EGCP 450-01 Lab 6: Serial Communication and the Seven-Segment Display

James Samawi – 12/18/2020

Introduction

In this lab, I will program a 4 digit 7-segment display (SSEG) (3461BS) driver to brighten internal LEDs using the common cathode method. For part one, we must allow the user to increment the number displayed on the SSEG or decrement using external buttons, and the single digit output must loop once reaching either zero or nine case. For part two, the user is able to enter a number between 0 and 9999 and the SSEG will output this number along with allowing the number to be incremented and decremented on the go. To achieve this, we will use a physical 8-bit shift register (SN74HC595N) prior to the SSEG and take advantage of time-multiplexing to send serial data with the optical allusion of parallelism. Working with 7-segment displays is a useful skill in embedded systems because this is a simpler method than interfacing with an LCD display or most displays.

Procedure/Discussion

The 3461BS 4 digit SSEG was used throughout part one and part two (Fig. 1). For part one, D1 was set to 3.3V and A through G segments were directly wired to port 4 pins 0 through 6. I also did not have 470 Ohm resistors, so I used a combination of 220 Ohm resistors in series as well as 500 Ohm resistors in parallel. Back to the SSEG, for part two, I followed the diagram in Fig. 3). The SSEG pins D1 to D4 (four digits) were set to port 4 pins 0 to 3 respectively. The Segments were instead outputs of the shift register (Fig. 2) pins Q_A through Q_G. Wiring for the shifter pins was followed according to Fig. 3. Also the buttons for incrementing and decrementing the SSEG were kept as external, positive logic buttons connected to port 6 pins 0 and 1 respectively. The buttons that were used would debounce for a long period of time (roughly a third of a second) and the sweet spot SysTick wait value would differ between the increment and decrement buttons (1100000 for increment, 1200000 for decrement).

For part one of this lab, the only real issue was finding out that my circuit only worked if the D1 pin was set to high (common anode) and not the intended low GND value (common cathode). Other than that, the program worked as intended. The trick is to test each segment of the SSEG so you know what 8-bit value to send to P4OUT to get it to light the correct A-G segment. For my case, setting a bit to zero would power a certain segment, and setting a bit to one would power it off, as seen in the "#define" section of "SSEG.c" file. Once those are tested, you set more "#define" values as an AND combination of each segment in order to make the digits 0 through 9. The rest of the code is simple, the "case" statement will take the single digit integer input and will link the integer ranging from 0-9 with the previously defined SSEG 8-bit value. For this part and the next part, I use an integer variable called "counter" which increments or decrements whenever the user enters a corresponding button. If the user enters a button, counter will overwrite the previous number. The only drawback that I see is that I must output a starting number to the SSEG before the user enters a inc/dec button press. Also, I edited the prototype functions in "SSEG.h" for both parts and removed any usage of the data type "char" since I felt more comfortable working with integers.

Part two was the most time-consuming part of this lab, and required a brand new circuit with the use of a shift register. Putting the menu that asks for user input aside, the flow of "SSEG.c" goes like the following. "**SSEG_Disp_Num**(**int** num)" will accept the user inputted value (after it is updated by "counter"), and its job is to separate the four digit integer into individual, separated digits. First, I created a typical digit counter that keeps dividing the number by ten until it reaches zero and incrementing the "digits" variable. I used my knowledge from lab 5 to create a "decrement for-loop" that takes the modulus-10 of the integer on the far right and stores the output into a "bits" array (defined globally) starting from the right until it reaches zero. Another for-loop is used to determine the value of "counter" by converting the "bits" array into an integer, for later use. Finally, the digit count is passed to **SSEG_Out**(**int** digits) which will use a case statement (dependent on how many digits in "bits") to determine the respective number to output to the SSEG (using the defined segment values from part 1). Before we dissect **SSEG_Shift_Out**(**int** data), its worth noting how to determine which digit to power on. The trick is to use if-statements on the digits variable. If digits is four, for example, than the furthest SSEG digit is powered on using P4OUT. The digit variable is then decremented for every reiteration of the outer for-loop, which means every digit is taken into account. It's also worth noting there is yet another delay that is added after each digit is powered on. The lecture slides suggest to delay it to .25 ms, but that is too fast for my particular SSEG and I needed a much higher time to wait to be able to legibly read the SSEG. Now, the **SSEG_Shift_Out**(**int** data) function takes advantage of the shift register's ability to output each segment in parallel by sending it a serial byte of predefined segment number values. It is very important to read the datasheet for this particular part (especially the chart in Fig. 4) before getting started with this. The SR clock must be active during the duration of the SER datastream, so I called the SR clock function before every iteration of the outer for-loop. Then if the data happens to start with bit "1", it will output to the shifter via P5OUT. Otherwise, a bit clear operation will occur using P5OUT again. It is also important to perform a shift operation on the data input in order to traverse all eight bits. After this for-loop completes, the latch will be thrown by calling the R clock function to pulse the clock only once, for that is all we need at this time. When all is said and done and if one wants to run this program, the SSEG will display the number but only for a split second, and will only keep the last digit. In order to see the number in its full length, it is necessary to continuously loop through the program using a while(1) statement.

The user must enter a value between 0 and 9999, which will then be diced into individual digits and then displayed onto the SSEG. This menu is done in "main.c" and is far from complex. If the user enters an integer outside of this range it will prompt the user to enter a new, valid number. If the user decides to be rebellious and enter a character or any non-integer, the program will behave erratically and will therefore be unusable and will require a restart. The C functions "printf" and "scanf" were used to gather user input. Sadly, the "scanf" function will halt the CCS debugger once called, so it is impossible to implement the instruction of allowing the user to enter a new number while still debugging. Anyways, once the number has been recognized as a valid input, the program will now loop and will judge whether a increment or decrement button has been pressed. More or less, the same logic from part one applies here when it comes to edge cases and debounce wait times.

In the end, the part 2 worked as expected. The only requirement it did not meet was the one about entering a new number while debugging. Again, this is impossible with the current setup and requires another method like UART to be implemented properly.

- Is it possible to create a shift register program without the physical shift register? Yes, with the use of a multi-threaded program this is a possibility. This is because the segments A though G are sent in parallel to the SSEG, and threading is the closest we have to true parallelism in programing.

Conclusion

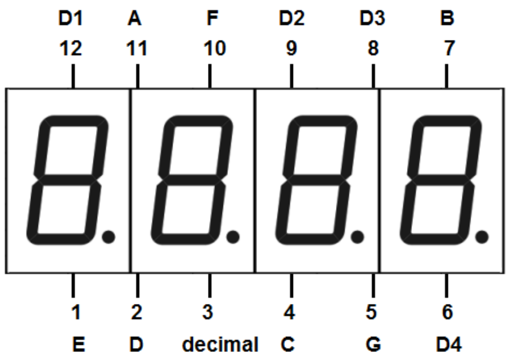
This lab taught me how to control a 7-segment display using a shift register. As mentioned before, working with SSEG displays is a lot simpler than most displays, and will become of use in future sophisticated embedded projects. If I was to do this lab again, I would skip part 1 since part 2 used a different circuit and not much of the preexisting code was used in the second part. I would also of avoided spending too much time on peculiar minor bugs and also figuring out the part where we had to let the user input a new number while the program was debugging.

References

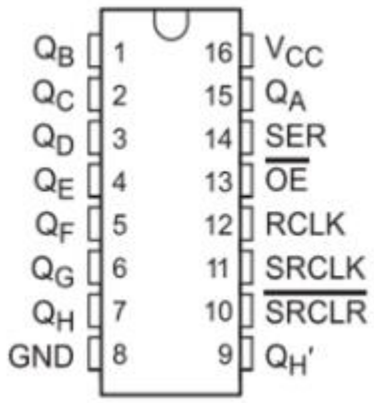
3461BS data sheet, SN74HC595N data sheet, lecture slides, and classmate Levi Randall.

Appendix

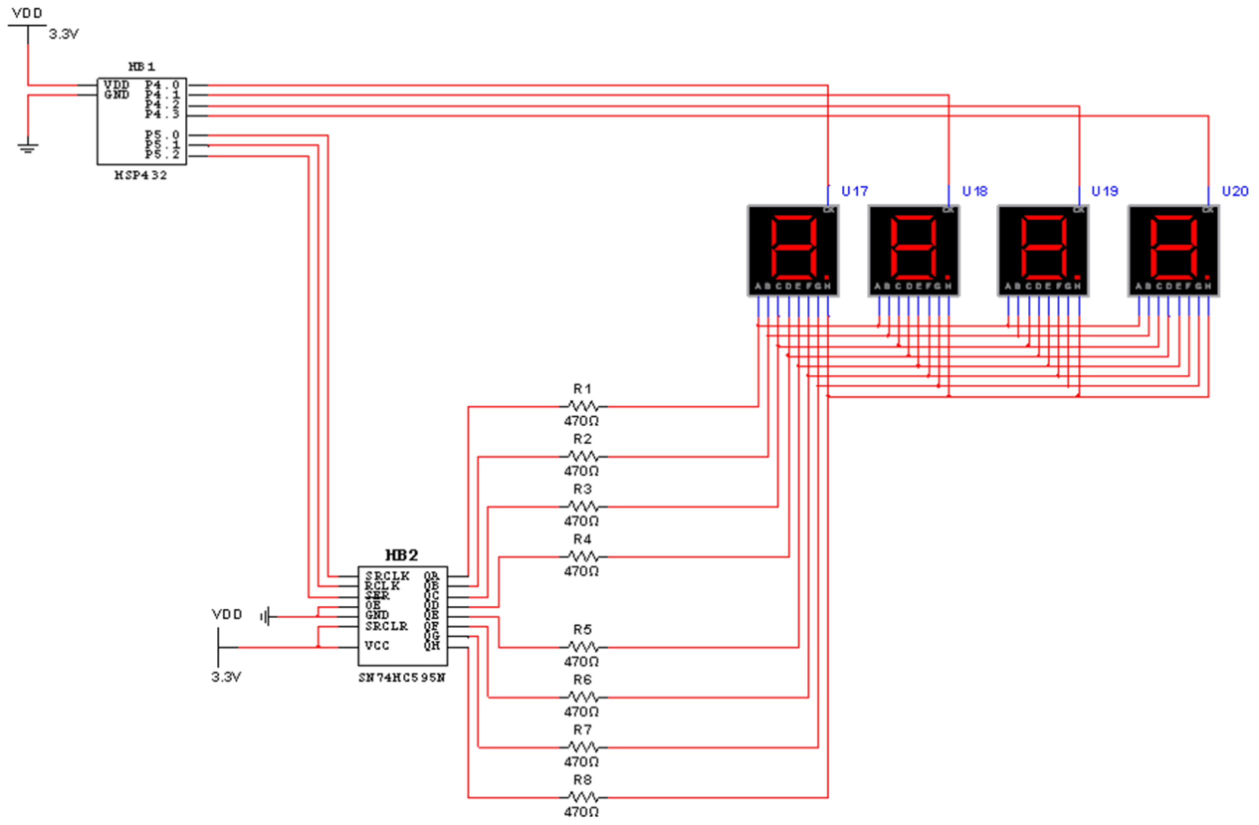
Images:



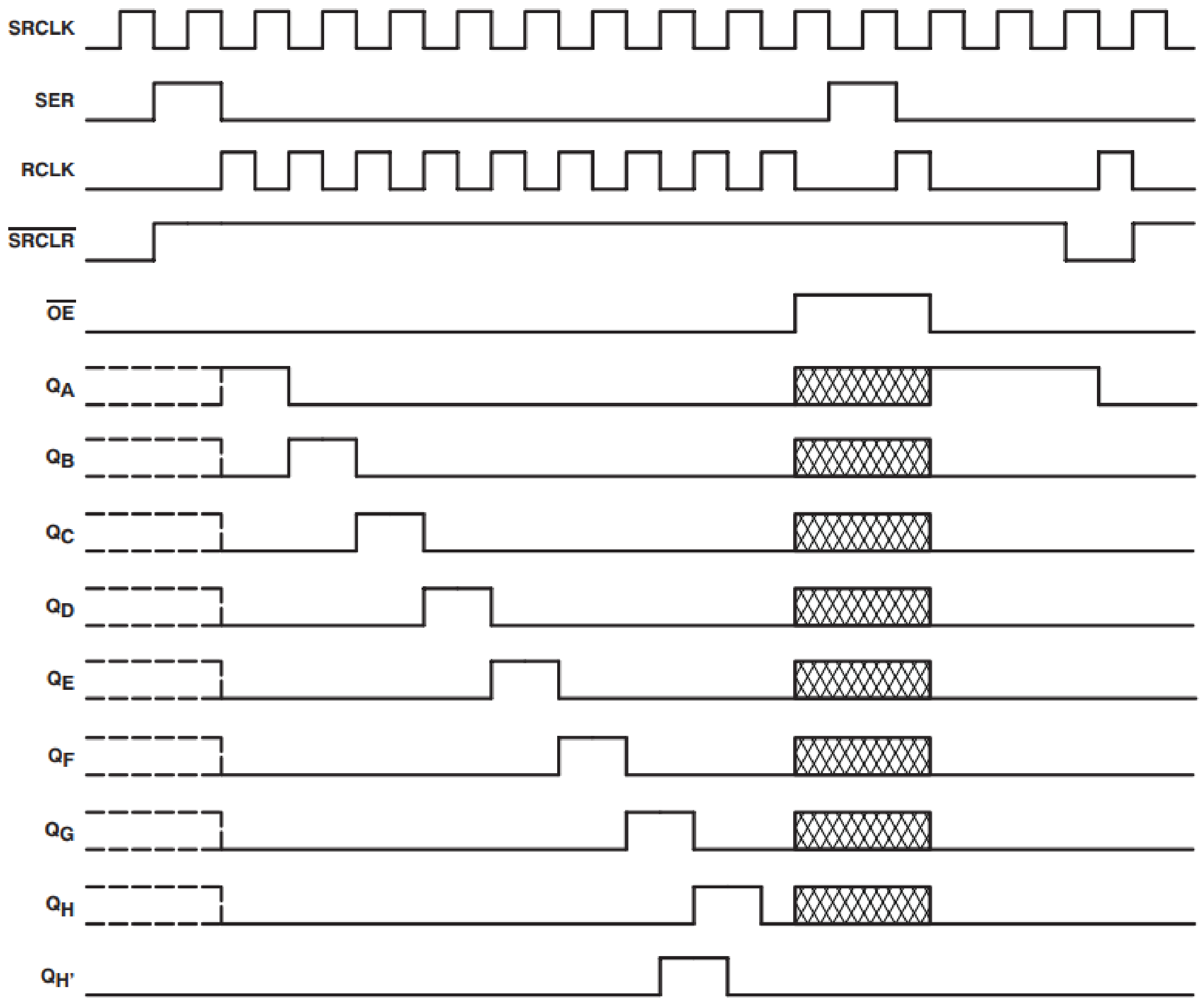
- Fig. 1: The 3461BS 7-Segment Display Pinout Diagram.



- Fig. 2: SN74HC595N 8-bit Shift Register IC pinout Diagram.



- Fig. 3: Part 2 Overall Circuit Diagram.



- Fig. 4: Shift Register Timing Diagram.

Part 1 Pseudocode:

```
• Initialize SysTick Timer.
    o Initialize increment/decrement buttons.
• Initialize the 1 digit 7-segment display driver.
• Output to SSEG a starting number.
• Start infinite while loop:
    o Check if the user pressed the increment button.
```

        - If so, did it increment to 10? Set to zero and output it, otherwise output current number.
    - Check if the user pressed the decrement button.
        - If so, did it decrement to 0? Set to nine and output it, otherwise output current number.
    - Debounce button by waiting a certain time for the metal contacts to steady.
- Repeat loop.

Part 2 Pseudocode:

- Initialize SysTick Timer.
- Initialize the 4 digit 7-segment display and shift register drivers.
    - Initialize increment/decrement buttons and clear display.
- Ask user to enter integer between 0 and 9999. The user enters a value. If the value is not within this range, it will prompt the user to enter another valid integer.
- Start infinite while loop:
    - Check if the user pressed the increment button.
        - If so, is the current value 9999? Yes, set SSEG to zero, otherwise increment counter.
        - Debounce button by waiting a certain time for the metal contacts to steady.
    - Check if the user pressed the decrement button.
        - If so, is the current value 0? Yes, set SSEG to nine thousand nine hundred and ninety-nine, otherwise decrement counter.
        - Debounce button by waiting a certain time for the metal contacts to steady.
    - If no buttons have been pressed, set counter to the user-inputted value.
- Repeat loop.

Source code:

- (Note) **SysTick.c** and **SysTick.h** were used in both parts but not edited in any way, so they will not be included below just to save space.

PART I

main.c

```
#include "msp432p401r.h"
#include "SysTick.h"
#include "SSEG.h"

void main() {
   SysTick_Init();
   SSEG_Init();
   SSEG_Out(0);
   while(1){
      PORT6_IRQHandler();
   }
}
```

SEGG.h

```
/**************** Public Functions ****************/
/*
 * SSEG_Init Function
 * Initialize 7-segment display
 * Inputs: none
 * Outputs: none
 */
void SSEG_Init();
```

```
/*
 * SSEG_Out Function
 * Output a number to a single digit of the 7-segment display
 * Inputs: a number between 0 and 15
 * Outputs: none
 */
void SSEG_Out(uint8_t num);

/*
 * Port 6 ISR to debounce switches and inc/dec SSEG
 * Uses P6IV to solve critical section/race
 */
void PORT6_IRQHandler();
```

SSEG.c

```
#include <stdint.h>
#include "SysTick.h"
#include "msp432p401r.h"
#include "SSEG.h"

#define G 0x3F // Segment g  ON = 0011 1111
#define F 0x5F // Segment f  ON = 0101 1111
#define E 0x6F // Segment e  ON = 0110 1111
#define D 0x77 // Segment d  ON = 0111 0111
#define C 0x7B // Segment c  ON = 0111 1011
#define B 0x7D // Segment b  ON = 0111 1101
#define A 0x7E // Segment a  ON = 0111 1110
```

```
#define ONE B & C
#define TWO A & B & D & E & G
#define THREE A & B & C & D & G
#define FOUR B & C & F & G
#define FIVE A & C & D & F & G
#define SIX A & C & D & E & F & G
#define SEVEN A & B & C
#define EIGHT A & B & C & D & E & F & G
#define NINE A & B & C & D & F & G
#define ZERO A & B & C & D & E & F

uint32_t counter = 0;

void DisableInterrupts();              // Disable interrupts
void EnableInterrupts();               // Enable interrupts
long StartCritical ();                 // previous I bit, disable interrupts
void EndCritical(long sr);             // restore I bit to previous value
void WaitForInterrupt();               // low power mode

/****************************************
 *       PRIVATE FUNCTIONS
 ****************************************/
void Button_Init(void){
   P6->SEL0 &= ~0x3;          // 0011 two buttons (extern, pos)
   P6->SEL1 &= ~0x3;
   P6->DIR &= ~0x3;           // as input
}
uint32_t Button_In(void){        // private function
```

```
    return P6->IN & 0x3;            // extract those two bits
}

/****************************************
 *        PUBLIC FUNCTIONS
 ****************************************/
/*
 * SSEG_Init Function
 * Initialize 7-segment display
 * Inputs: none
 * Outputs: none
 */
void SSEG_Init() {
    Button_Init();
    P4->SEL0 &= ~0x7F;              // 0111 1111
    P4->SEL1 &= ~0x7F;              // as GPIO
    P4->DIR |= 0x7F;                // as Out
}

/*
 * SSEG_Out Function
 * Output a number to a single digit of the 7-segment display
 * Inputs: a number between 0 and 9
 * Outputs: none
 */
void SSEG_Out(uint8_t num) {
    switch(num) {
        case 1:
```

```
    P4->OUT = ONE;
    break;
case 2:
    P4->OUT = TWO;
    break;
case 3:
    P4->OUT = THREE;
    break;
case 4:
    P4->OUT = FOUR;
    break;
case 5:
    P4->OUT = FIVE;
    break;
case 6:
    P4->OUT = SIX;
    break;
case 7:
    P4->OUT = SEVEN;
    break;
case 8:
    P4->OUT = EIGHT;
    break;
case 9:
    P4->OUT = NINE;
    break;
case 0:
    P4->OUT = ZERO;
```

```
            break;
        default:
            P4->OUT = 0x7F;         // LED off
            break;
    }
}

/*
 * Port 6 ISR to debounce switches and inc/dec SSEG
 * Uses P6IV to solve critical section/race
 */
void PORT6_IRQHandler() {
    if (Button_In() == 1){          // 0001 for increment SSEG
        if (counter == 10){
            counter = 0;
            SSEG_Out(counter);
        } else {
        counter++;
        SSEG_Out(counter);
        }
        /*
        switch(P4->OUT) {
            case ZERO:
                P4->OUT = ONE;
                break;
            case ONE:
                P4->OUT = TWO;
                break;
    case TWO:
        P4->OUT = THREE;
        break;
    case THREE:
        P4->OUT = FOUR;
        break;
    case FOUR:
        P4->OUT = FIVE;
        break;
    case FIVE:
        P4->OUT = SIX;
        break;
    case SIX:
        P4->OUT = SEVEN;
        break;
    case SEVEN:
        P4->OUT = EIGHT;
        break;
    case EIGHT:
        P4->OUT = NINE;
        break;
    case NINE:
        P4->OUT = ZERO;
        break;
    default:
        P4->OUT = 0x7F;         // LED off
        break;
}*/
```

```
}
else if (Button_In() == 2){     // 0010 for decrement SSEG
    if (counter == 0){
        counter = 9;
        SSEG_Out(counter);
    } else {
    counter--;
    SSEG_Out(counter);
    }
    /*
    switch(P4->OUT) {
        case ZERO:
            P4->OUT = NINE;
            break;
        case ONE:
            P4->OUT = ZERO;
            break;
        case TWO:
            P4->OUT = ONE;
            break;
        case THREE:
            P4->OUT = TWO;
            break;
        case FOUR:
            P4->OUT = THREE;
            break;
        case FIVE:
            P4->OUT = FOUR;
            break;
          case SIX:
            P4->OUT = FIVE;
            break;
          case SEVEN:
            P4->OUT = SIX;
            break;
          case EIGHT:
            P4->OUT = SEVEN;
            break;
          case NINE:
            P4->OUT = EIGHT;
            break;
          default:
            P4->OUT = 0x7F;         // LED off
            break;
        }*/
    }
    SysTick_Wait(1100000);            // debounce switches by about 366.7ms
}
```

PART II

main.c

```
// LAB 6 By James Samawi 12/18/2020
#include "msp432p401r.h"
#include "SysTick.h"
#include "SSEG.h"
```

```
#include <stdio.h> //printf, scanf

void main() {
    int userinput = 0;
    SysTick_Init();
    SSEG_Init();
    printf("\nEnter an integer between 0 and 9999: ");
    scanf("%d", &userinput);
    if (userinput < 0 || userinput > 9999){
        printf("Error: Entered value does not lie between 0 and 9999. Try again.\n");
        scanf("%d", &userinput);
    }
    else {
        PORT6_IRQHandler(userinput);
    }
}
```

SSEG.h

```
/**************** Public Functions ****************/
/*
 * SSEG_Init Function
 * Initialize 7-segment display
 * Inputs: none
 * Outputs: none
 */
void SSEG_Init();

/*
```

```
 * Port 6 ISR to debounce switches and inc/dec SSEG
 * Uses P6IV to solve critical section/race
 */
void PORT6_IRQHandler(int userinput);

/*
 * SSEG_Disp_Num Function
 * Separate the input number into 4 single digit
 * Inputs: num between 0 and 9999
 * Outputs: none
 */
void SSEG_Disp_Num(int num);

/*
 * SSEG_Out Function
 * Output a 4-digit number to the display
 * Inputs: digits
 * Outputs: none
 */
void SSEG_Out(int digits);

/*
 * SSEG_Shift_Out Function
 * Shifts data out serially
 * Inputs: 8-bit data
 * Outputs: none
 */
void SSEG_Shift_Out(int data);
```

SSEG.c

```
#include <stdint.h>
#include "SysTick.h"
#include "msp432p401r.h"
#include "SSEG.h"

#define A 0x3F  // Segment A  ON = 0011 1111
#define B 0x5F  // Segment B  ON = 0101 1111
#define C 0x6F  // Segment C  ON = 0110 1111
#define D 0x77  // Segment D  ON = 0111 0111
#define E 0x7B  // Segment E  ON = 0111 1011
#define F 0x7D  // Segment F  ON = 0111 1101
#define G 0x7E  // Segment G  ON = 0111 1110

#define  ONE    B & C
#define  TWO    A & B & D & E & G
#define  THREE  A & B & C & D & G
#define  FOUR   B & C & F & G
#define  FIVE   A & C & D & F & G
#define  SIX    A & C & D & E & F & G
#define  SEVEN  A & B & C
#define  EIGHT  A & B & C & D & E & F & G
#define  NINE   A & B & C & D & F & G
#define  ZERO   A & B & C & D & E & F

int bits[4];
int counter;
int difference = 0;

/****************************************
 *        PRIVATE FUNCTIONS
 ****************************************/
void Button_Init(void){
   P6->SEL0 &= ~0x3;              // 0011 two buttons (extern, pos)
   P6->SEL1 &= ~0x3;
   P6->DIR  &= ~0x3;              // as input
}
uint32_t Button_In(void){           // private function
   return P6->IN & 0x3;           // extract those two bits
}

void SRCLKPulse(void){              // Pulse the Shift Register Clock
   P5OUT |=  0x1;                 // HIGH
   P5OUT &= ~0x1;                 // LOW
}

void RCLKPulse(void){               // Pulse the Latch Clock, inverse of SR Clock
   P5OUT |=  0x2;                 //HIGH
   P5OUT &= ~0x2;                 //LOW
}

/*
void SSEG_On(){
   P4->OUT = 0x0F;                // All digits on
}
```

```
*/

void SSEG_Off(){
    P4->OUT = 0x00;                  // All digits off
}

/****************************************
 *          PUBLIC FUNCTIONS
 ****************************************/
/*
 * SSEG_Init Function
 * Initialize 7-segment display
 * Inputs: none
 * Outputs: none
 */
void SSEG_Init() {
    Button_Init();

    // For common cathode digit outputs:
    P4->SEL0 &= ~0x0F;              // 0000 1111
    P4->SEL1 &= ~0x0F;              // as GPIO
    P4->DIR  |=  0x0F;              // as OUTPUT
    // 0x1 = D1, 0x2 = D2, 0x4 = D3, 0x8 = D4

    SSEG_Off();

    // For Shift Register outputs:
    P5->SEL0 &= ~0x07;              // 0000 0111
    P5->SEL1 &= ~0x07;              // as GPIO
    P5->DIR  |=  0x07;              // as OUTPUT
    // 0x1 = SRCLK, 0x2 = RCLK, 0x4 = SER
}

/*
 * Port 6 ISR to debounce switches and inc/dec SSEG
 * Uses P6IV to solve critical section/race
 */
void PORT6_IRQHandler(int userinput) {
    while(1){                                   // loop here because counter had trouble updating its value
        if (Button_In() == 1){                  // 0001 for increment SSEG
            if (counter == 9999){
                userinput = 0;
                difference = 0;
            } else {
            counter++;
            difference++;
            }
            SysTick_Wait(1100000);              // debounce switches by about 366.7ms
        }
        else if (Button_In() == 2){             // 0010 for decrement SSEG
            if (counter == 0){
                userinput = 9999;
                difference = 0;
            } else {
            counter--;
            difference--;
            }
            SysTick_Wait(1200000);              // debounce switches by about 366.7ms
        } else {
            counter = userinput + difference;
            SSEG_Disp_Num(counter);
        }
    }
}

/*
 * SSEG_Disp_Num Function
 * Separate the input number into 4 single digit
 * Inputs: num between 0 and 9999
 * Outputs: none
 */
void SSEG_Disp_Num(int num){        // bits[] defined globally
    int num_temp = num;             // stores num divided by 10
    int digits = 0;                 // clear digits
    bits[0] = 0;                    // clear bits
    bits[1] = 0;
    bits[2] = 0;
    bits[3] = 0;
    while (num_temp > 0) {          // digit counter
        num_temp = num_temp/10;
        digits ++;
    }
    if (num == 0) digits = 1;       // the zero case
    int temp_n = num;               // stores num modulus 10
    int i;
    for(i = digits-1; i >= 0; i--) {
        temp_n = num %10;           // store modulus values backwards by decrementing
        bits[i] = temp_n;
        num /= 10;
    }
    counter = 0;                    // must be zero or else it will reuse previous counter value
    for (i = 0; i < digits; i++){
        counter = 10 * counter + bits[i]; // string to int conversion
    }
    SSEG_Out(digits);
}

/*
 * SSEG_Out Function
 * Output a 4-digit number to the display
 * Inputs: digits
 * Outputs: none
 */
void SSEG_Out(int digits) {
    int i;
    int digits_clone = digits;
    for (i = 0; i < digits; i++){
        switch(bits[i]) {
            case 1:
                SSEG_Shift_Out(ONE);
                break;
            case 2:
```

```
    SSEG_Shift_Out(TWO);
    break;
case 3:
    SSEG_Shift_Out(THREE);
    break;
case 4:
    SSEG_Shift_Out(FOUR);
    break;
case 5:
    SSEG_Shift_Out(FIVE);
    break;
case 6:
    SSEG_Shift_Out(SIX);
    break;
case 7:
    SSEG_Shift_Out(SEVEN);
    break;
case 8:
    SSEG_Shift_Out(EIGHT);
    break;
case 9:
    SSEG_Shift_Out(NINE);
    break;
case 0:
    SSEG_Shift_Out(ZERO);
    break;
default:
    SSEG_Off();
```

```
            break;
        }//end switch
        if (digits_clone == 4){
            P4->OUT = 0x1;         // first digit
            SysTick_Wait(3000);      // .8333 ms
            }
        else if (digits_clone == 3){
            P4->OUT = 0x2;         // second digit
            SysTick_Wait(3000);      // .8333 ms
            }
        else if (digits_clone == 2){
            P4->OUT = 0x4;         // third digit
            SysTick_Wait(3000);      // .8333 ms
            }
        else if (digits_clone == 1){
            P4->OUT = 0x8;         // fourth digit
            SysTick_Wait(3000);      // .8333 ms
            }
        digits_clone--;
    }//end for
}//end func

/*
 * SSEG_Shift_Out Function
 * Shifts data out serially
 * Inputs: 8-bit data
 * Outputs: none
 */
```

```
void SSEG_Shift_Out(int data){      // input: 1 byte, ZERO through NINE.
    int i;
    for(i=0;i<8;i++){               // Output 8 bits to SER
        SRCLKPulse();               // Continuously pulse the SR Clock, must be low before data
        if((data & 0x01) == 0x01){  // if output high = LED seg on
            P5->OUT |= 0x4;
        }else{                      // output low = LED seg off
            P5->OUT &= ~0x4;
        }
        data=data>>1;               // Shift to next bit
    }
    RCLKPulse();                    // output latch after all 8 bits transfered, must be high after data
}
```