## ECEN260 – Final Project

## Washing Machine Controller

Joshua Donaldson

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Program Specifications and Description**

This program is a washing machine controller. We will be able to look at the LCD display and cycle through options of what wash type we would like. The program will also posses a start and stop button. The starting and stopping of a cycle will be reflected on the other control board by LED 1. When the button is pressed, we will see the LED turn on and when it is pressed again we will see it turned off.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

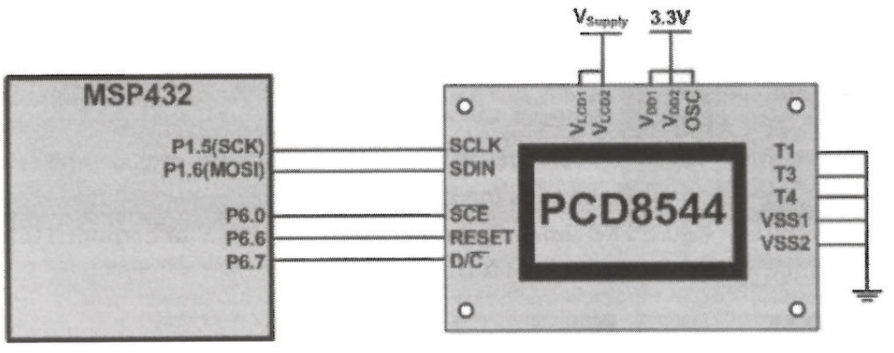
**Challenges**

The most time-consuming portion was the LCD display but the most difficult portion of this assignment for me was working with the pentameter to get the correct results displayed. I ran into a problem of the options not being displayed correctly when the pentameter was rotated. The other difficult portion was figuring out how to make it so that when I pressed the button the adjacent board would be able to turn off the light again or keep the light on in the case of another cycle being pressed.

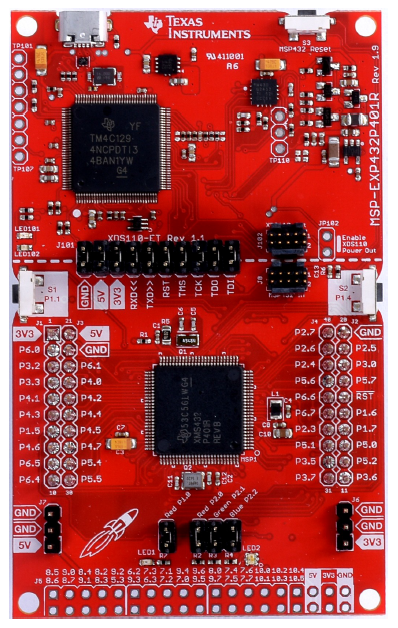
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Schematic or Wiring Diagram**

LCD Display wiring:



Uart hookup

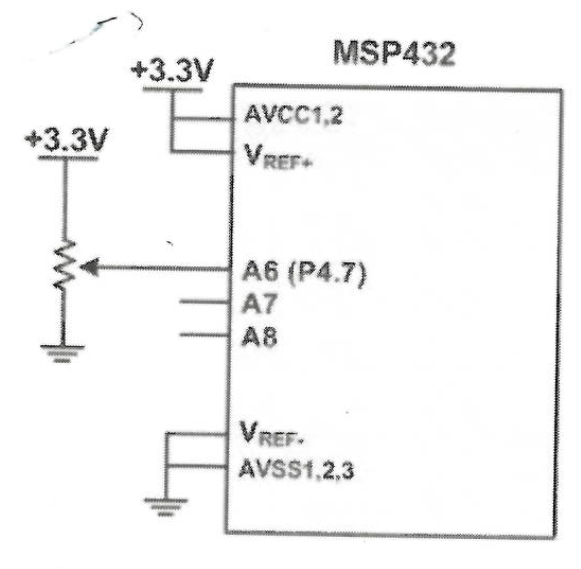


Connect a ground wire between both boards.

RXD (P3.2) of the first board goes to TXD (P3.3) of the second board.

TXD (P3.3) of the first board goes to RXD (P3.2) of the second board

Pentameter Wiring



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Test Plan**

In order to make sure that our program works first we will check the LCD display and turn the pentameter. When the Pentameter is rotated, we should see it cycle through 4 different options first should just say “welcome”, the next is “HEAVY DUTY 55 MIN”, “REGULAR 45 MIN”, and “DELICATES 30 MIN”. Next, we need to make sure that the UART is working correctly so we need to cycle through each option and press switch one. When switch one is pressed, we should see the RED LED1 light up on the other board. When pressed again on the same option we should see the LED turn off. We will do this for “HEAVY DUTY”, “REGULAR” and “DELICATES”. Then we need to push the button on the “WELCOME” when pushed we should see no change to the light.

If it passes those tests it’s working correctly.

Following the test a few functions that could work better is the pentameter is sensitive and can change the results making the button less responsive or unresponsive.

Another improvement is in the button when pushed it sometimes seems like it’s bouncing or can be somewhat unresponsive.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code**

Washing machine controller:

**#include** "msp.h"

/\* program function

\* Sets up ADC, UART, INPUT for switch1, and LCD display.

\* Runs conditional statements for ADC results to be sent to other microcontroller

\*/

//definitions for LCD

**#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

//definitions for ADC

**#define** G\_LED BIT0

**#define** R\_LED BIT1

**#define** B\_LED BIT2

//transmitter def

**#define** DELAY 300

**#define** S1 BIT1 //P1.1

//UART function

**void** **UART\_OutChar** (**char** data) {

**while** ( (EUSCI\_A2->IFG & BIT1) == 0); // Busy. Wait for previous output.

EUSCI\_A2->TXBUF = data; // Start transmission when IFG = 1.

}

//LCD 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);

/\* sample font table \*/

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

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

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

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

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

{0x7f, 0x41, 0x41, 0x41, 0x3e, 0x00}, /\* D fix this top of D up one bit \*/

{0x7f, 0x49, 0x49, 0x49, 0x49, 0x00}, /\* E \*/

{0x7f, 0x48, 0x48, 0x48, 0x48, 0x00}, /\* F \*/

{0x3e, 0x41, 0x51, 0x51, 0x72, 0x00}, /\* G \*/

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

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

{0x42, 0x41, 0x7E, 0x40, 0x40, 0x00}, /\* J \*/

{0x7f, 0x18, 0x24, 0x42, 0x01, 0x00}, /\* K \*/

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

{0x7e, 0x01, 0x1e, 0x01, 0x7e, 0x00}, /\* M \*/

{0x7f, 0x02, 0x0c, 0x10, 0x7f, 0x00}, /\* N \*/

{0x3e, 0x41, 0X41, 0X41, 0X3e, 0x00}, /\* O \*/

{0x7f, 0x48, 0x48, 0x48, 0x30, 0x00}, /\* P \*/

{0x3c, 0x42, 0X42, 0X43, 0X3d, 0x00}, /\* Q not finished \*/

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

{0x22, 0x51, 0x49, 0x45, 0x22, 0x00}, /\* S \*/

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

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

{0x03, 0x1c, 0x60, 0x1c, 0x03, 0x00}, /\* V \*/

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

{0x11, 0x0a, 0x04, 0x0a, 0x11, 0x00}, /\* x \*/

{0x01, 0x02, 0x7c, 0x02, 0x01, 0x00}, /\* Y \*/

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

{0x00, 0x6f, 0x6f, 0x6f, 0x00, 0x00}, /\* ! \*/

{0x00, 0x00, 0x7e, 0x81, 0xb5, 0xa1}, /\* left half face 28 \*/

{0xa1, 0xb5, 0x81, 0x7e, 0x00, 0x00}, /\* right half face 29\*/

{0x00, 0x60, 0x60, 0x60, 0x00, 0x00}, /\* . \*/

{0x22, 0x49, 0x49, 0x49, 0x36, 0x00}, /\* 3 31\*/

{0x3e, 0x43, 0X4d, 0X71, 0X3e, 0x00}, /\* 0 32\*/

{0x0f, 0x08, 0x08, 0x08, 0x7f, 0x00}, /\* 4 33\*/

{0x47, 0x45, 0x45, 0x45, 0x39, 0x00}, /\* 5 34\*/

}; /\*\*/

/\*\*

\* 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 \*/

**int** result; //create int for ADC result

//Configure ADC14

ADC14->CTL0 = 0x00000010; //power on ADC and disable during config

ADC14->CTL0 |= 0x04080310;// Condigure CTL0 according to this handout

ADC14->CTL1 = 0x00000020; // 12-bit resolution

ADC14->MCTL[5] = 0x06; // A6 input, single ended , Vref = AVCC

P4->SEL1 |= 0x80; //configure memory register 5

P4->SEL0 |= 0x80;

ADC14->CTL1 |= 0x00050000; //Configure for memory register 5

ADC14->CTL0 |= 0x02; //Enable ADC14 after configuration

//initialize UART

**int** i; //for loop index

**char** data = "0x00";

P1->DIR &= ~(S1); // set P1.1 as input

P1->REN |= S1; // turn on P1.1 pull resistor

P1->OUT |= S1; // configure P1.1 resistor as pull-up

// Configure port 3 as UART2.

P3->SEL0 |= BIT2 | BIT3; // Set bit 2 and bit 3 of P3SEL0 to 1

P3->SEL1 &= ~(BIT2 | BIT3); // Reset bit 2 and bit 3 of P3SEL1 to 0

// Set bit 0 in UCA2CTLW0 to 1. This resets the UART.

// Hold this bit at 1 during the rest of the configuration process.

EUSCI\_A2->CTLW0 |= BIT0; // Set WRST to put UART0 in reset

// Leave all other bits = 0

// While keeping bit 0 = 1, set the remaining control bits in UCA2CTLW0 to the

// values shown in section UART2 (UCA2) Registers.

EUSCI\_A2->CTLW0 |= BIT6 | BIT7; // Use SMCLK

// Calculate the hexadecimal number of the BWR for your desired baud rate

// using the method shown in section UART2 (UCA2) Registers of these

// laboratory instructions.

// Store the BRW in UCA2BRW to set the baud rate at 9600.

EUSCI\_A2->BRW= 0x139; // Baud rate = 9600

// Store 0x00 in UCA2MCTLW to disable oversampling.

EUSCI\_A2->MCTLW &= ~BIT0; // UCOS16 bit = 0

// Without altering the other UCA2CTLW0 bits, set bit 0 to 0. This makes the

// UART operational.

EUSCI\_A2->CTLW0 &= ~ BIT0; // Clear WRST to resume UART

// operation.

**while**(1)

{

ADC14->CTL0 |=1; //start a conversion

**while** (!ADC14->IFGR0); // wait until conversion is complete

result = ADC14->MEM[5]; //Read conversion result

**if**(result >> 8 == 4) //display option for heavy duty wash

{

GLCD\_clear();

GLCD\_putchar(8); //H

GLCD\_putchar(5); //E

GLCD\_putchar(1); //A

GLCD\_putchar(22); //V

GLCD\_putchar(25); //y

GLCD\_putchar(0); //

GLCD\_putchar(4); //D

GLCD\_putchar(21); //U

GLCD\_putchar(20); //T

GLCD\_putchar(25); //Y

GLCD\_putchar(0); //

GLCD\_putchar(34); //5

GLCD\_putchar(34); //5

GLCD\_putchar(0); //

GLCD\_putchar(13); // display letter M

GLCD\_putchar(9); // display letter I

GLCD\_putchar(14); // display letter N

}

**else** **if**(result >> 8 == 3) //set conditional for regular wash

{

GLCD\_clear();

GLCD\_putchar(18); //R

GLCD\_putchar(5); //E

GLCD\_putchar(7); //G

GLCD\_putchar(21); //U

GLCD\_putchar(12); //L

GLCD\_putchar(1); //A

GLCD\_putchar(18); //R

GLCD\_putchar(0); // display

GLCD\_putchar(33); //4

GLCD\_putchar(34); //5

GLCD\_putchar(0); //

GLCD\_putchar(13); // display letter M

GLCD\_putchar(9); // display letter I

GLCD\_putchar(14); // display letter N

}

**else** **if**(result >> 8 == 2) //set conditional for delicate wash

{

GLCD\_clear();

GLCD\_putchar(4); // display letter D

GLCD\_putchar(5); // display letter E

GLCD\_putchar(12); // display letter L

GLCD\_putchar(9); // display letter I

GLCD\_putchar(3); // display letter C

GLCD\_putchar(1); // display letter A

GLCD\_putchar(20); // display letter T

GLCD\_putchar(5); // display letter E

GLCD\_putchar(0); // display letter

GLCD\_putchar(31); // display number 3

GLCD\_putchar(32); // display number 0

GLCD\_putchar(0); //

GLCD\_putchar(0); //

GLCD\_putchar(0); // display

GLCD\_putchar(13); // display letter M

GLCD\_putchar(9); // display letter I

GLCD\_putchar(14); // display letter N

}

**else**

{

GLCD\_clear();

GLCD\_putchar(23); //W

GLCD\_putchar(5); //E

GLCD\_putchar(12); //L

GLCD\_putchar(3); //C

GLCD\_putchar(15); //O

GLCD\_putchar(13); //M

GLCD\_putchar(5); //E

}

// delay for switch debouncing

**for** (i = 0; i < DELAY; i++){}

**if**(result >> 8 == 4 && (P1->IN & S1) == 0x00){ //send heavy duty wash

data = 0x02;

UART\_OutChar(data);

}

**else** **if**(result >> 8 == 3 && (P1->IN & S1) == 0x00){ //send regular wash

data = 0x01;

UART\_OutChar(data);

}

**else** **if**(result >> 8 == 2 && (P1->IN & S1) == 0x00){ //send delicate wash

data = 0x02;

UART\_OutChar(data);

}

}

}

**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; /\* de-asssert 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 \*/

}

Cycle controller:

**#include** "msp.h"

/\*\*

\* this program will take input from another controller and based off the data it recieves will turn on LED1 or

\* Turn it off

\*/

**#define** S1 BIT1

**#define** LED1 BIT0

**char** **UART\_InChar** (**void**) {

**while** ( (EUSCI\_A2->IFG & BIT0) == 0); // Busy. Wait for received data.

**return** ( (**char**) (EUSCI\_A2->RXBUF) ); // Get new input when IFG = 1.

}

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

{

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

**char** data = "0x55";

**int** pause\_heavy = 0x01; //set flag for pause function

**int** pause\_regular = 0x01; //set flag for pause function

**int** pause\_delicate = 0x01; //set flag for pause function

P1->DIR |= LED1; //set P1 as output

// Configure port 3 as UART2.

P3->SEL0 |= BIT2 | BIT3; // Set bit 2 and bit 3 of P3SEL0 to 1

P3->SEL1 &= ~(BIT2 | BIT3); // Reset bit 2 and bit 3 of P3SEL1 to 0

// Set bit 0 in UCA2CTLW0 to 1. This resets the UART.

// Hold this bit at 1 during the rest of the configuration process.

EUSCI\_A2->CTLW0 |= BIT0; // Set WRST to put UART0 in reset

// Leave all other bits = 0

// While keeping bit 0 = 1, set the remaining control bits in UCA2CTLW0 to the

// values shown in section UART2 (UCA2) Registers.

EUSCI\_A2->CTLW0 |= BIT6 | BIT7; // Use SMCLK

// Calculate the hexadecimal number of the BWR for your desired baud rate

// using the method shown in section UART2 (UCA2) Registers of these

// laboratory instructions.

// Store the BRW in UCA2BRW to set the baud rate at 9600.

EUSCI\_A2->BRW= 0x139; // Baud rate = 9600

// Store 0x00 in UCA2MCTLW to disable oversampling.

EUSCI\_A2->MCTLW &= ~BIT0; // UCOS16 bit = 0

// Without altering the other UCA2CTLW0 bits, set bit 0 to 0. This makes the

// UART operational.

EUSCI\_A2->CTLW0 &= ~ BIT0; // Clear WRST to resume UART

// operation.

**while**(1){

data = UART\_InChar(); //call reciever function to read which data is being transfered

**if**(data == 0x02){ // run heavy wash condition

**if**(pause\_heavy == 0x01){ // check to see if the wash is paused to continue cycle

pause\_heavy = 0x00; //change pause to false

pause\_regular = 0x01; // change all other flags to true

pause\_delicate = 0x01;

P1->OUT |= LED1; //turn on light to indicate cycle running

}

**else** **if** (pause\_heavy == 0x00){ //check to see if cycle is in process

pause\_heavy = 0x01; //if cycle was running change pause to true and turn on LED

P1->OUT &= ~ LED1;

}

}

**else** **if**(data == 0x01){ //run regular wash function

**if**(pause\_regular == 0x01){ // check to see if the wash is paused to continue cycle

pause\_heavy = 0x01;

pause\_regular = 0x00; // if regualr was paused start regular and turn off other flags

pause\_delicate = 0x01;

P1->OUT |= LED1;

}

**else** **if** (pause\_regular == 0x00){ //check to see if cycle is in process

pause\_heavy = 0x01;

P1->OUT &= ~ LED1;

}

}

**else** **if**(data == 0x00){ //run delicate wash condition

**if**(pause\_delicate == 0x01){ // check to see if the wash is paused to continue cycle

pause\_heavy = 0x01;

pause\_regular = 0x01;

pause\_delicate = 0x00;

P1->OUT |= LED1;

}

**else** **if** (pause\_delicate == 0x00){ //check to see if cycle is in process

pause\_heavy = 0x01;

P1->OUT &= ~ LED1;

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_