# Lab 5 Report

EEL4742C - 00446

Yousef Awad

September 2025

## Contents

# Introduction

## 5.1 Printing on the LCD Display

```c
#include <msp430fr6989.h>
#include <stdint.h>

#define redLED BIT0 // Red at P1.0
#define greenLED BIT7 // Green at P9.7

// The array has the shapes of the digits (0 to 9)
const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };

//**********************************************************
// Initializes the LCD_C module
void Initialize_LCD()
{
  PJSEL0 = BIT4 | BIT5; // For LFXT
  LCDCPCTL0 = 0xFFD0;
  LCDCPCTL1 = 0xF83F;
  LCDCPCTL2 = 0x00F8;
  // 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 (do this at the end!)
  LCDCCTL0 |= LCDON;
  return;
}

void lcd_write_uint16 (uint16_t number)
{
  uint8_t digits[5]; // we have a max of 5 possible digits
  // Addresses for each possible digit.
  volatile unsigned char *display[5] = { &LCDM6, &LCDM4, &LCDM19, &LCDM15, &LCDM8 };

  uint8_t i = 0;
  for (i = 0; i < 5; i++)
  {
    // Finding all digits in the number inputted
    digits[4 - i] = number % 10;
    number /= 10;
  }

  int leading = 0;
  for (i = 0; i < 5; i++)
  {
    // Incrementing through all 5 digits
    if (digits[i])
    {
      // A digit exists at this display section
      leading = 1;
    }
    if ((i == 4) && (leading == 0))
    {
      // We don’t have any value at this display section
      *display[i] = LCD_Shapes[0];
    }
    else
    {
      // The digit we are on was not the 5th one OR it was the 5th
      // one and was not a leading digit
      if (leading)
      {
        // We have a leading digit
        *display[i] = LCD_Shapes[digits[i]];
      }
      else
      {
        // We do not have a leading digit
        *display[i] = 0;
      }
    }
  }

  // Exiting function
  return;
}

int main(void)
{
  volatile unsigned int n;
  WDTCTL = WDTPW | WDTHOLD; // Stop WDT
  PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
  P1DIR |= redLED; // Pins as output
  P9DIR |= greenLED;
  P1OUT |= redLED; // Red on
  P9OUT &= ~greenLED; // Green off
  // Initializes the LCD_C module
  Initialize_LCD();

  uint16_t number = 51487;
  lcd_write_uint16(number);

  // Flash the red LED
  for(;;)
  {
    for(n=0; n<=60000; n++)
    {
      // Delay loop
    }
    P1OUT ^= redLED;
  }
}
```

## 5.2 Implementing a Counter

When comparing the time on the counter for 5.2 that I've made it almost perfectly matches it, with the error most likely being due to human error and my own response time.

```c
#include <msp430fr6989.h>
#include <stdint.h>

#define redLED BIT0 // Red at P1.0
#define greenLED BIT7 // Green at P9.7

#define but1 BIT1 // Port 1.1
#define but2 BIT2 // Port 1.2

// The array has the shapes of the digits (0 to 9)
const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67, 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
uint16_t counter = 0;

//*************************************************************
// Initializes the LCD_C module
void Initialize_LCD()
{
  PJSEL0 = BIT4 | BIT5; // For LFXT
  LCDCPCTL0 = 0xFFD0;
  LCDCPCTL1 = 0xF83F;
  LCDCPCTL2 = 0x00F8;
  // 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 (do this at the end!)
  LCDCCTL0 |= LCDON;
  return;
}

void lcd_write_uint16 (uint16_t number)
{
  uint8_t digits[5]; // we have a max of 5 possible digits
  // Addresses for each possible digit.
  volatile unsigned char *display[5] = { &LCDM6, &LCDM4, &LCDM19, &LCDM15, &LCDM8 };

  uint8_t i = 0;
  for (i = 0; i < 5; i++)
  {
    // Finding all digits in the number inputted
    digits[4 - i] = number % 10;
    number /= 10;
  }

  int leading = 0;
  for (i = 0; i < 5; i++)
  {
    // Incrementing through all 5 digits
    if (digits[i])
    {
      // A digit exists at this display section
      leading = 1;
    }
    if ((i == 4) && (leading == 0))
    {
      // We don't have any value at this display section
      *display[i] = LCD_Shapes[0];
    }
    else
    {
      // The digit we are on was not the 5th one OR it was the 5th
      // one and was not a leading digit
      if (leading)
      {
        // We have a leading digit
        *display[i] = LCD_Shapes[digits[i]];
      }
      else
      {
        // We do not have a leading digit
        *display[i] = 0;
      }
    }
  }

  // Exiting function
  return;
}

// Configures ACLK to 32 KHz crystal
```

```c
void config_ACLK_to_32KHz_crystal()
{
  // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz

  // Reroute pins to LFXIN/LFXOUT functionality
  PJSEL1 &= ~BIT4;
  PJSEL0 |= BIT4;

  // Wait until the oscillator fault flags remain cleared
  CSCTL0 = CSKEY; // Unlock CS registers
  do
  {
    CSCTL5 &= ~LFXTOFFG; // Local fault flag
    SFRIFG1 &= ~OFIFG; // Global fault flag
  }
  while((CSCTL5 & LFXTOFFG) != 0);

  CSCTL0_H = 0; // Lock CS registers
  return;
}

int main(void)
{
  volatile unsigned int n;
  WDTCTL = WDTPW | WDTHOLD; // Stop WDT
  PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
  P1DIR |= redLED; // Pins as output
  P9DIR |= greenLED;
  P1OUT |= redLED; // Red on
  P9OUT &= ~greenLED; // Green off

  P1DIR &= ~(but1 | but2);
  P1REN |= (but1 | but2);
  P1OUT |= (but1 | but2);
  P1IES |= (but1 | but2);
  P1IFG &= (but1 | but2);
  P1IE  |= (but1 | but2);
  P1IFG = 0; // resetting flag for interrupt

  // Initializes the LCD_C module
  Initialize_LCD();

  config_ACLK_to_32KHz_crystal();

  // Starting timer
  TA0CCR0 = 32767;
  TA0CCTL0 = CCIE;
  TA0CCTL0 &= ~CCIFG;
  TA0CTL = TASSEL_1 | ID_0 | MC_1 | TAIE;
  TA0CTL &= ~TAIFG;

  _low_power_mode_3();

  return 0;
}

#pragma vector = TIMER0_A0_VECTOR
```

```c
__interrupt void TA0_ISR()
{
  lcd_write_uint16(counter);
  counter += 1;
  TA0CTL &= ~TAIFG;
}

#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR()
{
  if ((P1IFG & but1))
  {
    counter = 0;
    P1IFG &= ~but1; // clear flag
  }

  if ((P1IFG & but2))
  {
    counter += 1000;
    P1IFG &= ~but2; // clear flag
  }
}
```

## 5.3 Utility Chronometer

Just to get this out of the way, the maximum design of my chronometer is that of 12 hours like an AM/PM clock. This is due to the fact that I put the max value that the counter, in seconds, will go to, to be 43199. Now, onto the design. I basically redid the entirety of the 5.2 and 5.1, mainly due to the fact that I started with 5.3 and went backward in writing the code. I specifically chose to make the buttons work via polling once they are already activated, mainly due to the fact that it's just a lot easier for me, and energy consumption is not of a concern. Alongside this, I used a couple of while loops so that the speed-up function of the buttons can be easily looped indefinetely until they are released. That's really it, that's different about the versions from 5.2 and 5.1, when compared to 5.3.

```c
#include <msp430fr6989.h>
#include <stdbool.h>
#include <stdint.h>

#define red BIT0   // Port 1.0
#define green BIT7 // Port 9.7

#define but1 BIT1 // Port 1.1
#define but2 BIT2 // Port 1.2

bool state = true; // true is unpaused
volatile uint16_t totalSec = 0;
volatile uint16_t seconds = 0;
volatile uint16_t minutes = 0;
volatile uint16_t hours = 0;
volatile uint16_t counter = 0;
```

```c
uint16_t holding = 1;


const unsigned char LCD_Shapes[10] = { 0xFC, 0x60, 0xDB, 0xF3, 0x67
    , 0xB7, 0xBF, 0xE0, 0xFF, 0xE7 };
uint16_t digits[6] = {0};

int checkingBut2And1 = 0;

// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal()
{
  // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz

  // Reroute pins to LFXIN/LFXOUT functionality
  PJSEL1 &= ~BIT4;
  PJSEL0 |= BIT4;

  // Wait until the oscillator fault flags remain cleared
  CSCTL0 = CSKEY; // Unlock CS registers
  do
  {
    CSCTL5 &= ~LFXTOFFG; // Local fault flag
    SFRIFG1 &= ~OFIFG; // Global fault flag
  }
  while((CSCTL5 & LFXTOFFG) != 0);

  CSCTL0_H = 0; // Lock CS registers
  return;
}

void toggle_colon()
{
  // All this function does is toggle the colon
  LCDM7 ^= BIT2;
}

void toggle_exclamation()
{
  LCDM3 |= BIT0;
  LCDM3 &= ~BIT3;
}

void toggle_stopwatch()
{
  LCDM3 |= BIT3;
  LCDM3 &= ~BIT0;
}

// Initializes the LCD_C module
void Initialize_LCD()
{
  PJSEL0 = BIT4 | BIT5; // For LFXT
  LCDCPCTL0 = 0xFFD0;
  LCDCPCTL1 = 0xF83F;
  LCDCPCTL2 = 0x00F8;
  // Configure LFXT 32kHz crystal
  CSCTL0_H = CSKEY >> 8;
  CSCTL4 &= ~LFXTOFF;
  do
  {
    // Unlock CS registers
    // Enable LFXT
    CSCTL5 &= ~LFXTOFFG;
    SFRIFG1 &= ~OFIFG;
  }
  while (SFRIFG1 & OFIFG);
  CSCTL0_H = 0; // Clear LFXT fault flag
  // Test oscillator fault flag
  // 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; // Turn LCD on (do this at the end!)
  LCDCCTL0 |= LCDON;
  // Clear LCD memory
  return;
}

// This function does not initialize the timer, but instead is used for starting and stopping it.
// This function returns the time that was inputted if succesfull, and returns -1 if unsuccessful
int startTime(uint16_t time_given)
{
  if (time_given < 0)
  {
    // Returning false.
    return -1;
  }

  // time_given is a valid value
  TA0CCR0 = time_given;
  TA0CCTL0 = CCIE;
  TA0CCTL0 &= ~CCIFG;
  TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
  TA0CTL &= ~TAIFG; // Clearing interrupt flag (if any)

  // Returning success
  return time_given;
}

// This function initializes the timer to work.
void initializeTime()
{
  TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR | TAIE;
  TA0CTL &= ~TAIFG; // Clearing any previous flags (if any)
}
```

```c

void stopTime()
{
  // Clearing the timer.
  TA0CTL = MC_0 | TACLR;
}

void LCDframe()
{
  // Enable the decimal point
  LCDM20 = BIT0;

  if (state)
  {
    // we are playing time
    toggle_stopwatch();
    TA0CTL |= MC_1;
    if (totalSec > 43199)
    {
      totalSec = 0;
    }
  }
  else
  {
    toggle_exclamation();
    TA0CTL &= ~MC_3;
  }
  toggle_colon();

  seconds = totalSec % 60;
  minutes = (totalSec % 3600) / 60;
  hours = totalSec / 3600;

  uint8_t i = 0; // temporary counter

  uint8_t *display[6] = {&LCDM10, &LCDM6, &LCDM4, &LCDM19, &LCDM15, &LCDM8};

  uint16_t temp;

  temp = seconds;
  // Writing seconds
  for (i = 0; i < 2; i++)
  {
    digits[i] = temp % 10;
    temp /= 10; // go down
    *display[5 - i] = LCD_Shapes[digits[i]];
  }

  // Writing minutes
  temp = minutes;
  for (i = 0; i < 2; i++)
  {
    digits[i] = temp % 10;
    temp /= 10;
    *display[3 - i] = LCD_Shapes[digits[i]];
  }

  // Writing hours
  temp = hours;
  for (i = 0; i < 2; i++)
  {
    digits[i] = temp % 10;
    temp /= 10;
    *display[1 - i] = LCD_Shapes[digits[i]];
  }
}

int main(void)
{
  WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer
  PM5CTL0 &= ~LOCKLPM5;     // Enable the GPIO pins

  P1DIR |= red;  // Set output for red led
  P1OUT &= ~red; // Turn off red LED

  P9DIR |= green;  // Set output for green led
  P9OUT &= ~green; // Turn off green LED

  // Using default initialize function
  Initialize_LCD();
  LCDCMEMCTL = LCDCLRM; // Clear LCD

  // Configure buttons
  P1DIR &= ~(but1 | but2); // input
  P1REN |= (but1 | but2);  // enable resistors
  P1OUT |= (but1 | but2);  // pull-up

  P1IES |= (but1 | but2);   // interrupt on falling edge
  P1IE  |= (but1 | but2);   // enable interrupts
  P1IFG  = 0;               // set interrupt flag off

  config_ACLK_to_32KHz_crystal();

  // initializeTime(); // To set the interrupt flag forever.
  startTime(32767); // Clock max is 1 second per.

  _low_power_mode_0();

  return 0;
}

#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR()
{
  if ((P1IFG & but1))
  {
    holding = 1;
    // S1 was pressed

    while (!(P1IN & but1))
    {
      // S1 was held.
      holding += 1;
      _delay_cycles(10000);
      if (holding > 110)
      {
        break;
      }
    }

    if (holding > 100)
    {
      // We held it long enough.
      state = false;
      totalSec = 0;
      // Stop the state.
    }
    else
    {
      state = !state;
    }

    P1IFG &= ~but1; // clear flag
    LCDframe();
      __delay_cycles(50000); // 50k cycles for debouncing
  }

  if ((P1IFG & but2))
  {
    checkingBut2And1 = 1;
    while (!(P1IN & but2))
    {
      LCDframe();
      // You are pushing down only button2
      if (!(P1IN & but1))
      {
        totalSec--;
      }
      else
      {
        totalSec++;
      }
      __delay_cycles(100); // debounce
    }
    P1IFG &= ~but2;
  }
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR()
{
  // generate a new frame every time the timer overflows.
  totalSec += 1;
  LCDframe();
}

#pragma vector = TIMER0_A1_VECTOR
__interrupt void T0A1_ISR()
{
  // Do nothing
```

```
}
```

## Student Q&A

### 1

**Given:** Explain whether this statement is true or false. If false, explain the correct operation. “An LCD segment works just like a colored LED. It’s turned on/off by writing either digital high/low to it, respectively”.

The statement is true. It works the exact same way as an LED, its just that you have more than one LED to turn on and off at a given frequency (as it’s not always on, and instead flickers extremely fast so as to conserve energy).

### 2

**Given:** What is the name of the LCD controller that interfaces the LCD display of our board? Is the LCD controller located on the display module or in the microcontroller?

The LCD controller is specifically located inside the MCU of the MSP430, specifically it is called ’LCD_C’.

### 3

**Given:** In what multiplexing configuration is the LCD module wired (2-way, 4-way, etc)? What does this mean regarding the number of pins used at the microcontroller?

The LCD is wired in 4-way multiplexing so as to conserve the amount of pins that the microcontroller needs by a factor of 4. Since we have a 108 segment display, this therefore means we only need 27 pins to write the high/low values to the entirety of said display.