## ECEN260 – FINAL PROJECT

## TIC TAC TOE

**Explanation of the project:**

This project will simulate the game TIC TAC TOE. It will use **I/O, interrupt, and Display**.

It will display the content of the game on the screen by coping the pixel patterns into the display memory. After that the pixel patterns are stored in an array in the same way that they will be written to the GLCD memory. (As instructed in LAB 10)

It will use an interrupt signal to notify the MSP432 that a key on the keypad has been pressed. The program will then respond according to the key that is pressed, allowing the users to play the game.

The user will use the following configuration to play

Diagram

Description automatically generated

Sw1 will restart the game

Numbers 1-9 will be used to choose the box of the game board where the users want to play.

“A” key will be used to start the game

When the program starts, P2.0, P2.1, P2.2 are set as outputs. These are the Red, Green, and Blue LEDS from port2. The RED light indicates that the program is in the start Menu, ready to start. Blue LED indicates that is player 1’s turn to play, and Green LED indicates 2nd’s player turn. When someone wins the game, the three LEDS turn on.

The program uses P4.0, P4.1, P4.2, P4.3, P1.1, and P3.0 as inputs. The first 4 are for the input from the keypad controller. P1.1 is the input for the Switch 1 button. P1.1 and P3 (DA (used for the keypad) will use pull-up resistors and interrupts as well.

According to the different keys that the user press or the Sw1 being pressed, the program will send the interrupt signal and we will send the data to the respective output’s ports. SW1 will restart the game anytime when is being pressed, and the keys will make the program to display on the screen the choices of the users and turn on or off the LEDs (from P2) as is needed.

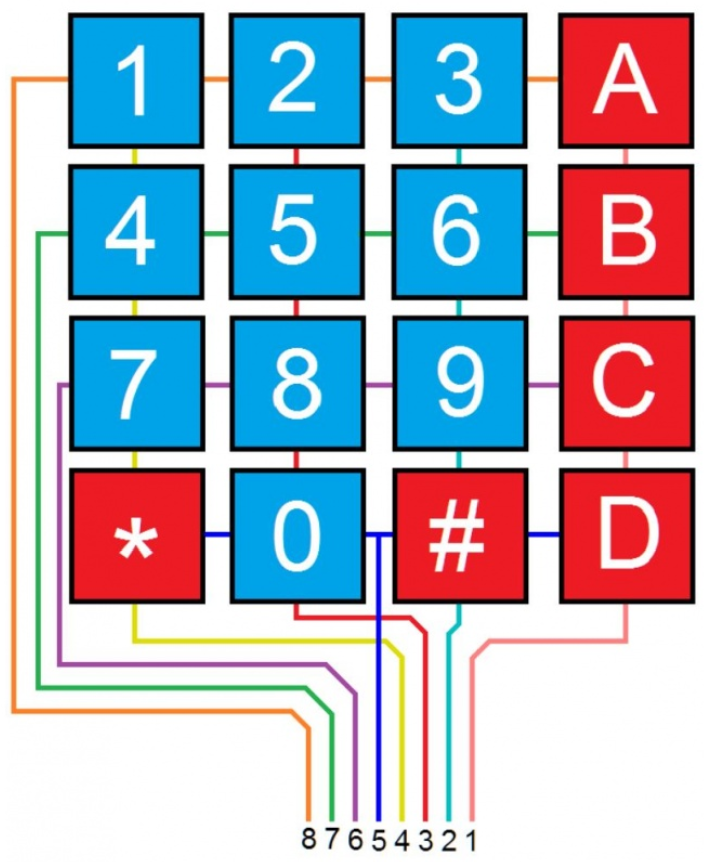
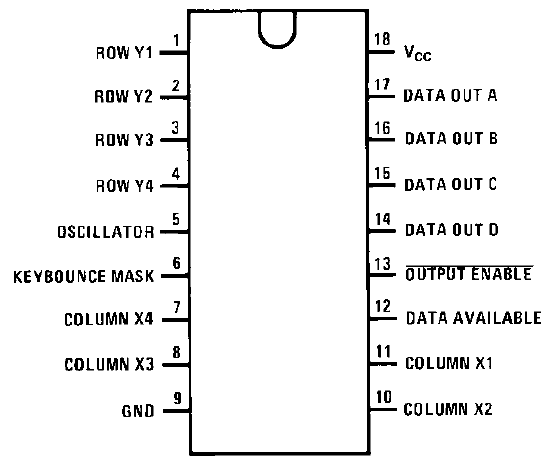
The first thing the program will display on the screen will be the message “TIC TAC TOE PRESS A TO START”. If one of the users wins the program will display a message saying “1st/2nd player wins. The program will display the message “DRAW” if nobody wins.

**Schematics/Wiring diagram**

**MSP 432 connected to the Keypad and Screen.**



**Keypad Connection with 74c922 16 key encoder IC**



**Screen with game running  
 A picture containing text, appliance

Description automatically generated**

**TEST PLAN**

1.- When the program starts running, I am going to press any key (except A) to see if something happens. Nothing should have until I press A and the game should start.

2 Once the game is running, I will press a number between 1-9 to see if the program draws an X in the box of the game chart that I chose. Then the light should change color from blue to green (announcing second player’s turn) and I am going to press another key and a circle should draw on the box selected. Blue light should change to green again indicating first player’s turn again.

3 I am going to make player 1 win (following the rules of the original tic tac toe game grouping 3 figures of the same kind vertical, horizontal, or diagonal) and a 1st player wins message should appear. If I do the same with player 2, the respective message should appear as well.

4 Finally, I will verify the draw function when nobody wins to see if it displays its message, and I am going to press SW1, and the game should restart.

**Code**

**#include** "msp.h"

**#define** CE 0x01 /\* P6.0 chip select \*/

**#define** RESET 0x40 /\* P6.6 reset \*/

**#define** DC 0x80 /\* P6.7 register select \*/

/\* define the pixel size of display \*/

**#define** GLCD\_WIDTH 84

**#define** GLCD\_HEIGHT 48

**#define** LED2RED BIT0

**#define** LED2GREEN BIT1

**#define** LED2BLUE BIT2

**#define** S1 BIT1

**int** i = 0;

**int** newGame = 0;

**int** player = 0;

**int** espacios[9];

//DISPLAY FUNCTIONS

**void** **GLCD\_setCursor**(**unsigned** **char** x, **unsigned** **char** y);

**void** **GLCD\_clear**(**void**);

**void** **GLCD\_init**(**void**);

**void** **GLCD\_data\_write**(**unsigned** **char** data);

**void** **GLCD\_command\_write**(**unsigned** **char** data);

**void** **GLCD\_putchar**(**int** c);

**void** **SPI\_init**(**void**);

**void** **SPI\_write**(**unsigned** **char** data);

//GAME FUNTIONS

**void** **squareOneTop**(**int** a);

**void** **squareOneBottom**(**int** a);

**void** **squareTwoTop**(**int** a);

**void** **squareTwoBottom**(**int** a);

**void** **squareThreeTop**(**int** a);

**void** **squareThreeBottom**(**int** a);

**void** **squareSevenBottom**(**int** a);

**void** **squareEightBottom**(**int** a);

**void** **squareNineBottom**(**int** a);

**void** **table**(**int** a, **int** b, **int** c, **int** d, **int** e, **int** f, **int** g, **int** h, **int** i);

**void** **zeros**(**void**);

**void** **zerosBottom**(**void**);

**void** **xTop**(**void**);

**void** **xBottom**(**void**);

**void** **xLast**(**void**);

**void** **cTop**(**void**);

**void** **cBottom**(**void**);

**void** **cLast**(**void**);

**void** **startMenu**(**void**);

**void** **checkWin**(**void**);

**void** **playerTwoWin**(**void**);

**void** **playerOneWin**(**void**);

**void** **draw**(**void**);

/\* sample font table \*/

**const** **char** font\_table[][6] = {

{0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, /\* -0 \*/

{0x7e, 0x11, 0x11, 0x11, 0x7e, 0x00}, /\* A-1 \*/

{0x7f, 0x49, 0x49, 0x49, 0x36, 0x00}, /\* B-2 \*/

{0x3e, 0x41, 0x41, 0x41, 0x22, 0x00}, /\* C-3 \*/

{0x7f, 0x41, 0x41, 0x41, 0x3e, 0x00}, /\* D-4 \*/

{0x7f, 0x49, 0x49, 0x49, 0x41, 0x00}, /\* E-5 \*/

{0x7f, 0x09, 0x09, 0x09, 0x01, 0x00}, /\* F-6 \*/

{0x3e, 0x41, 0x49, 0x49, 0x7a, 0x00}, /\* G-7 \*/

{0x7f, 0x08, 0x08, 0x08, 0x7f, 0x00}, /\* H-8 \*/

{0x41, 0x41, 0x7f, 0x41, 0x41, 0x00}, /\* I-9 \*/

{0x20, 0x40, 0x40, 0x40, 0x3f, 0x00}, /\* J-10 \*/

{0x7f, 0x08, 0x14, 0x22, 0x41, 0x00}, /\* K-11 \*/

{0x7f, 0x40, 0x40, 0x40, 0x40, 0x00}, /\* L-12 \*/

{0x7f, 0x02, 0x0c, 0x02, 0x7f, 0x00}, /\* M-13 \*/

{0x7f, 0x04, 0x08, 0x10, 0x7f, 0x00}, /\* N-14 \*/

{0x3e, 0x41, 0x41, 0x41, 0x3e, 0x00}, /\* O-15 \*/

{0x7f, 0x09, 0x09, 0x09, 0x06, 0x00}, /\* P-16 \*/

{0x3e, 0x41, 0x51, 0x61, 0x7e, 0x00}, /\* Q-17 \*/

{0x7f, 0x09, 0x19, 0x29, 0x46, 0x00}, /\* R-18 \*/

{0x26, 0x49, 0x49, 0x49, 0x32, 0x00}, /\* S-19 \*/

{0x01, 0x01, 0x7f, 0x01, 0x01, 0x00}, /\* T-20 \*/

{0x3f, 0x40, 0x40, 0x40, 0x3f, 0x00}, /\* U-21 \*/

{0x1f, 0x20, 0x40, 0x20, 0x1f, 0x00}, /\* V-22 \*/

{0x3f, 0x40, 0x38, 0x40, 0x3f, 0x00}, /\* W-23 \*/

{0x63, 0x14, 0x08, 0x14, 0x63, 0x00}, /\* X-24 \*/

{0x03, 0x04, 0x78, 0x04, 0x03, 0x00}, /\* Y-25 \*/

{0x61, 0x51, 0x49, 0x45, 0x43, 0x00}, /\* Z-26 \*/

{0x00, 0x00, 0x5f, 0x00, 0x00, 0x00}, /\* !-27 \*/

{0x00, 0x00, 0x7e, 0x81, 0xb5, 0xa1}, /\* Smile1-28 \*/

{0xa1, 0xb5, 0x81, 0x7e, 0x00, 0x00}, /\* Smile2-29 \*/

{0x00, 0x00, 0xff, 0xff, 0x00, 0x00}, /\* | 30 \*/

{0x80, 0x80, 0x80, 0x80, 0x80, 0x80}, // \_ 31

{0x00, 0x04, 0x08, 0x10, 0x20, 0x40}, //32

{0x80, 0x40, 0x20, 0x10, 0x08, 0x04}, //33

{0x80, 0x90, 0x88, 0x84, 0x82, 0x81}, //34

{0x80, 0x81, 0x82, 0x84, 0x88, 0x90}, //35

{0x00, 0x10, 0x08, 0x04, 0x02, 0x01}, //36

{0x00, 0x01, 0x02, 0x04, 0x08, 0x10}, //37

{0x80, 0x80, 0x8f, 0x90, 0x90, 0x90}, //38

{0x90, 0x90, 0x90, 0x90, 0x90, 0x8f}, //39

{0x00, 0x00, 0xf8, 0x04, 0x04, 0x04}, //40

{0x04, 0x04, 0x04, 0x04, 0x04, 0xf8}, //41

{0x00, 0x00, 0x0f, 0x10, 0x10, 0x10}, //42

{0x10, 0x10, 0x10, 0x10, 0x10, 0x0f}, //43

};

/\*\*

\* main.c

\*/

**int** **main**(**void**)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

GLCD\_init(); /\* initialize the GLCD controller \*/

GLCD\_clear(); /\* clear display and home the cursor \*/

P2->DIR |= (BIT0|BIT1|BIT2); /\* set up pins P2.0, P2.1, P2.2 (R, G, and B LEDs) as output \*/

P2->OUT &= ~(BIT0|BIT1|BIT2); /\* turn off RGB LED \*/

P4->DIR &= ~(BIT0|BIT1|BIT2|BIT3); /\* set up pins P4.0, P4.1, P4.2, P4.3 (pins for the

input from the keypad controller) as input \*/

P1->DIR &= ~BIT1; /\* set up pin P1.1 (S1) as input \*/

P1->REN |= BIT1; /\* connect pull resistor to pin P1.1 (S1) \*/

P1->OUT |= BIT1; /\* configure pull resistor as pull up \*/

P1->IFG &= ~BIT1; /\* clear the interrupt flag for pin P1.1 (S1) \*/

P1->IE |= BIT1; /\* enable the interrupt for pin P1.1 (S1) \*/

P3->DIR &= ~BIT0; /\* set up pin P3.0 as input \*/

P3->REN |= BIT0; /\* connect pull resistor to pin P3.0 \*/

P3->OUT |= BIT0; /\* configure pull resistor as pull up \*/

P3->IFG &= ~BIT0; /\* clear interrupt flag for pin P3.0 (DA (int pin)) \*/

P3->IE |= BIT0; /\* enable the interrupt for pin P3.0 (DA (interrupt pin)) \*/

NVIC->ISER[1] |= 0x20; /\* enable port 3 interrupts\*/

NVIC->ISER[1] |= 0x08; /\* enable port 1 interrupts (Table 6.2 in text)\*/

\_enable\_interrupts();

startMenu();

newGame = 1;

P2->OUT |= (LED2RED);

**while** (1); /\* wait for an interrupt \*/

}

/\*\*

\* Display chart Tic Tac Toe

\*/

**void** **table**(**int** a, **int** b, **int** c, **int** d, **int** e,

**int** f, **int** g, **int** h, **int** i)

{

squareOneTop(a); squareTwoTop(b); squareThreeTop(c);

squareOneBottom(a); squareTwoBottom(b); squareThreeBottom(c);

squareOneTop(d); squareTwoTop(e); squareThreeTop(f);

squareOneBottom(d); squareTwoBottom(e); squareThreeBottom(f);

squareOneTop(g); squareTwoTop(h); squareThreeTop(i);

squareSevenBottom(g); squareEightBottom(h); squareNineBottom(i);

}

/\*\*

\* Check if there is a winner

\*/

**void** **checkWin**()

{

**int** i;

**int** symbol;

**for** (i = 1; i < 3; i++)

{ //Check if we have three equals symbols in column, row, or diagonal

**if** (((espacios[0] == i) && (espacios[1] == i) && (espacios[2] == i)) ||

((espacios[3] == i) && (espacios[4] == i) && (espacios[5] == i)) ||

((espacios[6] == i) && (espacios[7] == i) && (espacios[8] == i)) ||

((espacios[0] == i) && (espacios[3] == i) && (espacios[6] == i)) ||

((espacios[1] == i) && (espacios[4] == i) && (espacios[7] == i)) ||

((espacios[2] == i) && (espacios[5] == i) && (espacios[8] == i)) ||

((espacios[0] == i) && (espacios[4] == i) && (espacios[8] == i)) ||

((espacios[2] == i) && (espacios[4] == i) && (espacios[6] == i)))

symbol = i;

}

**if** (symbol == 1)

playerOneWin();

**else** **if** (symbol == 2)

playerTwoWin();

//Check if all the spots are filled on the chart

**if**((espacios[0] != 0) && (espacios[1] != 0) && (espacios[2] != 0) &&

(espacios[3] != 0) && (espacios[4] != 0) && (espacios[5] != 0) &&

(espacios[6] != 0) && (espacios[7] != 0) && (espacios[8] != 0) &&

newGame == 0)

draw();

}

/\*\*

\* Display Message Player 1 Win

\*/

**void** **playerOneWin**()

{

//Print the start message

**int** i;

**for**(i=0; i<16; i++){GLCD\_putchar(0);}

GLCD\_putchar(16); //P

GLCD\_putchar(12); //L

GLCD\_putchar(1); //A

GLCD\_putchar(25); //Y

GLCD\_putchar(5); //E

GLCD\_putchar(18); //R

GLCD\_putchar(0);

GLCD\_putchar(15); //O

GLCD\_putchar(14); //N

GLCD\_putchar(5); //E

**for**(i=0; i<7; i++){GLCD\_putchar(0);}

GLCD\_putchar(23); //W

GLCD\_putchar(9); //I

GLCD\_putchar(14); //N

GLCD\_putchar(19); //S

**for**(i=0; i<47; i++){GLCD\_putchar(0);}

P2->OUT |= (LED2GREEN|LED2RED|LED2BLUE);

newGame = 3;

}

/\*\*

\* Display Message Player 2 Win

\*/

**void** **playerTwoWin**()

{

//Print the start message

**int** i;

**for**(i=0; i<16; i++){GLCD\_putchar(0);}

GLCD\_putchar(16); //P

GLCD\_putchar(12); //L

GLCD\_putchar(1); //A

GLCD\_putchar(25); //Y

GLCD\_putchar(5); //E

GLCD\_putchar(18); //R

GLCD\_putchar(0);

GLCD\_putchar(20); //T

GLCD\_putchar(23); //W

GLCD\_putchar(15); //O

**for**(i=0; i<7; i++){GLCD\_putchar(0);}

GLCD\_putchar(23); //W

GLCD\_putchar(9); //I

GLCD\_putchar(14); //N

GLCD\_putchar(19); //S

**for**(i=0; i<47; i++){GLCD\_putchar(0);}

P2->OUT |= (LED2GREEN|LED2RED|LED2BLUE);

newGame = 3;

}

/\*\*

\* Display Message Draw

\*/

**void** **draw**()

{

//Print the start message

**int** i;

**for**(i=0; i<33; i++){GLCD\_putchar(0);}

GLCD\_putchar(4); //D

GLCD\_putchar(18); //R

GLCD\_putchar(1); //A

GLCD\_putchar(23); //W

**for**(i=0; i<47; i++){GLCD\_putchar(0);}

P2->OUT |= (LED2GREEN|LED2RED|LED2BLUE);

newGame = 3;

}

/\*\*\*

\* IRQ handler for port 1

\*\*\*/

**void** **PORT1\_IRQHandler**(**void**)

{

uint32\_t status;

status = P1->IFG; /\* get the interrupt status for port 1 \*/

P1->IFG &= ~BIT1; /\* clear the interrupt for port 1, pin 1 \*/

**if**(status & BIT1)

{ /\* if SW was pressed: Game restart\*/

P2->OUT |= LED2RED; //turn on red LED at pin P2.0 instructions

P2->OUT &= ~(LED2GREEN|LED2BLUE); /\* turn off Players Lights\*/

startMenu(); //display menu message

newGame = 1;

}

}

/\*\*\*

\* IRQ handler for port 3

\*\*\*/

**void** **PORT3\_IRQHandler**(**void**)

{

uint32\_t status;

uint8\_t key = 0;

status = P3->IFG; /\* get the interrupt status for port 3 \*/

P3->IFG &= ~BIT0; /\* clear the interrupt for port 3, pin 0 \*/

**if**(status & BIT0)

{ /\* if any key was pressed \*/

key = keypad\_decode(); /\* determine which key was pressed \*/

**if**(key == 0xA)

{

**if**(newGame == 1) //check if "A" was pressed on the menu

{

P2->OUT &= ~(LED2RED); // turn off start red led

P2->OUT |= LED2BLUE; // turn on blue Led 1st player's turn

table(0,0,0,0,0,0,0,0,0); // show empty chart

**for**(i = 0; i<9;i++)

espacios[i] = 0; // no "X" or "O" saved

player = 1; //player 1 starts

newGame = 0;

}

}

**if** (newGame == 0) // if we are not on the start menu

{

**for**(i = 0; i < 9; i++)

{

**char** h = i+1;

**if**(key == h) //check which number is being pressed

{

**if**(espacios[i] == 0) //check if the space is empty to use

{

**if** (player == 1)

{

espacios[i] = 1; // fill space with an "X"

player = 2; // Now is player 2's turn

P2->OUT &= ~(LED2BLUE); // Turn off BLUE LED

P2->OUT |= LED2GREEN; // Announce next turn player 2

// Turn on GREEN LED

}

**else** **if**(player == 2)

{

espacios[i] = 2; // fill space with "O"

player = 1; // now is player 2's turn

P2->OUT &= ~(LED2GREEN); // Turn off GREEN LED

P2->OUT |= LED2BLUE; // Announce next turn player 1

// Turn on BLUE LIGHT

}

}

table(espacios[0], espacios[1], espacios[2], espacios[3], espacios[4],

espacios[5], espacios[6], espacios[7], espacios[8]); //show updated chart

checkWin(); //Check if someone wins

}

}

}

}

}

/\*\*\*

\* keypad decoder function

\*\*\*/

uint8\_t **keypad\_decode**(){

uint8\_t key = 0;

uint8\_t port = 0;

port += P4->IN & BIT0; /\* input value from pin P4.0 \*/

port += P4->IN & BIT1; /\* input value from pin P4.1 \*/

port += P4->IN & BIT2; /\* input value from pin P4.2 \*/

port += P4->IN & BIT3; /\* input value from pin P4.3 \*/

**switch**(port){

**case** 0x0D: key = 0x0; **break**; /\* 0 \*/

**case** 0x00: key = 0x1; **break**; /\* 1 \*/

**case** 0x01: key = 0x2; **break**; /\* 2 \*/

**case** 0x02: key = 0x3; **break**; /\* 3 \*/

**case** 0x04: key = 0x4; **break**; /\* 4 \*/

**case** 0x05: key = 0x5; **break**; /\* 5 \*/

**case** 0x06: key = 0x6; **break**; /\* 6 \*/

**case** 0x08: key = 0x7; **break**; /\* 7 \*/

**case** 0x09: key = 0x8; **break**; /\* 8 \*/

**case** 0x0A: key = 0x9; **break**; /\* 9 \*/

**case** 0x03: key = 0xA; **break**; /\* A \*/

**case** 0x07: key = 0xB; **break**; /\* B \*/

**case** 0x0B: key = 0xC; **break**; /\* C \*/

**case** 0x0F: key = 0xD; **break**; /\* D \*/

**case** 0x0C: key = 0xE; **break**; /\* \* \*/

**case** 0x0E: key = 0xF; **break**; /\* # \*/

}

**return** key;

}

////////////////////////////////////////////////////////

//DRAWING OF EACH SQUARE ACCORDING THE INPUT RECEIVED///

////////////////////////////////////////////////////////

**void** **squareOneTop**(**int** a)

{

**if** (a == 0){ zeros(); GLCD\_putchar(30);} // empty top of square

**else** **if**(a == 1){ xTop(); GLCD\_putchar(30);} // draw top of an X

**else** **if**(a == 2){ cTop(); GLCD\_putchar(30);} // draw top of a circle

}

**void** **squareOneBottom**(**int** a)

{

**if** (a == 0){ zerosBottom(); GLCD\_putchar(30);} // empty bottom of square

**else** **if**(a == 1){ xBottom(); GLCD\_putchar(30);} // draw bottom of X

**else** **if**(a == 2){ cBottom(); GLCD\_putchar(30);} // draw bottom of circle

}

**void** **squareTwoTop**(**int** a)

{

**if** (a == 0){ zeros(); GLCD\_putchar(30);} // empty top of square

**else** **if**(a == 1){ xTop(); GLCD\_putchar(30);} // draw top of an X

**else** **if**(a == 2){ cTop(); GLCD\_putchar(30);} // draw top of a circle

}

**void** **squareTwoBottom**(**int** a)

{

**if** (a == 0){ zerosBottom(); GLCD\_putchar(30);} // empty bottom of square

**else** **if**(a == 1){ xBottom(); GLCD\_putchar(30);} // draw bottom of X

**else** **if**(a == 2){ cBottom(); GLCD\_putchar(30);} // draw bottom of circle

}

**void** **squareThreeTop**(**int** a)

{

**if** (a == 0){ zeros();} // empty top of square

**else** **if**(a == 1){ xTop();} // draw top of an X

**else** **if**(a == 2){ cTop();} // draw top of a circle

}

**void** **squareThreeBottom**(**int** a)

{

**if** (a == 0){ zerosBottom();} // empty bottom of square

**else** **if**(a == 1){ xBottom();} // draw bottom of X

**else** **if**(a == 2){ cBottom();} // draw bottom of circle

}

**void** **squareSevenBottom**(**int** a)

{

**if** (a == 0) { zeros(); GLCD\_putchar(30);} // empty bottom of square

**else** **if**(a == 1){ xLast(); GLCD\_putchar(30);} // draw bottom of X

**else** **if**(a == 2){ cLast(); GLCD\_putchar(30);} // draw bottom of circle

}

**void** **squareEightBottom**(**int** a)

{

**if** (a == 0){ zeros(); GLCD\_putchar(30);} // empty bottom of square

**else** **if**(a == 1){ xLast(); GLCD\_putchar(30);} // draw bottom of X

**else** **if**(a == 2){ cLast(); GLCD\_putchar(30);} // draw bottom of circle

}

**void** **squareNineBottom**(**int** a)

{

**if** (a == 0){ zeros();} // empty bottom of square

**else** **if**(a == 1){ xLast();} // draw bottom of X

**else** **if**(a == 2){ cLast();} // draw bottom of circle

}

**void** **zeros**(){GLCD\_putchar(0);GLCD\_putchar(0);GLCD\_putchar(0);GLCD\_putchar(0);} //GLCD configuration for empty square

**void** **zerosBottom**(){GLCD\_putchar(31);GLCD\_putchar(31);GLCD\_putchar(31);GLCD\_putchar(31);} //GLCD configuration for empty square

**void** **xTop**(){GLCD\_putchar(0);GLCD\_putchar(32);GLCD\_putchar(33);GLCD\_putchar(0);} //GLCD configuration for half of X

**void** **xBottom**(){GLCD\_putchar(31);GLCD\_putchar(34);GLCD\_putchar(35);GLCD\_putchar(31);} //GLCD configuration for half of x

**void** **xLast**(){GLCD\_putchar(0);GLCD\_putchar(36);GLCD\_putchar(37);GLCD\_putchar(0);} //GLCD configuration for last row with x

**void** **cTop**(){GLCD\_putchar(0);GLCD\_putchar(40);GLCD\_putchar(41);GLCD\_putchar(0);} //GLCD configuration for half circle

**void** **cBottom**(){GLCD\_putchar(31);GLCD\_putchar(38);GLCD\_putchar(39);GLCD\_putchar(31);} //GLCD configuration for half of circle

**void** **cLast**(){GLCD\_putchar(0);GLCD\_putchar(42);GLCD\_putchar(43);GLCD\_putchar(0);} //GLCD configuration for last row with circle

**void** **startMenu**()

{

//Print the start message

**int** i;

**for**(i=0; i<16; i++){GLCD\_putchar(0);}

GLCD\_putchar(20); //T

GLCD\_putchar(9); //I

GLCD\_putchar(3); //C

GLCD\_putchar(0);

GLCD\_putchar(20); //T

GLCD\_putchar(1); //A

GLCD\_putchar(3); //C

GLCD\_putchar(0);

GLCD\_putchar(20); //T

GLCD\_putchar(15); //O

GLCD\_putchar(5); //E

**for**(i=0; i<5; i++){GLCD\_putchar(0);}

GLCD\_putchar(16); //P

GLCD\_putchar(18); //R

GLCD\_putchar(5); //E

GLCD\_putchar(19); //S

GLCD\_putchar(19); //S

GLCD\_putchar(0);

GLCD\_putchar(1); //A

**for**(i=0; i<6; i++){GLCD\_putchar(0);}

GLCD\_putchar(20); //T

GLCD\_putchar(15); //O

GLCD\_putchar(0);

GLCD\_putchar(0);

GLCD\_putchar(19); //S

GLCD\_putchar(20); //T

GLCD\_putchar(1); //A

GLCD\_putchar(18); //R

GLCD\_putchar(20); //T

**for**(i=0; i<30; i++){GLCD\_putchar(0);}

}

**void** **GLCD\_putchar**(**int** c)

{

**int** i;

**for**(i = 0; i < 6; i++)

GLCD\_data\_write(font\_table[c][i]);

}

**void** **GLCD\_setCursor**(**unsigned** **char** x, **unsigned** **char** y)

{

GLCD\_command\_write(0x80 | x); /\* column \*/

GLCD\_command\_write(0x40 | y); /\* bank (8 rows per bank) \*/

}

/\* clears the GLCD by writing zeros to the entire screen \*/

**void** **GLCD\_clear**(**void**)

{

int32\_t index;

**for**(index = 0; index < (GLCD\_WIDTH \* GLCD\_HEIGHT / 8); index++)

GLCD\_data\_write(0x00);

GLCD\_setCursor(0, 0); /\* return to the home position \*/

}

/\* send the initialization commands to PCD8544 GLCD controller \*/

**void** **GLCD\_init**(**void**)

{

SPI\_init();

/\* hardware reset of GLCD controller \*/

P6->OUT |= RESET; /\* deasssert reset \*/

GLCD\_command\_write(0x21); /\* set extended command mode \*/

GLCD\_command\_write(0xB8); /\* set LCD Vop for contrast \*/

GLCD\_command\_write(0x04); /\* set temp coefficient \*/

GLCD\_command\_write(0x14); /\* set LCD bias mode 1:48 \*/

GLCD\_command\_write(0x20); /\* set normal command mode \*/

GLCD\_command\_write(0x0C); /\* set display normal mode \*/

}

/\* write to GLCD controller data register \*/

**void** **GLCD\_data\_write**(**unsigned** **char** data)

{

P6->OUT |= DC; /\* select data register \*/

SPI\_write(data); /\* send data via SPI \*/

}

/\* write to GLCD controller command register \*/

**void** **GLCD\_command\_write**(**unsigned** **char** data)

{

P6->OUT &= ~DC; /\* select command register \*/

SPI\_write(data); /\* send data via SPI \*/

}

**void** **SPI\_init**(**void**)

{

EUSCI\_B0->CTLW0 = 0x0001; /\* put UCB0 in reset mode \*/

EUSCI\_B0->CTLW0 = 0x69C1; /\* PH=0, PL=1, MSB first, Master, SPI, SMCLK \*/

EUSCI\_B0->BRW = 3; /\* 3 MHz / 3 = 1MHz \*/

EUSCI\_B0->CTLW0 &= ~0x001; /\* enable UCB0 after config \*/

P1->SEL0 |= 0x60; /\* P1.5, P1.6 for UCB0 \*/

P1->SEL1 &= ~0x60;

P6->DIR |= (CE | RESET | DC); /\* P6.7, P6.6, P6.0 set as output \*/

P6->OUT |= CE; /\* CE idle high \*/

P6->OUT &= ~RESET; /\* assert reset \*/

}

**void** **SPI\_write**(**unsigned** **char** data)

{

P6->OUT &= ~CE; /\* assert /CE \*/

EUSCI\_B0->TXBUF = data; /\* write data \*/

**while**(EUSCI\_B0->STATW & 0x01);/\* wait for transmit done \*/

P6->OUT |= CE; /\* deassert /CE \*/

}

**TEST RESULTS**

I tested the program as I stated in the Test Plan and it works properly. When I press the “A” key the program starts if the game is in the start menu. If I choose a square the program display on the screen the respective figure (circle or X depending on whose turn it is) and the light indicating which player has to choose works as indicated. The message when a player wins or if there is a draw also works as expected.

The hardest part of the program was to create the draw functions. I had to create different drawings for each situation and make them work. Making the X or 0 to appear on the screen were with no doubt the hardest part, I had to rewrite the whole square every time to make it seem as if a circle or x appeared in the middle of the chosen box. I struggle with the interrupts as well, because when I pressed some keys that were not supposed to be used for the game in a certain moment, the programmed received signal from them and it had an effect on the game, but I could fix it with if statements.

Here is the video

[**https://youtu.be/5uoCrtbMvqE**](https://youtu.be/5uoCrtbMvqE)