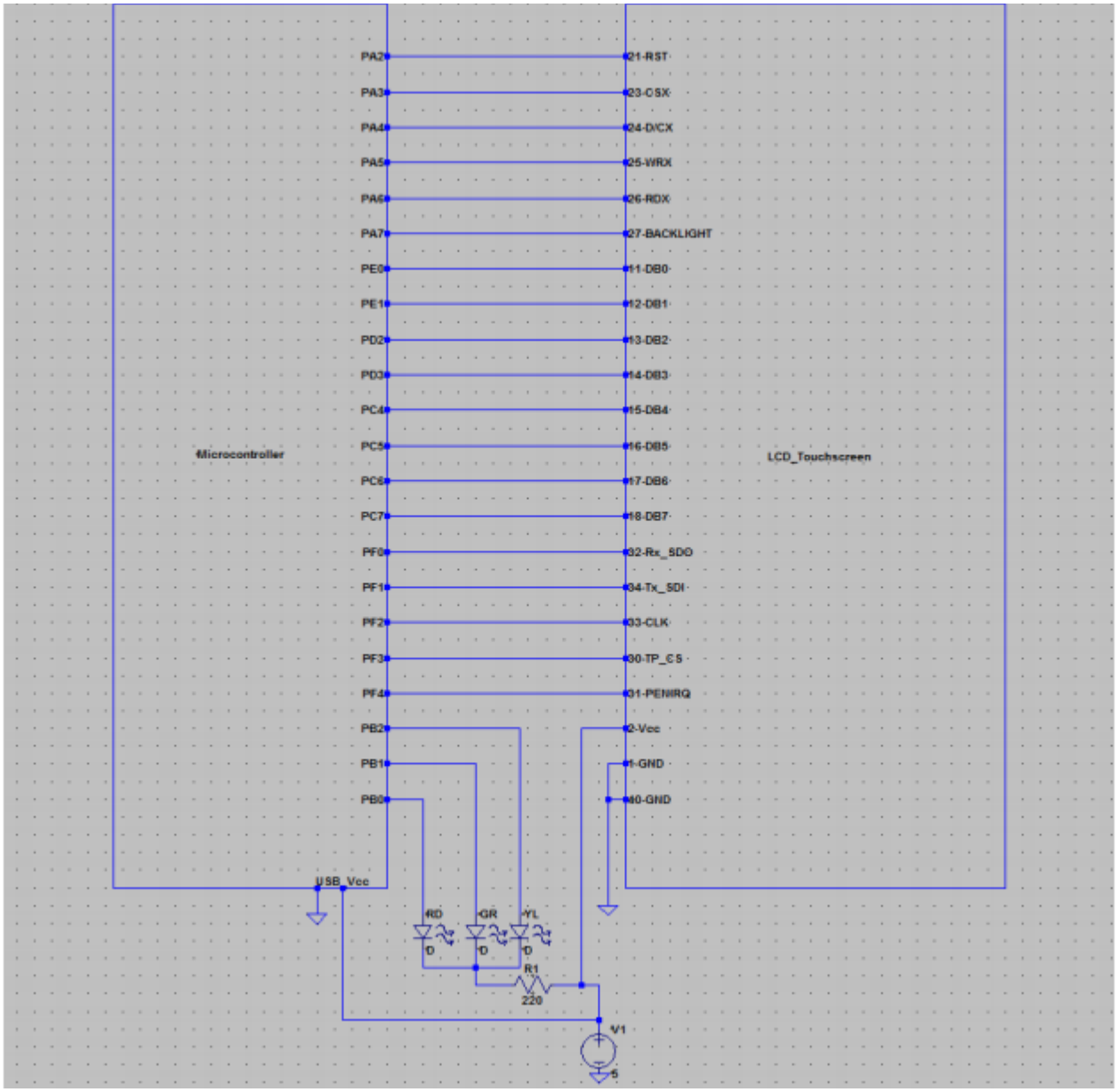
Hardware details:

The microcontroller was connected to the LCD touchscreen using MCU 8080-II series parallel interface. This saved some pins on the microcontroller and allowed the Tiva to send the same data only slightly slower. For the touchscreen portion, the microcontroller used SPI at about 2 MHz. The Fss was used for the chip select and the touchscreen PENIRQ pin was used as an interrupt for the microcontroller. A schematic of the wiring is given below:

Software details:

After the initial configuration (drawing the LCD screen), the microcontroller did nothing (but could have been computing pi) while it waited to be interrupted by input from the user. The interrupt was configured for a logic low, so the interrupt continued to be triggered for as long as the user touched the screen. This removed any problems for touching and holding. For each interrupt, the microcontroller asked the touchscreen for the X and Y coordinates of the input, and added to a total value and count. When the user stopped touching the screen, the interrupts ceased and the total values for both X and Y were divided by the number of coordinates to get the average. It was this value that was used to determine the input location.

Once the location of input was known, and quick check on the bounds of X and Y determined if the location fell within the squares and if so which one. If yes, the state of the box and the LED changed to the appropriate value. The following flowchart shows the design logic:

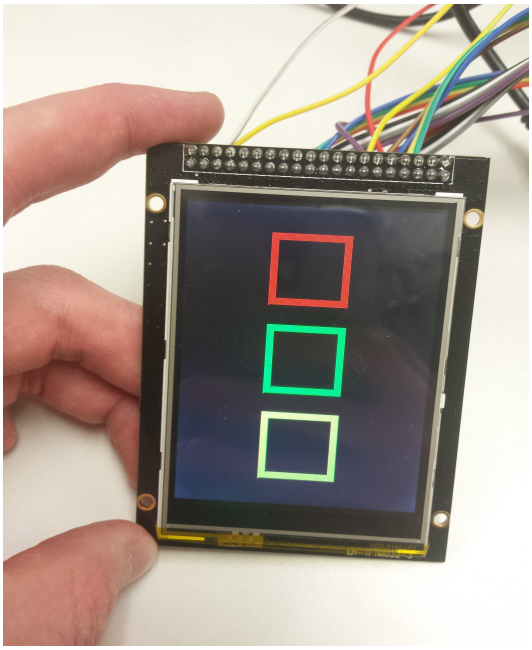


SPI (SSI):

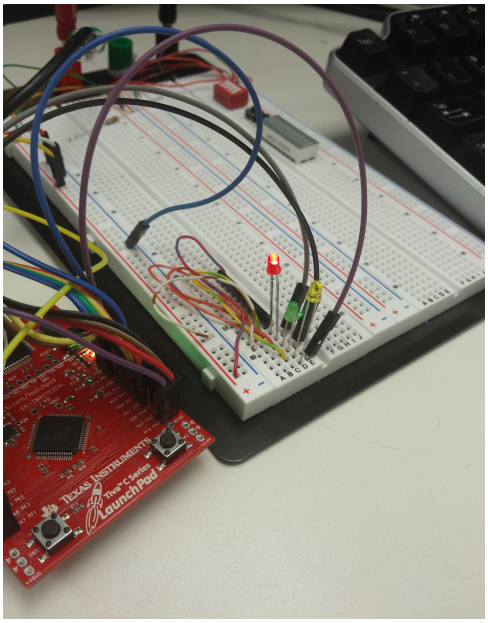
Due to previous wiring and laziness, we used Port F's alternate function for the SPI module at about 2 MHz (66.67 MHz / 14), because that's what worked and there was no reason to change it. The clock was set to 66.67 MHz because we couldn't figure out how to make it faster. Everything worked quite well at that speed although we wish it was faster because the redrawing is still visible to the eye.

Requirements:

1. Three buttons must be drawn on the LCD. One red, one green, and one yellow. The buttons will be blank colored outlines initially (like the red and yellow button in figure 1).
2. When a button is touched the button will be filled (like the green button in figure 1). When that button is touched again the button should be return to a blank outline.

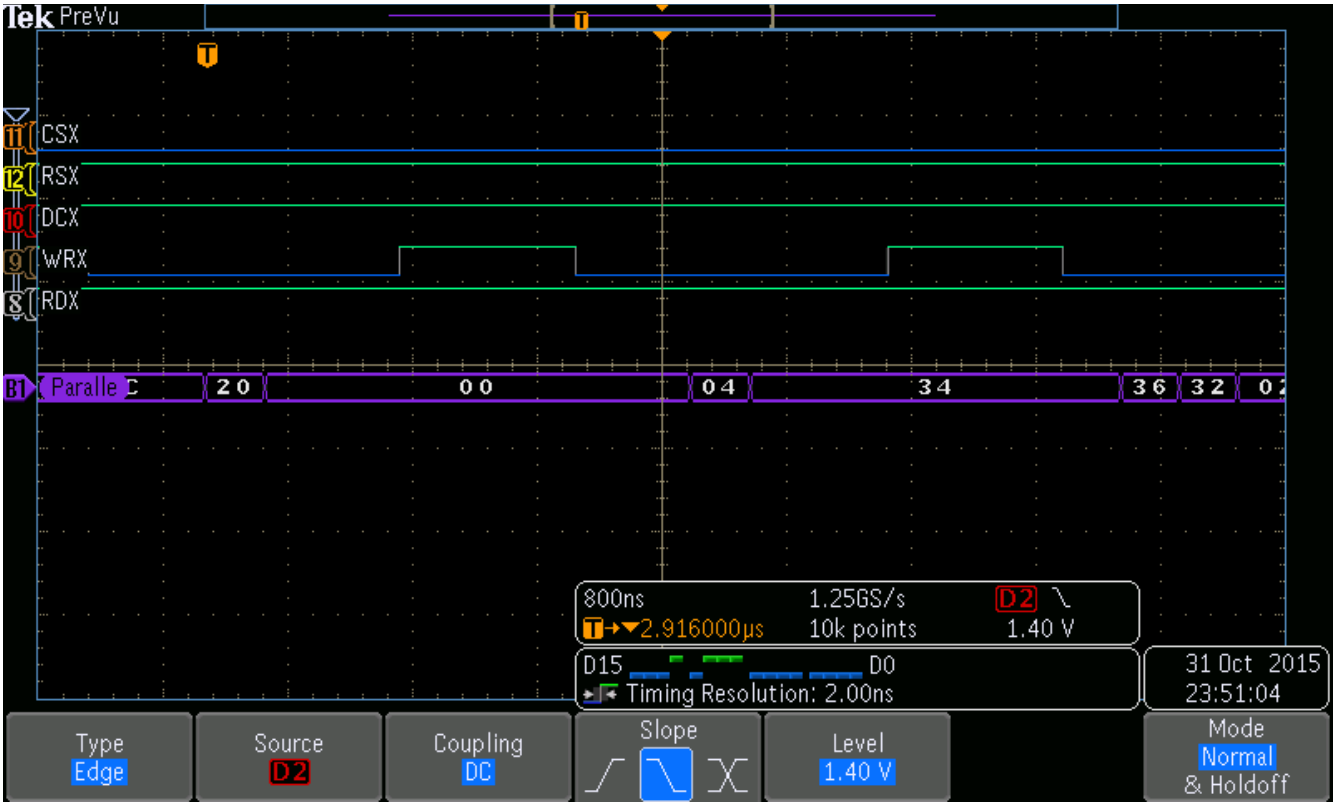


3. The red, green, and yellow LEDs should be wired to three available ports for output. The LEDs must be lit up with the corresponding LCD button. If the LCD button is solid, the LED should be on and vice versa.



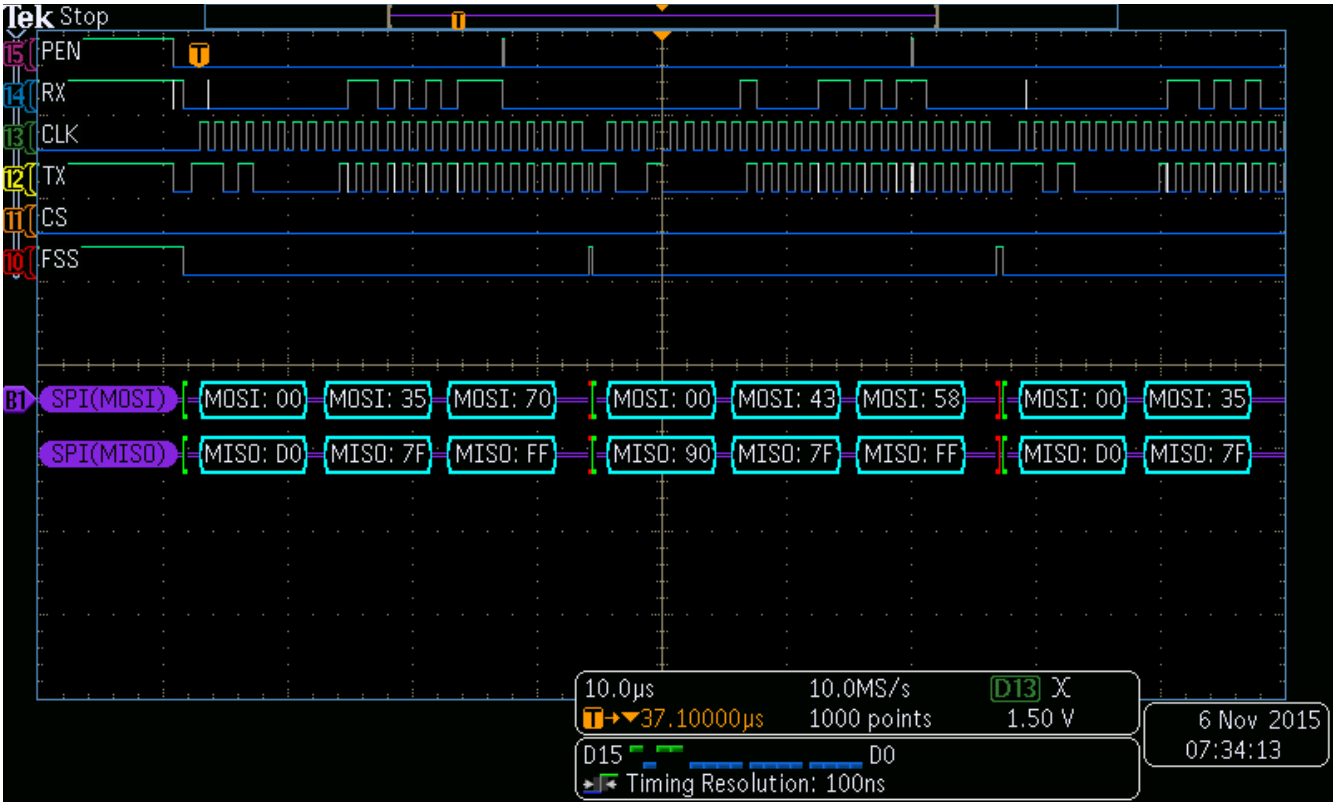
4. The screen refresh rate must be acceptably fast. What is the maximum clock rate that the ILI9341 controller can handle?

As far as we know, the 66.67 MHz could be the maximum rate the LCD screen can handle. Below is a screenshot of data being written to the LCD screen. Write pin goes low, data goes onto the bus, write pin goes high:

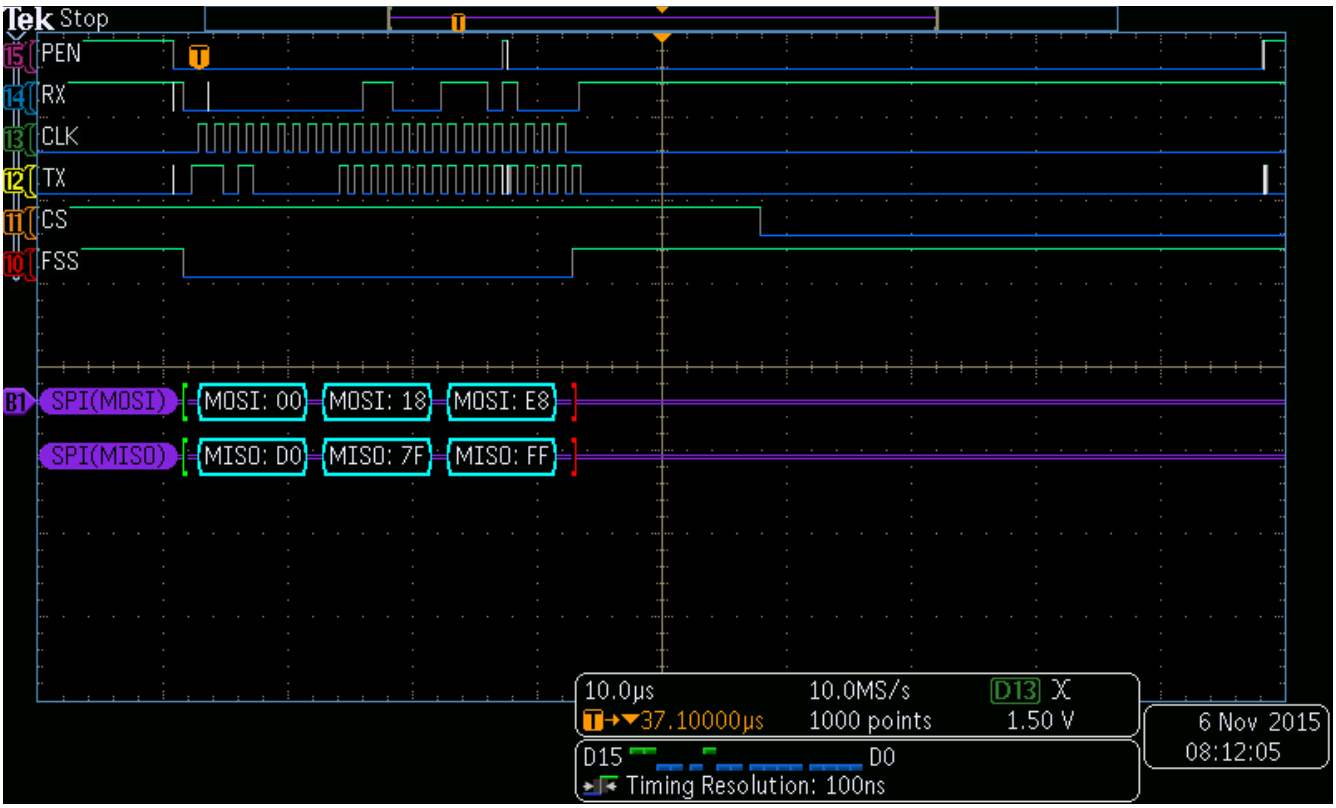


Touchscreen:

Upon being touched by the user, the touchscreen sent an interrupt request to the microcontroller, which then asked for X and Y coordinates from the user repeatedly until it stopped being touched. Here is a screenshot of the touchscreen sending those coordinates to the microcontroller. Towards the end, it is clear to see the data repeats itself for the second X coordinate (as it should):



Below is a screenshot of a single coordinate being transmitted. The CSX (mislabeled) line represents when the receive FIFO is read. A faster configuration is probably possible, but this implementation worked just fine.



Conclusion:

The touchscreen is a little finicky and only works sometimes. The main reason for this is a poor design choice that allows the microcontroller to turn off the interrupt request by simply making the last bit written a logic '1', whether intentional or no. Otherwise, the response and accuracy of the touchscreen is quite high and although the refresh rate of the LCD is slow, it is still good quality.

Code:

The rest of this document contains only code. Here it is:

LCD.h

```
#ifndef LCD_H
#define LCD_H

// page 238 of the data sheet.
#define white         0xFFFF
#define black         0x0001
#define grey          0xF7DE
#define blue          0x001F
#define red           0xF800
#define magenta       0xF81F
#define green         0x07E0
#define cyan          0x7FFF
#define yellow        0xFFE0
#define ENDCOL        0x00EF          //total columns
#define ENDPAGE       0x013F          //total rows
#define OUTSQSC       0x52            //Outer square starting column (82)
#define OUTSQEC       0x9D            //Outer square ending column (157)
#define INSQSC        0x59            //Inner square starting column (92)
#define INSQEC        0x96            //Inner square ending column (147)
#define TOTALAREA     0x12C00         //total screen area
#define SQUAREAREA    0x15F9          //total area per square (75^2)
#define INNERSQAREA   0xE89           //total area for inner part of square (55^2)
#define LOWX                  0x578           //1400
#define HIGHX                 0xAF0           //2800
#define REDLY                 0x190           //400
#define REDHY                 0x3E8           //1000
#define GREENLY         0x514           //1300
#define GREENHY         0x7D0           //2000
#define YELLY                 0x8FC           //2300
#define YELHY                 0xBB8           //3000

extern unsigned char *SYSCTL;

extern unsigned char *PA;
extern unsigned char *PB;
extern unsigned char *PC;
extern unsigned char *PD;
extern unsigned char *PE;
extern unsigned char *PF;
extern unsigned char *CORE;
extern unsigned char *TM0;
extern unsigned char *TM1;
extern unsigned char *TM2;
```

```
extern unsigned char *SSI1;

void writeCmd(unsigned char CMD);
void writeDat(unsigned char DAT);
void writeDat2(unsigned short DAT);
void writeDat4(unsigned int DAT);

void setArea(unsigned short x1, unsigned short x2, unsigned short y1, unsigned short y2);
void RGB(unsigned char RED, unsigned char GREEN, unsigned char BLUE);
void writeColor(unsigned short color);

void LCD_Init(void);
void drawSizeColor(unsigned int AREA, unsigned char RED, unsigned char GREEN, unsigned char
BLUE);

void drawRedSquare(void);
void drawGreenSquare(void);
void drawYelSquare(void);

void drawRedEmpty(void);
void drawGreenEmpty(void);
void drawYelEmpty(void);


#endif
```

LCD.c

```
#include "LCD.H"

void drawRedSquare(void)
{
  setArea(OUTSQSC, OUTSQEC, 0x1B, 0x66);          //27,102
  writeCmd(0x2C);
  drawSizeColor(SQUAREAREA, 0xFF, 0x00, 0x00);      //red
}

void drawGreenSquare(void)
{
  setArea(OUTSQSC, OUTSQEC, 0x7A, 0xC5);           //122,197
  writeCmd(0x2C);
  drawSizeColor(SQUAREAREA, 0x00, 0xFF, 0x00);      //green
}

void drawYelSquare(void)
{
  setArea(OUTSQSC, OUTSQEC, 0xD9, 0x124);          //217, 292
```

```
  writeCmd(0x2C);
  drawSizeColor(SQUAREAREA, 0xFF, 0xFF, 0x00);      //yellow
}

void drawRedEmpty(void)
{
  setArea(INSQSC, INSQEC, 0x22, 0x5F);                  // 34, 95
  drawSizeColor(INNERSQAREA, 0x00, 0x00, 0x01);      //black
}

void drawGreenEmpty(void)
{
  setArea(INSQSC, INSQEC, 0x81, 0xBE);              // 129, 190
  drawSizeColor(INNERSQAREA, 0x00, 0x00, 0x01);      //black
}

void drawYelEmpty(void)
{
  setArea(INSQSC, INSQEC, 0xE0, 0x11D);               // 224, 285
  drawSizeColor(INNERSQAREA, 0x00, 0x00, 0x01);      //black
}


void drawSizeColor(unsigned int Area, unsigned char RED, unsigned char GREEN, unsigned char
BLUE)
{
        unsigned int k;
        writeCmd(0x2C);
        for(k = 0; k < Area; k++)
        {
                RGB(RED, GREEN, BLUE);
        }
}

void writeCmd(unsigned char input)
{
        PA[0x3FC] &= 0xEF;                                      // D/CX = 0 -> command
        PA[0x3FC] &= 0xDF;                                      // WRX = 0 -> going to write
        PE[0x3FC] = input & 0x3;
        PD[0x3FC] = input & 0xF;
        PC[0x3FC] = input & 0xF0;
        PA[0x3FC] |= 0x20;                                  // WRX = 1 -> reads on pos edge

        PA[0x3FC] |= 0x10;                                  // D/CX = 1 -> default
}

void writeDat(unsigned char input) // 1 byte
{
        PA[0x3FC] &= 0xDF;                                      // WRX = 0 -> going to write
```

```
        PE[0x3FC] = input & 0x3;
        PD[0x3FC] = input & 0xF;
        PC[0x3FC] = input & 0xF0;
        PA[0x3FC] |= 0x20;                                  // WRX = 1 -> reads on pos edge

}

void RGB(unsigned char RED, unsigned char GREEN, unsigned char BLUE)
{
        unsigned short color = ((RED & 0x1F) << 11);
        color |= ((GREEN & 0x3F) << 5);
        color |= (BLUE & 0x1F);
        writeColor(color);
}

void writeColor(unsigned short color)
{
        writeDat((color & 0xFF00) >> 8);
        writeDat((color & 0xFF));
}

void setArea(unsigned short x1, unsigned short x2, unsigned short y1, unsigned short y2)
{
  writeCmd(0x2A);                                   //column address set, max 0xEF
        writeDat( (x1 >> 8) );    //SC MSB
        writeDat( (x1 & 0xFF) );  //SC LSB
        writeDat( (x2 >> 8) );    //EC MSB
        writeDat( (x2 & 0xFF) );  //EC LSB

        writeCmd(0x2B);                                 //page address set, max 0x13F
        writeDat( (y1 >> 8) );    //SP MSB
        writeDat( (y1 & 0xFF) );  //SP LSB
        writeDat( (y2 >> 8) );    //EP MSB
        writeDat( (y2 & 0xFF) );  //EP LSB
}

void LCD_Init()
{
        int i;
    writeCmd(0xCB);
    writeDat(0x39);
    writeDat(0x2C);
    writeDat(0x00);
    writeDat(0x34);
    writeDat(0x02);

    writeCmd(0xCF);
    writeDat(0x00);
    writeDat(0XC1);
```

```
writeDat(0X30);

writeCmd(0xE8);
writeDat(0x85);
writeDat(0x00);
writeDat(0x78);

writeCmd(0xEA);
writeDat(0x00);
writeDat(0x00);

writeCmd(0xED);
writeDat(0x64);
writeDat(0x03);
writeDat(0X12);
writeDat(0X81);

writeCmd(0xF7);
writeDat(0x20);

writeCmd(0xC0);    //Power control
writeDat(0x23);   //VRH[5:0]

writeCmd(0xC1);    //Power control
writeDat(0x10);   //SAP[2:0];BT[3:0]

writeCmd(0xC5);    //VCM control
writeDat(0x3e); //¶Ô±È¶Èµ÷½Ú
writeDat(0x28);

writeCmd(0xC7);    //VCM control2
writeDat(0x86);  //--

writeCmd(0x36);    // Memory Access Control
writeDat(0x48); //C8       //48 68ÊúÆÁ//28 E8 ºáÆÁ

writeCmd(0x3A);
writeDat(0x55);

writeCmd(0xB1);
writeDat(0x00);
writeDat(0x18);

writeCmd(0xB6);    // Display Function Control
writeDat(0x08);
writeDat(0x82);
writeDat(0x27);

writeCmd(0xF2);    // 3Gamma Function Disable
```

```
    writeDat(0x00);

    writeCmd(0x26);    //Gamma curve selected
    writeDat(0x01);

    writeCmd(0xE0);    //Set Gamma
    writeDat(0x0F);
    writeDat(0x31);
    writeDat(0x2B);
    writeDat(0x0C);
    writeDat(0x0E);
    writeDat(0x08);
    writeDat(0x4E);
    writeDat(0xF1);
    writeDat(0x37);
    writeDat(0x07);
    writeDat(0x10);
    writeDat(0x03);
    writeDat(0x0E);
    writeDat(0x09);
    writeDat(0x00);

    writeCmd(0XE1);    //Set Gamma
    writeDat(0x00);
    writeDat(0x0E);
    writeDat(0x14);
    writeDat(0x03);
    writeDat(0x11);
    writeDat(0x07);
    writeDat(0x31);
    writeDat(0xC1);
    writeDat(0x48);
    writeDat(0x08);
    writeDat(0x0F);
    writeDat(0x0C);
    writeDat(0x31);
    writeDat(0x36);
    writeDat(0x0F);

    writeCmd(0x11);    //Exit Sleep
                for( i = 0; i < 20000; i++) { i++;}

    writeCmd(0x29);    //Display on
}
```

Main.c:

```
// This function draws three empty boxes. The user touches the box
```

```
// to fill the box and turn on a corresponding external light.

#include "LCD.H"
#include "stdbool.h"

unsigned char *SYSCTL = (unsigned char *) 0x400FE000;
unsigned char *PA = (unsigned char *) 0x40004000;
unsigned char *PB = (unsigned char *) 0x40005000;
unsigned char *PC = (unsigned char *) 0x40006000;
unsigned char *PD = (unsigned char *) 0x40007000;
unsigned char *PE = (unsigned char *) 0x40024000;
unsigned char *PF = (unsigned char *) 0x40025000;
unsigned char *CORE = (unsigned char *) 0xE000E000;
unsigned char *TM0 = (unsigned char *) 0x40030000;
unsigned char *TM1 = (unsigned char *) 0x40031000;
unsigned char *TM2 = (unsigned char *) 0x40032000;
unsigned char *SSI1 = (unsigned char *) 0x40009000;
unsigned int *SSIDR = (unsigned int *) 0x40009008;

//globals for TP interrupts, modified in GPIOF_Handler
volatile unsigned int CNTX = 0;        //number of valid interrupts for x
volatile unsigned int CNTY = 0;                                        // number of valid interrupts for y
volatile unsigned int SUMX = 0;        //sum x coordinates of press
volatile unsigned int SUMY = 0;       //sum y coordinates of press
volatile bool input = false;

void GPIOF_Handler(void);

void enableClock(void);
void enableSSI1(void);
void enablePortA(void);
void enablePortB(void);
void enablePortC(void);
void enablePortD(void);
void enablePortE(void);
void enablePortF(void);
void configurePorts(void);

void flipRed(void);
void flipGreen(void);
void flipYel(void);

int main(void)
{
  volatile unsigned int xcoord;
  volatile unsigned int ycoord;
          unsigned int i;
  //SSI1[0x020] = 0x02;       // RTIC clear SSIICR... maybe
          configurePorts();
```

```
        LCD_Init();

        setArea(0x00, ENDCOL, 0x00, ENDPAGE);
  drawSizeColor(TOTALAREA, 0xFF, 0xFF, 0xFF);      //white
        drawSizeColor(TOTALAREA, 0x00, 0x00, 0x01);      //black
  //draw outlines
  drawRedSquare();
  drawGreenSquare();
  drawYelSquare();
  //default off
  drawRedEmpty();
  drawGreenEmpty();
  drawYelEmpty();

   while (1)
   {
                    //SSI1[0x008] = 0xD0;
                    if(input){
      input = false;
      xcoord = SUMX / CNTX;
      ycoord = SUMY / CNTY;

                                    if(LOWX < xcoord && xcoord < HIGHX){
                                            if(REDLY < ycoord && ycoord < REDHY){
                                                    flipRed();
                                            }
                                            else if(GREENLY < ycoord && ycoord < GREENHY){
                                                    flipGreen();
                                            }
                                            else if(YELLY < ycoord && ycoord < YELHY){
                                                    flipYel();
                                            }
                                    }
      SUMX = 0;
      SUMY = 0;
      CNTX = 0;
                                    CNTY = 0;

                                    for(i = 0; i < 0xFFFF; i++);
     }
   }
}

volatile unsigned short RXX = 0x0000;
volatile unsigned short RXY = 0x0000;

void GPIOF_Handler(void)
{
  unsigned char TX = 0xD0;                                  //x coordinate request
```

```
        unsigned j = 0;

        while((SSI1[0x00C] & 0x1) == 0x0);              //wait for transmit FIFO to be empty
        while((SSI1[0x00C] & 0x4) == 0x4){              //while receive FIFO is not empty
                RXX = SSIDR[0];
        }
  SSI1[0x008] = TX;             //ask for x coordinate
        for(j = 0; j < 0x1FF; j++);                                     // wait for receive FIFO

  RXX = SSIDR[0];   //read data
        RXX = (RXX & 0x7FFF) >> 3;                              //disregard first bit
        SUMX += RXX;
        CNTX++;

  TX = 0x90;      //y coordinate request
  while((SSI1[0x00C] & 0x2) == 0x0);            //wait TNF = 0; TNF = 1, continue
        while((SSI1[0x00C] & 0x4) == 0x4){
                RXY = SSIDR[0];                 //read data
        }
  SSI1[0x008] = TX;             //ask for x coordinate
  for(j = 0; j < 0x1FF; j++);                                   //wait for receive FIFO
        RXY = SSIDR[0];                 //read data
        RXY = (RXY & 0x7FFF) >> 3;                              //disregard first bit
        SUMY += RXY;
        CNTY++;

  input = true;
        PF[0x41C] |= 0x10;    //ack
        CORE[0x283] |= 0x40;        //unpend
}

void enableClock(){
        unsigned int i;
//  *(int*)&SYSCTL[0x060] = 0x8E3D40;           //sysclk 16MHz
  *(int*)&SYSCTL[0x060] = 0x14E1540;            //sysclk 66.67 MHz
  for(i = 0; i < 0xFF; i++);
        SYSCTL[0x608] = 0x3F;                           //enable PA PB PC PD PE PF
  SYSCTL[0x61C] = 0x02;       // enable SSI1

        __asm__ {nop};
        __asm__ {nop};
        __asm__ {nop};
}
void enableSSI1(void)
{
  SSI1[0x004] = 0x00;          // SSICR1 SSI disable
        SSI1[0xFC8] = 0x00;          // SSICC = SYSCLK
  SSI1[0x010] = 0x0A;          // SSICPSR (clk prescale) CPSDVSR = 2
  // SSICR0 SCR = 1 -> ~22.22 MHz / ( 2 * (1 + 1))
```

```
  *(int*)&SSI1[0x000] = 0x62E;          // SSICR0 Microwire, 16bits, SCR = 7
  SSI1[0x004] = 0x02;          // SSICR1 SSI enable
}

void configurePorts()
{
        enableClock();

        enablePortA();
        enablePortB();
        enablePortC();
        enablePortD();
        enablePortE();
  enablePortF();
  enableSSI1();

        PA[0x3FC] = 0x00;
        PB[0x3FC] = 0xFF;
        PC[0x3FC] = 0x00;
        PD[0x3FC] = 0x00;
        PE[0x3FC] = 0x00;

        PA[0x3FC] |= 0xC4;                              //RESET and RDX high
        PA[0x3FC] &= 0xF7;                      // CSX chip select low
        PB[0x3FC] = 0x07;                               // LEDs active low, TP_CS low

        //interrupts
  PF[0x404] = 0x10;             // PF4 level sensitive GPIOIS
  PF[0x40C] = 0x00;              // low level GPIOIEV
  PF[0x410] = 0x10;             // PF4 GPIOIM
        CORE[0x103] = 0x40;                     // enable interrupts PF
}

void enablePortA(void)
{
        PA[0x420] = 0x00;                               //AFSEL
        PA[0x51C] = 0xFF;                               //DEN
        PA[0x400] = 0xFC;                               //DIR
        PA[0x514] = 0xFF;                       //PDR
}

void enablePortB(void)
{
        PB[0x420] = 0x00;                               //AFSEL
        PB[0x51C] = 0xFF;                               //DEN
        PB[0x400] = 0xFF;                               //DIR
        PB[0x514] = 0xFF;                               //PDR
}
```

```
void enablePortC(void)
{
	PC[0x51C] = 0xFF;                                   //DEN
	PC[0x400] = 0xFF;                                   //DIR
	PC[0x514] = 0xFF;                                   //PDR
}

void enablePortD(void)
{
	*(int*)&PD[0x520] = 0x4C4F434B;                             //GPIO Unlcok
	PD[0x524] = 0x80;
	//Commit register

	PD[0x51C] = 0xFF;                                   //DEN
	PD[0x400] = 0xFF;                                   //DIR
	PD[0x514] = 0xFF;                                   //PDR
}

void enablePortE(void)
{
	PE[0x51C] = 0xFF;                                   //DEN
	PE[0x400] = 0xFF;                                   //DIR
	PE[0x514] = 0xFF;                                   //PDR
}


void enablePortF(void)
{
  *(int*)&PF[0x520] = 0x4C4F434B;                             //GPIO Unlock
	PF[0x524] = 0x01;
	//Commit register

  PF[0x420] = 0x0F;                 //AFSEL
	PF[0x51C] = 0x1F;                                     //DEN
	PF[0x400] = 0x0E;                                   //DIR - PF0 input
	PF[0x510] = 0x1D;                                                                   //PUR
	PF[0x514] = 0x02;                                   //PDR
//	PF[0x50C] = 0x02;
  *(int*)&PF[0x52C] = 0x00002222;     //PCTL - SSI1
}

void flipRed(void)
{
	//PB0
	unsigned char val = (PB[0x3FC] & 0x1);
	if(val == 0x1){
		drawRedSquare();
		PB[0x3FC] &= 0xFE;
	}
```

```
        else{
                drawRedEmpty();
                PB[0x3FC] |= 0x1;
        }
}
void flipGreen(void)
{
        //PB1
        unsigned char val = (PB[0x3FC] & 0x2);
        if(val == 0x2){
                drawGreenSquare();
                PB[0x3FC] &= 0xFD;
        }
        else{
                drawGreenEmpty();
                PB[0x3FC] |= 0x2;
        }
}
void flipYel(void)
{
        //PB2
        unsigned char val = (PB[0x3FC] & 0x4);
        if(val == 0x4){
                drawYelSquare();
                PB[0x3FC] &= 0xFB;
        }
        else{
                drawYelEmpty();
                PB[0x3FC] |= 0x4;
        }
}


// PA2 - RST
// PA3 - CSX
// PA4 - D/CX
// PA5 - WRX
// PA6 - RDX
// PA7 - Backlight

// PB0 - Red LED
// PB1 - Green LED
// PB2 - Blue LED
// PB3 - TP_CS pin 30

// PF - SSI1
// PF0 - Rx - SDO pin 32
// PF1 - Tx - SDI pin 34
// PF2 - Clk - CLK pin 33
```

```
// PF3 - Fss - TP_CS pin 30
// PF4 - PENIRQ - IRQ pin 31
```