// Lab 5: LCD Display

// 5.1 Printing numbers on the LCD display

// Interfacing the LCD screen to the microcontroller via

// an LCD controller, located inside the MCU.

// The LCD display requires more than just low/high signals

// to turn the segments on/off.

// An LCD segment --> Having two terminals that should always oscillate.

// The segment is provided with a continuous voltage. --> Causing burn out!

// The LCD controller is responsible for providing the alternating

// voltages on the segments' terminals and uses a clock signal for

// this purpose.

// The LCD controller on our microcontroller is the LCD\_C module.

// Interfacing LCD displays --> The number of segments that can

// easily exceed the number of available pins on the microcontroller.

// The "static drive" configuration --> When a pin on the microcontroller

// controls one LCD segment (no multiplexing).

// Multiplexing usually ranges from 2-way to 8-way according to

// which the latter uses the least number of pins on the MCU.

// The down-side of multiplexing is that a pin controls multiple

// segments --> One at a time or not simultaneously!

// Doing it fast enough that the user's eyes don't see the flickering.

// Multiplexing scale is larger --> 8-way multiplexing

// The contrast control could become poorer.

// Segment memory mapping and shapes of digits

// LCD Display on LaunchPad --> Having six alphanumeric characters.

// They are 14-segment displays.

// They can display letters and numbers.

// If only numbers are displayed, the digits can be slightly

// different.

// The segments of the LCD display are mapped to the memory variables of

// the LCD controller.

// The Memory Variables: LCDM1 and LCDM2

// The full layout of the LCD display and the names of the segments.

// Zooming in on the leftmost character to see the segment names

// of the alphanumeric character.

// Seeing that the segments A, B, C, D, E, F, G, M --> They make

// the outer ring and the middle horizontal bars.

// Eight Segments --> Used to display the digits 0 to 9.

// Finding the mapping of these segments to the memory variables LCDMx.

// The six alphanumeric characters on the display --> 1 to 6.

// The variable LCDM8 --> Corresponding to the segments A6A, A6B,

// ..., A6M

// LCDM8 --> Used to display a digit on the rightmost character.

// Displaying the digits on the second and the third characters

// from the right (characters 5 and 4) using the variables

// LCDM15 and LCDM19 respectively.

// Facilitating display of the digits 0 to 9 on the alphanumeric characters.

// A good idea --> For declaration of an array that stores the shapes of the digits.

// Display a digit that has been looked up its shape from the array.

// Writing it to the LCDMx variable.

// Array declaration --> Calling LCD\_Num

// Shape of zero corresponds to the segments A, B, C, D, E, F!

// Format of LCDM8 --> All the segments should be turned on except

// the rightmost two.

// Binary Value: 1111 1100 (=0xFC)

// LCD\_Num --> Storing the shape of zero (0xFC) at index zero.

// Displaying 1 on the character

// Segments B and C should be ON.

// A binary value of 0110 0000 that is 0x60!

// Storing the shape of 1 (0x60) at index 1.

// Complete the array by storing the shape 2 at index 2

// All the memory variables (LCDM8, LCDM15, LCDM19).

// The array LCD\_Num is initialized.

// unsigned char LCD\_Num[10] = {0xFC, 0x60, 0xDB, ...}

// Writing the code that prints the number 430 on

// the rightmost three digits of the display.

// Sample code that prints 430 on the LCD monitor

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LEDs pin location is at P1.0

**#define** greenLED BIT7 // Green LEDs pin location is at P9.7

// prototype functions used in the main code.

**void** **Initialize\_LCD**();

**void** **display\_num\_lcd**(**unsigned** **int** n);

// Array that enables the ease of displaying alphanumeric characters in the LCD.

**const** **unsigned** **char** LCD\_Num[10] = {0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xF7};

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

**volatile** **unsigned** **int** n; // Delay counter

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Configuration and initialization of LEDs

P1DIR |= redLED; // Red LED pin is set as output

P9DIR |= greenLED; // Green LED is set as output

P1OUT &= ~redLED; // Red LED starts off

P9OUT &= ~greenLED; // Green LED starts off

// LCD module initialization

Initialize\_LCD();

// Clears all the segments

LCDCMEMCTL = LCDCLRM;

// Function that displays an alphanumeric character in the LCD.

display\_num\_lcd(12);

**for**(;;){

**for**(n=0; n<=50000; n++){} // Delay Loop

P1OUT ^= redLED; // Toggles on/off the red LED.

P9OUT ^= greenLED; // Toggles on/off the green LED.

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Initializes the LCD\_C module

// \*\*\* Source: Function obtained from MSP430FR6989’s Sample Code \*\*\*

**void** **Initialize\_LCD**() {

PJSEL0 = BIT4 | BIT5; // For LFXT

// Initialize LCD segments 0 - 21; 26 - 43

LCDCPCTL0 = 0xFFFF;

LCDCPCTL1 = 0xFC3F;

LCDCPCTL2 = 0x0FFF;

// Configure LFXT 32kHz crystal

CSCTL0\_H = CSKEY >> 8; // Unlock CS registers

CSCTL4 &= ~LFXTOFF; // Enable LFXT

**do** {

CSCTL5 &= ~LFXTOFFG; // Clear LFXT fault flag

SFRIFG1 &= ~OFIFG;

}**while** (SFRIFG1 & OFIFG); // Test oscillator fault flag CSCTL0\_H = 0; // Lock CS registers

// Initialize LCD\_C // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX

LCDCCTL0 = LCDDIV\_\_1 | LCDPRE\_\_16 | LCD4MUX | LCDLP;

// VLCD generated internally, // V2-V4 generated internally, v5 to ground // Set VLCD voltage to 2.60v // Enable charge pump and select internal reference for it

LCDCVCTL = VLCD\_1 | VLCDREF\_0 | LCDCPEN;

LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled

LCDCMEMCTL = LCDCLRM; // Clear LCD memory

//Turn LCD on

LCDCCTL0 |= LCDON;

**return**;

}

**void** **display\_num\_lcd**(**unsigned** **int** n){

**volatile** **unsigned** **int** digit = 0, i = 0;

// Ensures the display of characters in the LCD from right to left.

**volatile** **unsigned** **char** \*ptr[6] = {&LCDM8, &LCDM15, &LCDM19, &LCDM4, &LCDM6, &LCDM10};

// Extracts the digits from the input variable and displays it in the LCD.

**do**{

digit = n % 10;

\*ptr[i] = LCD\_Num[digit];

n = n/10;

i++;

}**while**(n != 0);

// Clears the unused LCDMX variables

**while**(i <= 7 ){

\*ptr[i] = 0;

i++;

}

}

// 5.2 Implementing a Stopwatch

// Writing a code to implement a stopwatch

// Timer\_A Module with the 32 KHz crystal since it is a precise

// clock signal

// Using this function from earlier labs that configuring ACLK to

// the 32 KHz crystal.

// Using the timer in the up mode to generate a delay of one second

// Start printing 0 on the display and when a second elapses!

// The number counts up to 1 and 2.

// No need to implement hours/minutes/second.

// Continuous counting up: 59, 60, 61, ..., 65535, 0 , 1.

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LEDs pin location is at P1.0

**#define** greenLED BIT7 // Green LEDs pin location is at P9.7

**#define** BUT1 BIT1 // Push Button S1's location is at P1.1

**#define** BUT2 BIT2 // Push Button S2's location is at P1.2

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

**void** **config\_ACLK\_to\_32KHz\_crystal**() {

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

// prototype functions used in the main code.

**void** **Initialize\_LCD**();

**void** **display\_num\_lcd**(**unsigned** **int** n);

// Array that enables the ease of displaying alpha numerical characters in the LCD.

**const** **unsigned** **char** LCD\_Num[10] = {0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xF7};

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

**volatile** **unsigned** **int** n = 0; // Display counter

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Configuration and initialization of LEDs

P1DIR |= redLED; // Red LED pin is set as output

P9DIR |= greenLED; // Green LED is set as output

P1OUT &= ~redLED; // Red LED starts off

P9OUT &= ~greenLED; // Green LED starts off

// We configure the Auxiliary Clock to reroute to the 32KHz crystal.

config\_ACLK\_to\_32KHz\_crystal();

// LCD module initialization

Initialize\_LCD();

// Clears all the segments

LCDCMEMCTL = LCDCLRM;

// Configure Timer\_A in Up mode

// We set the value to 32768 which is a 1 second delay since 32768/32768 = 1.

TA0CCR0 = 32768;

// Configuring Timer\_A0 by setting ACLK, frequency division by 1, up mode, and clear TAR.

TA0CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

// Ensure the flag is cleared at the start

TA0CTL &= ~TAIFG;

**for**(;;)

{

**while**((TA0CTL & TAIFG)==0){} // Generate Delay of 1 second

display\_num\_lcd(n); // Displays the input number n.

TA0CTL &= ~TAIFG; // Clears the flag

n++; // Increments the display counter

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Initializes the LCD\_C module

// \*\*\* Source: Function obtained from MSP430FR6989’s Sample Code \*\*\*

**void** **Initialize\_LCD**() {

PJSEL0 = BIT4 | BIT5; // For LFXT

// Initialize LCD segments 0 - 21; 26 - 43

LCDCPCTL0 = 0xFFFF;

LCDCPCTL1 = 0xFC3F;

LCDCPCTL2 = 0x0FFF;

// Configure LFXT 32kHz crystal

CSCTL0\_H = CSKEY >> 8; // Unlock CS registers

CSCTL4 &= ~LFXTOFF; // Enable LFXT

**do** {

CSCTL5 &= ~LFXTOFFG; // Clear LFXT fault flag

SFRIFG1 &= ~OFIFG;

}**while** (SFRIFG1 & OFIFG); // Test oscillator fault flag CSCTL0\_H = 0; // Lock CS registers

// Initialize LCD\_C // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX

LCDCCTL0 = LCDDIV\_\_1 | LCDPRE\_\_16 | LCD4MUX | LCDLP;

// VLCD generated internally, // V2-V4 generated internally, v5 to ground // Set VLCD voltage to 2.60v // Enable charge pump and select internal reference for it

LCDCVCTL = VLCD\_1 | VLCDREF\_0 | LCDCPEN;

LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled

LCDCMEMCTL = LCDCLRM; // Clear LCD memory

//Turn LCD on

LCDCCTL0 |= LCDON;

**return**;

}

**void** **display\_num\_lcd**(**unsigned** **int** n){

**volatile** **unsigned** **int** digit = 0, i = 0;

// Ensures the display of characters in the LCD from right to left.

**volatile** **unsigned** **char** \*ptr[6] = {&LCDM8, &LCDM15, &LCDM19, &LCDM4, &LCDM6, &LCDM10};

// Extracts the digits from the input variable and displays it in the LCD.

**do**{

digit = n % 10;

\*ptr[i] = LCD\_Num[digit];

n = n/10;

i++;

}**while**(n != 0);

// Clears the unused LCDMX variables

**while**(i <= 7 ){

\*ptr[i] = 0;

i++;

}

}

// 5.3 Stopwatch with Halt/Resume and Reset Functions

// Pushing the button S1.

// Stopping the counting and remaining the current number on the display

// Pushing S1 again should resume the counting from the currently displayed value.

// Pushing S2 should reset the count to zero.

// Performing the reset during counting.

// The value should go to zero and the counting go to zero.

// The counting should continue.

// Performing the reset when the counting is halted.

// Result: The value goes to zero and the stopwatch remains halted.

// Also, it is indicated that the counting/halted stats using the LED lights.

// Counting the stop-watch and the green LED turns ON.

// Halting the stop-watch and the red LED should be ON.

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LEDs pin location is at P1.0

**#define** greenLED BIT7 // Green LEDs pin location is at P9.7

**#define** BUT1 BIT1 // Push Button S1's location is at P1.1

**#define** BUT2 BIT2 // Push Button S2's location is at P1.2

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

**void** **config\_ACLK\_to\_32KHz\_crystal**() {

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

// prototype functions used in the main code.

**void** **Initialize\_LCD**();

**void** **display\_num\_lcd**(**unsigned** **int** n);

// Array that enables the ease of displaying alpha numerical characters in the LCD.

**const** **unsigned** **char** LCD\_Num[10] = {0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xF7};

// Flag for the reset function

**volatile** **unsigned** **int** flag = 0;

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

**volatile** **unsigned** **int** n = 0;

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Configuration and initialization of LEDs

P1DIR |= redLED; // Red LED pin is set as output

P9DIR |= greenLED; // Green LED is set as output

P1OUT &= ~redLED; // Red LED starts off

P9OUT &= ~greenLED; // Green LED starts off

// Configuring buttons with interrupt

P1DIR &= ~(BUT1|BUT2); // Sets the push buttons S1 and S2 as input

P1REN |= (BUT1|BUT2); // Enables the internal resistors of both buttons

P1OUT |= (BUT1|BUT2); // Sets the resistors as pull up to VCC

P1IE |= (BUT1|BUT2); //Enables the interrupt events of both buttons

P1IES |= (BUT1|BUT2); // Configures the interrupt as falling edge

P1IFG &= ~(BUT1|BUT2); // Clears both interrupt flags.

// We configure the Auxiliary Clock to reroute to the 32KHz crystal.

config\_ACLK\_to\_32KHz\_crystal();

// LCD module initialization

Initialize\_LCD();

// Clears all the segments

LCDCMEMCTL = LCDCLRM;

// Configure Timer\_A in Up mode

// We set the value to 32768 which is a 1 second delay since 32768/32768 = 1.

TA0CCR0 = 32768;

// Configuring Timer\_A0 by setting ACLK, frequency division by 1, up mode, and clear TAR.

TA0CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

// Ensure the flag is cleared at the start

TA0CTL &= ~TAIFG;

// Enables the global interrupt enable.

\_enable\_interrupts();

**for**(;;)

{

// Timer delay of 1 second

**while**((TA0CTL & TAIFG)==0)

{

// Reset function that sets the counter to 0 regardless of the status of the stopwatch.

**if**(flag == 1)

{

n= 0;

display\_num\_lcd(n);

flag=0;

}

}

// Displays the display counter

display\_num\_lcd(n);

TA0CTL &= ~TAIFG; // Clears the timer interrupt flag

n++; // Increment the display counter

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Initializes the LCD\_C module

// \*\*\* Source: Function obtained from MSP430FR6989’s Sample Code \*\*\*

**void** **Initialize\_LCD**() {

PJSEL0 = BIT4 | BIT5; // For LFXT

// Initialize LCD segments 0 - 21; 26 - 43

LCDCPCTL0 = 0xFFFF;

LCDCPCTL1 = 0xFC3F;

LCDCPCTL2 = 0x0FFF;

// Configure LFXT 32kHz crystal

CSCTL0\_H = CSKEY >> 8; // Unlock CS registers

CSCTL4 &= ~LFXTOFF; // Enable LFXT

**do** {

CSCTL5 &= ~LFXTOFFG; // Clear LFXT fault flag

SFRIFG1 &= ~OFIFG;

}**while** (SFRIFG1 & OFIFG); // Test oscillator fault flag CSCTL0\_H = 0; // Lock CS registers

// Initialize LCD\_C // ACLK, Divider = 1, Pre-divider = 16; 4-pin MUX

LCDCCTL0 = LCDDIV\_\_1 | LCDPRE\_\_16 | LCD4MUX | LCDLP;

// VLCD generated internally, // V2-V4 generated internally, v5 to ground // Set VLCD voltage to 2.60v // Enable charge pump and select internal reference for it

LCDCVCTL = VLCD\_1 | VLCDREF\_0 | LCDCPEN;

LCDCCPCTL = LCDCPCLKSYNC; // Clock synchronization enabled

LCDCMEMCTL = LCDCLRM; // Clear LCD memory

//Turn LCD on

LCDCCTL0 |= LCDON;

**return**;

}

**void** **display\_num\_lcd**(**unsigned** **int** n){

**volatile** **unsigned** **int** digit = 0, i = 0;

// Ensures the display of characters in the LCD from right to left.

**volatile** **unsigned** **char** \*ptr[6] = {&LCDM8, &LCDM15, &LCDM19, &LCDM4, &LCDM6, &LCDM10};

// Extracts the digits from the input variable and displays it in the LCD.

**do**{

digit = n % 10;

\*ptr[i] = LCD\_Num[digit];

n = n/10;

i++;

}**while**(n != 0);

// Clears the unused LCDMX variables

**while**(i <= 7 ){

\*ptr[i] = 0;

i++;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#pragma** vector = PORT1\_VECTOR

**\_\_interrupt** **void** **Port1\_ISR**() {

// If S1 push button was pushed

**if**((P1IFG & BUT1) == BUT1){

P1OUT ^= redLED; //Toggles on/off the red LED

P9OUT ^= greenLED; //Toggles on/off the green LED

TA0CTL ^= MC\_1; // Toggles between up mode and stop mode

flag = 0; // Set flag to 0

P1IFG &= ~BUT1; // Clear the flag raised by S1 push button

}

// If S2 push button was pushed

**if**((P1IFG & BUT2) == BUT2){

flag = 1; // Sets the flag to 1

P1IFG &= ~BUT2; // Clear the flag raised by S2 push button

}

}