Question 1:

main.c

```
#include <stdint.h>
#include "SysTick.h"
#include "PLL.h"
#include "tm4c123gh6pm.h"
#include "Timer0A.h"


void GPIOPortB_Init() {
   SYSCTL_RCGC2_R |= 0b10;                  // 1) turns on clock B
   GPIO_PORTB_AMSEL_R &= ~(0xFF);  // 3) disable analog function on PB5-0
   GPIO_PORTB_PCTL_R &= ~(0xFF);      // 4) enable regular GPIO
   GPIO_PORTB_DIR_R |= 0xFF;              // 5) outputs on PB5-0
   GPIO_PORTB_AFSEL_R &= ~(0xFF);   // 6) regular function on PB5-0
   GPIO_PORTB_DEN_R |= 0xFF;             // 7) enable digital on PB5-0
}

void GPIOPortE_Init() {
   SYSCTL_RCGC2_R |= 0b10000;            // 1) turns on clock for E
   SysTick_Wait10ms(2);
   GPIO_PORTE_AMSEL_R &= ~(0xFF);   // 3) disable analog function on PE2-0
   GPIO_PORTE_PCTL_R &= ~(0xFF);      // 4) enable regular GPIO
   GPIO_PORTE_DIR_R &= ~(0xFF);        // 5) inputs on PE2-0
   GPIO_PORTE_AFSEL_R &= ~(0xFF);  // 6) regular function on PE2-0
   GPIO_PORTE_DEN_R |= 0xFF;            // 7) enable digital on PE2-0

   NVIC_EN0_R |= 1 << 4;            // set 1 to 4th bit
   NVIC_PRI4_R |= 0;

   GPIO_PORTE_IM_R |= (1<<0)|(1<<1)|(1<<3);             // enable interrupts
   GPIO_PORTE_IS_R &= ~(1<<0)|~(1<<1)|~(1<<3);     // enable edge trigger
   GPIO_PORTE_IEV_R &= (1<<0)|(1<<1)|(1<<3);        // enable negative trigger
   GPIO_PORTE_IBE_R &= ~(1<<0)|~(1<<1)|~(1<<3);
}


/* Seven-segment display counter
 *
 * This program counts number 0-3 on the seven segment display.
```

```
 * The seven segment display is driven by a shift register which is
 * connected to SSI2 in SPI mode.
 *
 * Built and tested with Keil MDK-ARM v5.28 and TM4C_DFP v1.1.0
 */

// #include "TM4C123.h" // use diff header files

void delayMs(int n);
void sevenseg_init(void);
void SSI2_write(unsigned char data);

int main(void) {

    sevenseg_init();    // initialize SSI2 that connects to the shift registers
    SysTick_Init();
    Timer0A_Init(80000000);
    GPIOPortE_Init();

    Timer0A_Wait(2);

    while(1) {
        // was moved to Timer0A
        // here to catch processor
    }

}

// enable SSI2 and associated GPIO pins (need to change to fit our board)
void sevenseg_init(void) {
    SYSCTL_RCGC2_R |= 0x02;   // enable clock to GPIOB (change first part to
SYSCTC_RCGCGPIO_R)
    SYSCTL_RCGC2_R |= 0x04;   // enable clock to GPIOC
    SYSCTL_RCGCSSI_R |= 0x04;    // enable clock to SSI2

    // PORTB 7, 4 for SSI2 TX and SCLK
    GPIO_PORTB_AMSEL_R &= ~0x90;          // turn off analog of PORTB 7, 4
    GPIO_PORTB_AFSEL_R |= 0x90;              // PORTB 7, 4 for alternate function
    GPIO_PORTB_PCTL_R &= ~0xF00F0000; // clear functions for PORTB 7, 4
    GPIO_PORTB_PCTL_R |= 0x20020000;      // PORTB 7, 4 for SSI2 function
```

```
    GPIO_PORTB_DEN_R |= 0x90;                   // PORTB 7, 4 as digital pins (change first part to
GPIO_PORTB_DEN_R)

    // PORTC 7 for SSI2 slave select
    GPIO_PORTC_AMSEL_R &= ~0x80;      // disable analog of PORTC 7
    GPIO_PORTC_DATA_R |= 0x80;            // set PORTC 7 idle high
    GPIO_PORTC_DIR_R |= 0x80;               // set PORTC 7 as output for SS
    GPIO_PORTC_DEN_R |= 0x80;              // set PORTC 7 as digital pin

    SSI2_CR1_R = 0;                   // turn off SSI2 during configuration
    SSI2_CC_R = 0;                     // use system clock
    SSI2_CPSR_R = 16;              // clock prescaler divide by 16 gets 1 MHz clock
    SSI2_CR0_R = 0x0007;        // clock rate div by 1, phase/polarity 0 0, mode freescale, data
size 8
    SSI2_CR1_R = 2;                  // enable SSI2 as master
}


// This function enables slave select, writes one byte to SSI2,
// wait for transmit complete and deassert slave select.
/*void SSI2_write(unsigned char data) {
    GPIO_PORTC_DATA_R &= ~0x80;       // assert slave select
    SSI2_DR_R = data;                              // write data
    while (SSI2_SR_R & 0x10) {}               // wait for transmit done
    GPIO_PORTC_DATA_R |= 0x80;          // deassert slave select
}*/

// get rid of below since not set for 80 MHz
/* delay n milliseconds (50 MHz CPU clock) */
/*void delayMs(int n) {
    int i, j;
    for(i = 0 ; i< n; i++)
        for(j = 0; j < 6265; j++)
            {}  // do nothing for 1 ms
}*/
```

Timer0A.c

```
// Timer0A.c
// Runs on Tiva-C
```

```
// Adapted from SysTick.c from the book:
/* "Embedded Systems: Introduction to MSP432 Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
   Volume 1, Program 4.7
*/

#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "Timer0A.h"
#include "SysTick.h"

uint32_t sysClkFreq = 80000000;                    // Assume 80 MHz clock by default

// Set clock freq. so Timer0A_Wait10ms delays for exactly 10 ms if clock is not 80 MHz
void Timer0A_Init( uint32_t clkFreq ){
  sysClkFreq = clkFreq;
  SYSCTL_RCGCTIMER_R |= 0x00000001;  // 0) Activate Timer0
  TIMER0_CTL_R &= ~0x00000001;           // 1) Disable Timer0A during setup
  TIMER0_CFG_R = 0;                              // 2) Configure for 32-bit timer mode
  TIMER0_TAMR_R = 1;                           // 3) Configure for one-shot mode, only calls
interrupt once
  TIMER0_TAPR_R = 0;                            // 5) No prescale
  TIMER0_IMR_R = 1;                              // 6-9) Yes interrupts

  // NVIC
  NVIC_EN0_R |= 1 << 19;                      // set 1 to 19th bit
  NVIC_PRI4_R |= 1 << 29;

  return;
}

// Time delay using busy wait
// The delay parameter is in units of the core clock (units of 12.5 nsec for 80 MHz clock)
//   Adapted from Program 9.8 from the book:
/*   "Embedded Systems: Introduction to ARM Cortex-M Microcontrollers",
     ISBN: 978-1477508992, Jonathan Valvano, copyright (c) 2013
     Volume 1, Program 9.8
*/
void Timer0A_Wait( uint32_t delay ){
```

```
  if(delay <= 1){ return; } // Immediately return if requested delay less than one clock
  TIMER0_TAILR_R = delay - 1;                 // 4) Specify reload value
  TIMER0_CTL_R |= 0x00000001;                 // 10) Enable Timer0A

  //while( TIMER0_TAR_R ){} // Doesn't work; Wait until timer expires (value equals 0)
  // Or, clear interrupt and wait for raw interrupt flag to be set
  // in busy wait mode v want to let timer interrupt the program

  // while( !(TIMER0_RIS_R & 0x1) ){}
  return;
}

// Time delay using busy wait
// This assumes 80 MHz system clock
void Timer0A_Wait1ms( uint32_t delay ){
  uint32_t i;
  for( i = 0; i < delay; i++ ){
    Timer0A_Wait(sysClkFreq/1000);  // wait 1ms
  }
  return;
}

void SSI2_write(unsigned char data) {
    GPIO_PORTC_DATA_R &= ~0x80;       // assert slave select
    SSI2_DR_R = data;                 // write data
    while (SSI2_SR_R & 0x10) {}       // wait for transmit done
    GPIO_PORTC_DATA_R |= 0x80;        // deassert slave select
}

/* delay n milliseconds (50 MHz CPU clock) */
void delayMs(int n) {
    int i, j;
    for(i = 0 ; i< n; i++)
        for(j = 0; j < 6265; j++)
            {}  /* do nothing for 1 ms */
}

// sets i and x
const static unsigned char digitPattern[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
0x80, 0x90};
```

```
int i = 0;
int x1 = 0;
int x2 = 0;
int x3 = 0;
int x4 = 0;
int check = 0;

void Timer_Handle(){
   if (!check) {
      SSI2_write(digitPattern[x4]);    // write digit pattern to the seven segments
      SSI2_write((1 << 0));            // select digit
      SysTick_Wait1ms(2);              // give time for LED

      SSI2_write(digitPattern[x3]);    // write digit pattern to the seven segments
      SSI2_write((1 << 1));            // select digit
      SysTick_Wait1ms(2);

      SSI2_write(digitPattern[x2]);    // write digit pattern to the seven segments
      SSI2_write((1 << 2));            // select digit
      SysTick_Wait1ms(2);

      SSI2_write(digitPattern[x1] ^ 0x80);    // write digit pattern to the seven segments
      SSI2_write((1 << 3));                   // select digit
      SysTick_Wait1ms(2);
   } else if (check) {
      SSI2_write(digitPattern[x4]);    // write digit pattern to the seven segments
      SSI2_write((1 << 0));            // select digit
      if (++x4 > 9) {
         x4 = 0;                 // fixes digit
         ++x3;                   // ups z3 by 1
      }
      SysTick_Wait1ms(2);

      SSI2_write(digitPattern[x3]);    // write digit pattern to the seven segments
      SSI2_write((1 << 1));            // select digit
      if (x3 > 9) {
         x3 = 0;
         ++x2;
      }
      SysTick_Wait1ms(2);
```

```
        SSI2_write(digitPattern[x2]);    // write digit pattern to the seven segments
        SSI2_write((1 << 2));            // select digit
        if (x2 > 9) {
            x2 = 0;
            ++x1;
        }
        SysTick_Wait1ms(2);

        SSI2_write(digitPattern[x1] ^ 0x80);    // write digit pattern to the seven segments
        SSI2_write((1 << 3));                   // select digit
        if (x1 > 9) {
            x1 = 0;
        }
        SysTick_Wait1ms(2);
    }
    //if (++i > 9) {
    //    i = 0;
    //}
    //if (++x1 > 3){
    //    x1 = 0;
    //}
    //delayMs(2);                       // 1000 / 60 / 4 = 4.17

    TIMER0_ICR_R = 1;              // interrupt flag addy, clear bit in here to no handle
    SysTick_Wait1ms(2);
    Timer0A_Wait(2);

}

void Port_Handle() {
    SysTick_Wait10ms(2);
    if (GPIO_PORTE_MIS_R & 0b0001){         // if start/pause was pressed (PE0)
        // call same functions but repeat the value
        if (check) {
            check = 0;
        } else if (!check) {
            check = 1;
        }
```

```
        GPIO_PORTE_ICR_R|= 0b0001;
    } else if (GPIO_PORTE_MIS_R & 0b0010){   // if increment was pressed (PE1)
        if (!check) {
            check = 1;
            Timer_Handle();                          // note: test if will work, if not copy whole
function here
            check = 0;
        } else {
            Timer_Handle();
        }

        GPIO_PORTE_ICR_R |= 0b0010;
    } else if (GPIO_PORTE_MIS_R & 0b1000){   // if reset was pressed (PE3)
        x4 = 0;                                      // fixes digit to 0
        SysTick_Wait1ms(2);

        x3 = 0;
        SysTick_Wait1ms(2);

        x2 = 0;
        SysTick_Wait1ms(2);

        x1 = 0;
        SysTick_Wait1ms(2);
        GPIO_PORTE_ICR_R |= 0b1000;
    }
}
```

Timer0A.h

```
// Timer0A.h
// Runs on Tiva-C

// Adapted from SysTick.h from the book:
/* "Embedded Systems: Introduction to MSP432 Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
   Volume 1, Program 4.7
*/

// protects the files
#ifndef __TIMER0A_H__
```

```
#define __TIMER0A_H__

// Set clock freq. so Timer0A_Wait10ms delays for exactly 10 ms if clock is not 80 MHz
void Timer0A_Init( uint32_t clkFreq );

// Time delay using busy wait
// The delay parameter is in units of the core clock (units of 12.5 nsec for 80 MHz clock)
void Timer0A_Wait( uint32_t delay );

// Time delay using busy wait
// This assumes 80 MHz system clock
void Timer0A_Wait1ms( uint32_t delay );

void Timer_Handle();

#endif
```

tm4c123gh6pm_startup_ccs.c

```
//*****************************************************************************
*
//
// Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2011-2014 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING,
BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
```

```
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
//****************************************************************************
*

#include <stdint.h>

//****************************************************************************
*
//
// Forward declaration of the default fault handlers.
//
//****************************************************************************
*
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

//****************************************************************************
*
//
// External declaration for the reset handler that is to be called when the
// processor is started
//
//****************************************************************************
*
extern void _c_int00(void);

//****************************************************************************
*
//
// Linker variable that marks the top of the stack.
//
//****************************************************************************
*
extern uint32_t __STACK_TOP;
```

```
//*****************************************************************************
*
//
// External declarations for the interrupt handlers used by the application. (put customs here)
//
//*****************************************************************************
*
// To be added by user
extern void Timer_Handle(); // extern tells starter file to look outside for function
extern void Port_Handle();
//*****************************************************************************
*
//
// The vector table.  Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000 or at the start of
// the program if located at a start address other than 0.
//
//*****************************************************************************
*
#pragma DATA_SECTION(g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_TOP),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // SVCall handler
    IntDefaultHandler,                      // Debug monitor handler
    0,                                      // Reserved
    IntDefaultHandler,                      // The PendSV handler
    IntDefaultHandler,                      // The SysTick handler
    IntDefaultHandler,                      // GPIO Port A
```

```
IntDefaultHandler,      // GPIO Port B
IntDefaultHandler,      // GPIO Port C
IntDefaultHandler,      // GPIO Port D
Port_Handle,            // GPIO Port E
IntDefaultHandler,      // UART0 Rx and Tx
IntDefaultHandler,      // UART1 Rx and Tx
IntDefaultHandler,      // SSI0 Rx and Tx
IntDefaultHandler,      // I2C0 Master and Slave
IntDefaultHandler,      // PWM Fault
IntDefaultHandler,      // PWM Generator 0
IntDefaultHandler,      // PWM Generator 1
IntDefaultHandler,      // PWM Generator 2
IntDefaultHandler,      // Quadrature Encoder 0
IntDefaultHandler,      // ADC Sequence 0
IntDefaultHandler,      // ADC Sequence 1
IntDefaultHandler,      // ADC Sequence 2
IntDefaultHandler,      // ADC Sequence 3
IntDefaultHandler,      // Watchdog timer
Timer_Handle,           // Timer 0 subtimer A, will replace later, what to look for
IntDefaultHandler,      // Timer 0 subtimer B
IntDefaultHandler,      // Timer 1 subtimer A
IntDefaultHandler,      // Timer 1 subtimer B
IntDefaultHandler,      // Timer 2 subtimer A
IntDefaultHandler,      // Timer 2 subtimer B
IntDefaultHandler,      // Analog Comparator 0
IntDefaultHandler,      // Analog Comparator 1
IntDefaultHandler,      // Analog Comparator 2
IntDefaultHandler,      // System Control (PLL, OSC, BO)
IntDefaultHandler,      // FLASH Control
IntDefaultHandler,      // GPIO Port F
IntDefaultHandler,      // GPIO Port G
IntDefaultHandler,      // GPIO Port H
IntDefaultHandler,      // UART2 Rx and Tx
IntDefaultHandler,      // SSI1 Rx and Tx
IntDefaultHandler,      // Timer 3 subtimer A
IntDefaultHandler,      // Timer 3 subtimer B
IntDefaultHandler,      // I2C1 Master and Slave
IntDefaultHandler,      // Quadrature Encoder 1
IntDefaultHandler,      // CAN0
IntDefaultHandler,      // CAN1
```

```
0,                          // Reserved
0,                          // Reserved
IntDefaultHandler,          // Hibernate
IntDefaultHandler,          // USB0
IntDefaultHandler,          // PWM Generator 3
IntDefaultHandler,          // uDMA Software Transfer
IntDefaultHandler,          // uDMA Error
IntDefaultHandler,          // ADC1 Sequence 0
IntDefaultHandler,          // ADC1 Sequence 1
IntDefaultHandler,          // ADC1 Sequence 2
IntDefaultHandler,          // ADC1 Sequence 3
0,                          // Reserved
0,                          // Reserved
IntDefaultHandler,          // GPIO Port J
IntDefaultHandler,          // GPIO Port K
IntDefaultHandler,          // GPIO Port L
IntDefaultHandler,          // SSI2 Rx and Tx
IntDefaultHandler,          // SSI3 Rx and Tx
IntDefaultHandler,          // UART3 Rx and Tx
IntDefaultHandler,          // UART4 Rx and Tx
IntDefaultHandler,          // UART5 Rx and Tx
IntDefaultHandler,          // UART6 Rx and Tx
IntDefaultHandler,          // UART7 Rx and Tx
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
IntDefaultHandler,          // I2C2 Master and Slave
IntDefaultHandler,          // I2C3 Master and Slave
IntDefaultHandler,          // Timer 4 subtimer A
IntDefaultHandler,          // Timer 4 subtimer B
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
0,                          // Reserved
```

```
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
0,                      // Reserved
IntDefaultHandler,      // Timer 5 subtimer A
IntDefaultHandler,      // Timer 5 subtimer B
IntDefaultHandler,      // Wide Timer 0 subtimer A
IntDefaultHandler,      // Wide Timer 0 subtimer B
IntDefaultHandler,      // Wide Timer 1 subtimer A
IntDefaultHandler,      // Wide Timer 1 subtimer B
IntDefaultHandler,      // Wide Timer 2 subtimer A
IntDefaultHandler,      // Wide Timer 2 subtimer B
IntDefaultHandler,      // Wide Timer 3 subtimer A
IntDefaultHandler,      // Wide Timer 3 subtimer B
IntDefaultHandler,      // Wide Timer 4 subtimer A
IntDefaultHandler,      // Wide Timer 4 subtimer B
IntDefaultHandler,      // Wide Timer 5 subtimer A
IntDefaultHandler,      // Wide Timer 5 subtimer B
IntDefaultHandler,      // FPU
0,                      // Reserved
0,                      // Reserved
IntDefaultHandler,      // I2C4 Master and Slave
IntDefaultHandler,      // I2C5 Master and Slave
IntDefaultHandler,      // GPIO Port M
IntDefaultHandler,      // GPIO Port N
IntDefaultHandler,      // Quadrature Encoder 2
0,                      // Reserved
0,                      // Reserved
IntDefaultHandler,      // GPIO Port P (Summary or P0)
IntDefaultHandler,      // GPIO Port P1
IntDefaultHandler,      // GPIO Port P2
IntDefaultHandler,      // GPIO Port P3
IntDefaultHandler,      // GPIO Port P4
```

```
    IntDefaultHandler,                      // GPIO Port P5
    IntDefaultHandler,                      // GPIO Port P6
    IntDefaultHandler,                      // GPIO Port P7
    IntDefaultHandler,                      // GPIO Port Q (Summary or Q0)
    IntDefaultHandler,                      // GPIO Port Q1
    IntDefaultHandler,                      // GPIO Port Q2
    IntDefaultHandler,                      // GPIO Port Q3
    IntDefaultHandler,                      // GPIO Port Q4
    IntDefaultHandler,                      // GPIO Port Q5
    IntDefaultHandler,                      // GPIO Port Q6
    IntDefaultHandler,                      // GPIO Port Q7
    IntDefaultHandler,                      // GPIO Port R
    IntDefaultHandler,                      // GPIO Port S
    IntDefaultHandler,                      // PWM 1 Generator 0
    IntDefaultHandler,                      // PWM 1 Generator 1
    IntDefaultHandler,                      // PWM 1 Generator 2
    IntDefaultHandler,                      // PWM 1 Generator 3
    IntDefaultHandler                       // PWM 1 Fault
};

//****************************************************************************
*
//
// This is the code that gets called when the processor first starts execution
// following a reset event.  Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called.  Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
//
//****************************************************************************
*
void
ResetISR(void)
{
    //
    // Jump to the CCS C initialization routine.  This will enable the
    // floating-point unit as well, so that does not need to be done here.
    //
    __asm("    .global _c_int00\n"
```

```
        "    b.w     _c_int00");
}

//*****************************************************************************
*
//
// This is the code that gets called when the processor receives a NMI.  This
// simply enters an infinite loop, preserving the system state for examination
// by a debugger.
//
//*****************************************************************************
*
static void
NmiSR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
*
//
// This is the code that gets called when the processor receives a fault
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
*
static void
FaultISR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
```

```
    }
}

//*****************************************************************************
*
//
// This is the code that gets called when the processor receives an unexpected
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
*
static void
IntDefaultHandler(void)
{
    //
    // Go into an infinite loop.
    //
    while(1)
    {
    }
}
```

SysTick.c

```
// SysTick.c (Revised for 80 MHz clock)
// Runs on LM4F120/TM4C123
// Provide functions that initialize the SysTick module, wait at least a
// designated number of clock cycles, and wait approximately a multiple
// of 10 milliseconds using busy wait.  After a power-on-reset, the
// LM4F120 gets its clock from the 16 MHz precision internal oscillator,
// which can vary by +/- 1% at room temperature and +/- 3% across all
// temperature ranges.  If you are using this module, you may need more
// precise timing, so it is assumed that you are using the PLL to set
// the system clock to 80 MHz.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// September 11, 2013

/* This example accompanies the books
```

```
  "Embedded Systems: Introduction to ARM Cortex M Microcontrollers",
  ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
  Volume 1, Program 4.7

  "Embedded Systems: Real Time Interfacing to ARM Cortex M Microcontrollers",
  ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015
  Program 2.11, Section 2.6

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include <stdint.h>
#include "tm4c123gh6pm.h"
#define NVIC_ST_CTRL_COUNT      0x00010000  // Count flag
#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode
#define NVIC_ST_RELOAD_M        0x00FFFFFF  // Counter load value

// Initialize SysTick with busy wait running at bus clock
void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;                   // disable SysTick during setup
  NVIC_ST_RELOAD_R = NVIC_ST_RELOAD_M;  // maximum reload value
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
                                        // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC;
}
// Time delay using busy wait
```

```
// The delay parameter is in units of the core clock (units of 12.5 nsec for 80 MHz clock)
void SysTick_Wait(uint32_t delay){
  // method #1: set Reload Value Register, clear Current Value Register, poll COUNTFLAG in
Control and Status Register
  //if(delay <= 1){
    // without this step:
    // if delay == 0, this function will wait 0x00FFFFFF cycles
    // if delay == 1, this function will never return (because COUNTFLAG is set on 1->0
transition)
  //  return;  // do nothing; at least 1 cycle has already passed anyway
  //}
  //NVIC_ST_CTRL_R = (delay - 1);  // count down to zero
  //NVIC_ST_CURRENT_R = 0;        // any write to CVR clears it and COUNTFLAG in CSR
  //while((NVIC_ST_CTRL_R&0x00010000) == 0){};
  // method #2: repeatedly evaluate elapsed time
    volatile uint32_t elapsedTime;
  uint32_t startTime = NVIC_ST_CURRENT_R;
  do{
    elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
  }
  while(elapsedTime <= delay);
}
// Time delay using busy wait
// This assumes 80 MHz system clock
void SysTick_Wait10ms(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    SysTick_Wait(800000);  // wait 10ms (assumes 80 MHz clock)
  }
}

void SysTick_Wait1ms(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    SysTick_Wait(1400);  // wait 10ms (assumes 80 MHz clock)
  }
}
```

SysTick.h

```
// SysTick.h (Revised for 80 MHz clock)
```

```
// Runs on LM4F120/TM4C123
// Provide functions that initialize the SysTick module, wait at least a
// designated number of clock cycles, and wait approximately a multiple
// of 10 milliseconds using busy wait.  After a power-on-reset, the
// LM4F120 gets its clock from the 16 MHz precision internal oscillator,
// which can vary by +/- 1% at room temperature and +/- 3% across all
// temperature ranges.  If you are using this module, you may need more
// precise timing, so it is assumed that you are using the PLL to set
// the system clock to 80 MHz.  This matters for the function
// SysTick_Wait10ms(), which will wait longer than 10 ms if the clock is
// slower.
// Daniel Valvano
// September 11, 2013

/* This example accompanies the books
   "Embedded Systems: Introduction to ARM Cortex M Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2014
   Volume 1, Program 4.7

   "Embedded Systems: Real Time Interfacing to ARM Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
   Program 2.11, Section 2.6

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#ifndef __SYSTICK_H__
#define __SYSTICK_H__
```

```
// Initialize SysTick with busy wait running at bus clock
void SysTick_Init(void);

// Time delay using busy wait
// The delay parameter is in units of the core clock (units of 12.5 nsec for 80 MHz clock)
void SysTick_Wait(uint32_t delay);

// Time delay using busy wait
// This assumes 80 MHz system clock
void SysTick_Wait10ms(uint32_t delay);
void SysTick_Wait1ms(uint32_t delay);

#endif
```

Question 2:

The communication to the display was done by using the SPI and incrementing the display digit by digit. This is done by having the displays be dependent on the previous right display and the far right display running on its own. Now the far right will keep looping through the digital pattern and once it reaches zero it will increment the next display to the left up by 1. That left display will then increment based on the right and will increment the next left display once it reaches zero just like the previous display. The other displays will all have this function; the last left display will not increment any other display and will only display its own value.

Question 3:

The stopwatch system was implemented by attaching 3 button switches to PortE Pin 0, Pin 1, and Pin 3. From there the buttons signal a flag to GPIO_PORTE_MIS_R in the Port_Handle(), which triggers either start/pause, increment, and reset. For the start/pause, there is one interrupt that happens prior to the start/pause flag trigger, which is there to allow time for the display to load. Now in the start/pause, there is the global variable int check, which tells whether the start/pause button was used and how the displays should be shown. Now in the Time_Handle() there are four interrupts, each after each display is updated. The reason for these interrupts is to allow the display some time to show the values it is currently at. Additionally in the Time_Handle(), there are global values int x1 to x4. The values are to help keep track of which value is to be displayed during the digital pattern.

Question 4:

The SRCLK is used to push all the values for each display to the ports. This is done only when the previous display reaches a full cycle, which is getting back to 0, then the next display would be updated with a new value. This pattern continues for each display till the last display that holds the second value. Now the RCLK pushes the values out onto the display for every new value set in the shift register.

Question 5:

The modifications made to TimerA code was adding a SysTick clock reference to create a delay. Now for the periodic interrupt, A Timer_Handle function was created. There are 6 global values created for this function, one of which is used to check if the start/pause button was pressed and the others were used for the timer. For the check variable, depending on the value of check there would be two different displays for the timer. If the value of check was 1, then it would trigger the timer to pause at the current time. If the check was 0 then it would continue to count the timer up. How this worked is that it would start counting 0 to 9 on the far right display and once it reached 0 the next display incremented up. The rest of the displays would then wait for the previous one to reach 0 to increment. Now for each digit display, there is a SysTick delay to allow time for the display to show the value.

Question 6:

A student I spoke with was Brenden Dack.

[1] Microcontroller Labs, Potter, and Microcontrollers Lab, “GPIO interrupts TM4C123 Tiva Launchpad - external interrupts,” Microcontrollers Lab, https://microcontrollerslab.com/gpio-interrupts-tm4c123-tiva-launchpad-edge-level-triggered/ (accessed Oct. 28, 2024).