**Question 1:**

***main.c***

```
#include <stdint.h>
#include "LCD.h"
#include "TimerA0.h"
#include "PLL.h"
#include "ADCSWTrigger.h"
#include "SysTickInts.h"

extern int boxflag;

// step 3
   // convert ADC to Fahrenheit (make func later)
float farh(uint32_t x) {
   float c = (((x * 3.3) / 4096) * 100);   // temp in C
   float f = (c * 9 / 5 + 32) * 10;        // temp in f
   return f;
}

void main()
{
   PLL_Init(Bus80MHz);        // set system clock to 80 MHz

   LCD_Init();

   // part 1 test
      //LCD_OutString("Hello World");
      //LCD_OutUFix(0);
      //LCD_OutCmd(0xC0);
      //LCD_OutUDec(123);
      //LCD_OutUFix(12945);

      /*// A
      LCD_OutChar(0x41);
      TimerA0_Wait10ms(100);     // Wait 1s

      // B
      LCD_OutChar(0x42);
      TimerA0_Wait10ms(100);     // Wait 1s
```

```
    // C
    LCD_OutChar(0x43);
    TimerA0_Wait10ms(100);      // Wait 1s

    // Move cursor to the 2nd line
    LCD_OutCmd(0xC0);

    // 1
    LCD_OutChar(0x31);
    TimerA0_Wait10ms(100);      // Wait 1s

    // 2
    LCD_OutChar(0x32);
    TimerA0_Wait10ms(100);      // Wait 1s

    // 3
    LCD_OutChar(0x33);*/

// step 1 testing ADC
    //ADC0_InitSWTriggerSeq3_Ch8();

    //uint32_t x;
    //x = ADC0_InSeq3();

// step 2
    //LCD_OutUDec(x);
    //LCD_OutCmd(0xC0);

SysTick_Init(8000000);


//LCD_OutUFix(f);



// step 4 create a flag for mailbox
    //uint32_t ADCvalue = SysTick_Mailbox();
    //float y = farh(ADCvalue);
    //LCD_OutUFix(y);
    //boxflag = 0;
```

```
    LCD_OutCmd(0xC0);

    while(1){                    // Main loop
        if (boxflag == 1){
            uint32_t ADCvalue = SysTick_Mailbox();
            float y = farh(ADCvalue);
            LCD_OutUFix(y);
            LCD_OutCmd(0xC0);
            boxflag = 0;
        }
    }
}
```

***LCD.c***

```
/*
  size is 1*16
  if do not need to read busy, then you can tie R/W=ground
  ground = pin 1    Vss
  power  = pin 2    Vdd   +3.3V or +5V (VBUS) depending on the device
  ground = pin 3    Vlc   grounded for highest contrast
  PE0   = pin 4    RS    (1 for data, 0 for control/status)
  ground = pin 5    R/W   (1 for read, 0 for write)
  PC6   = pin 6    E     (enable)
  PA2   = pin 11   DB4   (4-bit data)
  PA3   = pin 12   DB5
  PA4   = pin 13   DB6
  PA5   = pin 14   DB7
16 characters are configured as 1 row of 16
addr  00 01 02 03 04 05 ... 0F
*/

#include <stdint.h>
#include "LCD.h"
#include "TimerA0.h"
#include "tm4c123gh6pm.h"

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
```

```
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

// Macros
#define BusFreq 80                              // assuming a 80 MHz system clock
#define T6us 6*BusFreq                          // 6us
#define T40us 40*BusFreq                   // 40us
#define T160us 160*BusFreq                 // 160us
#define T1ms 1000*BusFreq                  // 1ms
#define T1600us 1600*BusFreq               // 1.60ms
#define T5ms 5000*BusFreq                  // 5ms
#define T15ms 15000*BusFreq                     // 15ms

// Global Vars
uint8_t LCD_RS, LCD_E;                          // LCD Enable and Register Select

/**************** Private Functions ***************/

void Out_RS_E() {
   GPIO_PORTE_DATA_R = (GPIO_PORTE_DATA_R & ~0x01) | LCD_RS;
   GPIO_PORTC_DATA_R = (GPIO_PORTC_DATA_R & ~0x40) | (LCD_E<<6);
}

void SendPulse() {
        Out_RS_E();
        TimerA0_Wait(T6us);                     // wait 6us
        LCD_E = 1;                                   // E=1, R/W=0, RS=1
        Out_RS_E();
        TimerA0_Wait(T6us);                     // wait 6us
        LCD_E = 0;                                   // E=0, R/W=0, RS=1
        Out_RS_E();
}

void SendChar() {
        LCD_E = 0;
        LCD_RS = 1;                                  // E=0, R/W=0, RS=1
        SendPulse();
        TimerA0_Wait(T1600us);                  // wait 1.6ms
}
```

```
void SendCmd() {
        LCD_E = 0;
        LCD_RS = 0;                                        // E=0, R/W=0, RS=0
        SendPulse();
        TimerA0_Wait(T40us);                      // wait 40us
}

/**************** Public Functions ****************/
// Clear the LCD
// Inputs: none
// Outputs: none
void LCD_Clear() {
        LCD_OutCmd(0x01);                         // Clear Display
        LCD_OutCmd(0x80);                         // Move cursor back to 1st position
}

// Initialize LCD
// Inputs: none
// Outputs: none
void LCD_Init() {
  SYSCTL_RCGC2_R |= 0x00000015;            // 1) activate clock for Ports A, C, and E
  while((SYSCTL_PRGPIO_R&0x015) != 0x015){};  // ready?
  GPIO_PORTA_AMSEL_R &= ~0x3C;              // 3) disable analog function on PA5-2
  GPIO_PORTC_AMSEL_R &= ~0x40;              //    disable analog function on PC6
  GPIO_PORTE_AMSEL_R &= ~0x01;              //    disable analog function on PE0
  GPIO_PORTA_PCTL_R &= ~0x00FFFF00;         // 4) configure PA5-2 as GPIO
  GPIO_PORTC_PCTL_R &= ~0x0F000000;         //    configure PC6 as GPIO
  GPIO_PORTE_PCTL_R &= ~0x0000000F;         //    configure PE0 as GPIO
  GPIO_PORTA_DIR_R |= 0x3C;                 // 5) set direction register
  GPIO_PORTC_DIR_R |= 0x40;
  GPIO_PORTE_DIR_R |= 0x01;
  GPIO_PORTA_AFSEL_R &= ~0x3C;              // 6) regular port function
  GPIO_PORTC_AFSEL_R &= ~0x40;
  GPIO_PORTE_AFSEL_R &= ~0x01;
  GPIO_PORTA_DEN_R |= 0x3C;                 // 7) enable digital port
  GPIO_PORTC_DEN_R |= 0x40;
  GPIO_PORTE_DEN_R |= 0x01;
  GPIO_PORTA_DR8R_R |= 0x3C;                // enable 8 mA drive
  GPIO_PORTC_DR8R_R |= 0x40;
  GPIO_PORTE_DR8R_R |= 0x01;
```

```
   LCD_E = 0;
   LCD_RS = 1;
   Out_RS_E();
   TimerA0_Wait(T15ms);                 // Wait >15 ms after power is applied
   LCD_OutCmd(0x30);                    // command 0x30 = Wake up
   TimerA0_Wait(T5ms);                  // must wait 5ms, busy flag not available
   LCD_OutCmd(0x30);                    // command 0x30 = Wake up #2
   TimerA0_Wait(T160us);                // must wait 160us, busy flag not available
   LCD_OutCmd(0x30);                    // command 0x30 = Wake up #3
   TimerA0_Wait(T160us);                // must wait 160us, busy flag not available
   LCD_OutCmd(0x28);                    // Function set: 4-bit/2-line
   LCD_Clear();
   LCD_OutCmd(0x10);                    // Set cursor
   //LCD_OutCmd(0x0C);                  // Display ON; Cursor ON
   LCD_OutCmd(0x06);                    // Entry mode set
}

// Output a character to the LCD
// Inputs: letter is ASCII character, 0 to 0x7F
// Outputs: none
void LCD_OutChar(char letter) {
        unsigned char let_low = (0x0F&letter)<<2;
        unsigned char let_high = (letter>>2)&0x3C;
   long intstatus = StartCritical();

        GPIO_PORTA_DATA_R = (GPIO_PORTA_DATA_R & ~0x3C) | let_high;
        SendChar();
   GPIO_PORTA_DATA_R = (GPIO_PORTA_DATA_R & ~0x3C) | let_low;
        SendChar();
        EndCritical( intstatus );
        TimerA0_Wait(T1ms);                     // wait 1ms
}

// Output a command to the LCD
// Inputs: 8-bit command
// Outputs: none
void LCD_OutCmd(unsigned char command) {
        unsigned char com_low = (0x0F&command)<<2;
        unsigned char com_high = (command>>2)&0x3C;
```

```
  long intstatus = StartCritical();

  GPIO_PORTA_DATA_R = (GPIO_PORTA_DATA_R & ~0x3C) | com_high;
        SendCmd();
  GPIO_PORTA_DATA_R = (GPIO_PORTA_DATA_R & ~0x3C) | com_low;
        SendCmd();
        EndCritical( intstatus );
        TimerA0_Wait(T1ms);                          // wait 1ms
}

//------------LCD_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
// Output: none
void LCD_OutString(char *ptr) {
  int i = 0;
  while(ptr[i] != 0){
    LCD_OutChar(ptr[i]);
    TimerA0_Wait(1);
    i++;
  }
}

//-----------------------LCD_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void LCD_OutUDec(uint32_t n) {
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  uint32_t x = n % 10;
  if (n > 9) {
    LCD_OutUDec(n / 10);
  }
  LCD_OutChar(x + 0x30);

}

//-----------------------LCD_OutUHex-----------------------
```

```
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void LCD_OutUHex(uint32_t number) {
// This function uses recursion to convert the number of
//   unspecified length as an ASCII string

}

// ----------------------LCD_OutUFix---------------------
// Output characters to LCD display in fixed-point format
// unsigned decimal, resolution 0.001, range 0.000 to 9.999
// Inputs:  an unsigned 32-bit number
// Outputs: none
// E.g., 0,    then output "0.000 "
//       3,    then output "0.003 "
//       89,   then output "0.089 "
//       123,  then output "0.123 "
//       9999, then output "9.999 "
//       9999, then output "*.*** "
void LCD_OutUFix(uint32_t number) {
   if (number > 9999){
      LCD_OutString("*.*");
   } else if (number < 10){
      LCD_OutString("0.");
      LCD_OutChar(number + 0x30);
   } else if (number > 9){
      uint32_t y = number / 10;
      LCD_OutUDec(y);
      LCD_OutChar('.');
      LCD_OutUDec(number % 10);
   }
}
```

***LCD.h***

```
/*
  size is 1*16
  if do not need to read busy, then you can tie R/W=ground
```

```
  ground = pin 1    Vss
  power  = pin 2    Vdd   +3.3V or +5V (VBUS) depending on the device
  ground = pin 3    Vlc   grounded for highest contrast
  PE0   = pin 4    RS    (1 for data, 0 for control/status)
  ground = pin 5    R/W   (1 for read, 0 for write)
  PC6   = pin 6    E     (enable)
  PA2   = pin 11   DB4   (4-bit data)
  PA3   = pin 12   DB5
  PA4   = pin 13   DB6
  PA5   = pin 14   DB7
16 characters are configured as 1 row of 16
addr  00 01 02 03 04 05 ... 0F
*/

#ifndef __LCD_H__
#define __LCD_H__

// Clear the LCD
// Inputs: none
// Outputs: none
void LCD_Clear();

// Initialize LCD
// Inputs: none
// Outputs: none
void LCD_Init(void);

// Output a character to the LCD
// Inputs: letter is ASCII character, 0 to 0x7F
// Outputs: none
void LCD_OutChar(char letter);

// Output a command to the LCD
// Inputs: 8-bit command
// Outputs: none
void LCD_OutCmd(unsigned char command);

//------------LCD_OutString------------
// Output String (NULL termination)
// Input: pointer to a NULL-terminated string to be transferred
```

```
// Output: none
void LCD_OutString(char *ptr);

//----------------------LCD_OutUDec----------------------
// Output a 32-bit number in unsigned decimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void LCD_OutUDec(uint32_t n);

//----------------------LCD_OutUHex----------------------
// Output a 32-bit number in unsigned hexadecimal format
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1 to 8 digits with no space before or after
void LCD_OutUHex(uint32_t number);

// ----------------------LCD_OutUFix---------------------
// Output characters to LCD display in fixed-point format
// unsigned decimal, resolution 0.001, range 0.000 to 9.999
// Inputs:  an unsigned 32-bit number
// Outputs: none
void LCD_OutUFix(uint32_t number);

#endif
```

***TimerA0.c***

```
// TimerA0.c
// Runs on MSP432

// 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"
```

```
// 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 TimerA0_Wait( uint32_t delay ){

  if(delay <= 1){ return; } // Immediately return if requested delay less than one clock

  SYSCTL_RCGC1_R |= 0x00010000;  // 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
  TIMER0_TAILR_R = delay - 1;    // 4) Specify reload value
  TIMER0_TAPR_R = 0;             // 5) No prescale
  TIMER0_IMR_R = 0;              // 6-9) No interrupts
  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
  TIMER0_ICR_R = 1;
  while( !(TIMER0_RIS_R & 0x1) ){}
  return;
}

// Time delay using busy wait
// This assumes 80 MHz system clock
void TimerA0_Wait10ms( uint32_t delay ){
  uint32_t i;
  for( i = 0; i < delay; i++ ){
    TimerA0_Wait(800000);  // wait 10ms (assumes 80 MHz clock)
  }
  return;
}
```

## *TimerA0.h*

```
// Timer32.h
// Runs on MSP432

// 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
*/

#ifndef __TIMERA0_H__
#define __TIMERA0_H__

// 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 TimerA0_Wait( uint32_t delay );

// Time delay using busy wait
// This assumes 80 MHz system clock
void TimerA0_Wait10ms( uint32_t delay );

#endif
```

## *SysTickInts.c*

```
// SysTickInts.c
// Edited to run on Tiva-C
// Use the SysTick timer to request interrupts at a particular period.
// Daniel Valvano, Jonathan Valvano
// June 1, 2015

/* This example accompanies the books
   "Embedded Systems: Introduction to MSP432 Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
   Volume 1 Program 9.7

 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 "ADCSWTrigger.h"
#include "tm4c123gh6pm.h"

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

volatile uint32_t ADCvalue;
int boxflag = 0;

// **************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 12.5ns (assuming 80 MHz clock)
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
volatile uint32_t Counts;
uint32_t wait_per;

void SysTick_Init(uint32_t period) {
	long sr = StartCritical();
	wait_per = period;

	ADC0_InitSWTriggerSeq3_Ch8();       // initialize ADC sample PE5/A8
```

```
	Counts = 0;

	NVIC_ST_CTRL_R = 0;                    // disable SysTick during setup
	NVIC_ST_RELOAD_R = period - 1;             // maximum reload value
	NVIC_ST_CURRENT_R = 0;                  // any write to current clears it
	NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;      //
priority 2
	NVIC_ST_CTRL_R = 0x00000007;             // enable SysTick with interrupts

	EnableInterrupts();

	EndCritical(sr);
}

void SysTick_Handler() {
	ADCvalue = ADC0_InSeq3();
	boxflag = 1;
}

uint32_t SysTick_Mailbox() {
	return ADCvalue;
}
```

***SysTickInts.h***

```
// SysTickInts.h
// Edited to run on Tiva-C
// Use the SysTick timer to request interrupts at a particular period.
// Jonathan Valvano
// June 1, 2015

/* This example accompanies the books
   "Embedded Systems: Introduction to MSP432 Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
   Volume 1 Program 9.7

 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/
 */

#ifndef __SYSTICKINTS_H__
#define __SYSTICKINTS_H__

// **************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 12.5ns (assuming 80 MHz clock)
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
void SysTick_Init(uint32_t period);

uint32_t SysTick_Mailbox();

#endif
```

### ***ADCSWTrigger.c***

```
// ADCSWTrigger.c
// Runs on TM4C123
// Provide functions that initialize ADC0 SS3 to be triggered by
// software and trigger a conversion, wait for it to finish,
// and return the result.
// Daniel Valvano
// August 6, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
```

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015



```
#include <stdint.h>
#include "tm4c123gh6pm.h"

// There are many choices to make when using the ADC, and many
// different combinations of settings will all do basically the
// same thing.  For simplicity, this function makes some choices
// for you.  When calling this function, be sure that it does
// not conflict with any other software that may be running on
// the microcontroller.  Particularly, ADC0 sample sequencer 3
// is used here because it only takes one sample, and only one
// sample is absolutely needed.  Sample sequencer 3 generates a
// raw interrupt when the conversion is complete, but it is not
// promoted to a controller interrupt.  Software triggers the
// ADC0 conversion and waits for the conversion to finish.  If
// somewhat precise periodic measurements are required, the
// software trigger can occur in a periodic interrupt.  This
// approach has the advantage of being simple.  However, it does
// not guarantee real-time.
//
// A better approach would be to use a hardware timer to trigger
// the ADC0 conversion independently from software and generate
// an interrupt when the conversion is finished.  Then, the
// software can transfer the conversion result to memory and
// process it after all measurements are complete.
```

```
// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain9 (PE4)
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitSWTriggerSeq3_Ch9(void){
  SYSCTL_RCGCADC_R |= 0x0001;   // 7) activate ADC0

  SYSCTL_RCGCGPIO_R |= 0x10;      // 1) activate clock for Port E
  while((SYSCTL_PRGPIO_R&0x10) != 0x10){};
  GPIO_PORTE_DIR_R &= ~0x10;      // 2) make PE4 input
  GPIO_PORTE_AFSEL_R |= 0x10;     // 3) enable alternate function on PE4
  GPIO_PORTE_DEN_R &= ~0x10;      // 4) disable digital I/O on PE4
  GPIO_PORTE_AMSEL_R |= 0x10;     // 5) enable analog functionality on PE4

//  while((SYSCTL_PRADC_R&0x0001) != 0x0001){};    // good code, but not yet implemented
in simulator


  ADC0_PC_R &= ~0xF;              // 7) clear max sample rate field
  ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
  ADC0_SSPRI_R = 0x0123;          // 8) Sequencer 3 is highest priority
  ADC0_ACTSS_R &= ~0x0008;        // 9) disable sample sequencer 3
  ADC0_EMUX_R &= ~0xF000;         // 10) seq3 is software trigger
  ADC0_SSMUX3_R &= ~0x000F;       // 11) clear SS3 field
  ADC0_SSMUX3_R += 9;             //    set channel
  ADC0_SSCTL3_R = 0x0006;         // 12) no TS0 D0, yes IE0 END0
  ADC0_IM_R &= ~0x0008;           // 13) disable SS3 interrupts
  ADC0_ACTSS_R |= 0x0008;         // 14) enable sample sequencer 3
}


// This initialization function sets up the ADC according to the
```

```
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain8 (PE5)
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitSWTriggerSeq3_Ch8(void){
  SYSCTL_RCGCADC_R |= 0x0001;   // 7) activate ADC0

  SYSCTL_RCGCGPIO_R |= 0x10;      // 1) activate clock for Port E
  while((SYSCTL_PRGPIO_R&0x10) != 0x10){};
  GPIO_PORTE_DIR_R &= ~0x20;      // 2) make PE5 input
  GPIO_PORTE_AFSEL_R |= 0x20;     // 3) enable alternate function on PE5
  GPIO_PORTE_DEN_R &= ~0x20;      // 4) disable digital I/O on PE5
  GPIO_PORTE_AMSEL_R |= 0x20;     // 5) enable analog functionality on PE5

//  while((SYSCTL_PRADC_R&0x0001) != 0x0001){};    // good code, but not yet implemented
in simulator


  ADC0_PC_R &= ~0xF;              // 7) clear max sample rate field
  ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
  ADC0_SSPRI_R = 0x0123;          // 8) Sequencer 3 is highest priority
  ADC0_ACTSS_R &= ~0x0008;        // 9) disable sample sequencer 3
  ADC0_EMUX_R &= ~0xF000;         // 10) seq3 is software trigger
  ADC0_SSMUX3_R &= ~0x000F;       // 11) clear SS3 field
  ADC0_SSMUX3_R += 8;             //    set channel
  ADC0_SSCTL3_R = 0x0006;         // 12) no TS0 D0, yes IE0 END0
  ADC0_IM_R &= ~0x0008;           // 13) disable SS3 interrupts
  ADC0_ACTSS_R |= 0x0008;         // 14) enable sample sequencer 3
}


//------------ADC0_InSeq3------------
// Busy-wait Analog to digital conversion
// Input: none
```

```
// Output: 12-bit result of ADC conversion
uint32_t ADC0_InSeq3(void){  uint32_t result;
  ADC0_PSSI_R = 0x0008;            // 1) initiate SS3
  while((ADC0_RIS_R&0x08)==0){};   // 2) wait for conversion done
    // if you have an A0-A3 revision number, you need to add an 8 usec wait here
  result = ADC0_SSFIFO3_R&0xFFF;   // 3) read result
  ADC0_ISC_R = 0x0008;             // 4) acknowledge completion
  return result;
}
```

***ADCSWTrigger.h***

```
// ADCSWTrigger.h
// Runs on TM4C123
// Provide functions that initialize ADC0 SS3 to be triggered by
// software and trigger a conversion, wait for it to finish,
// and return the result.
// Daniel Valvano
// August 6, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015

 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/
 */

#ifndef __ADCSWTRIGGER_H__
```

```
#define __ADCSWTRIGGER_H__

// There are many choices to make when using the ADC, and many
// different combinations of settings will all do basically the
// same thing.  For simplicity, this function makes some choices
// for you.  When calling this function, be sure that it does
// not conflict with any other software that may be running on
// the microcontroller.  Particularly, ADC0 sample sequencer 3
// is used here because it only takes one sample, and only one
// sample is absolutely needed.  Sample sequencer 3 generates a
// raw interrupt when the conversion is complete, but it is not
// promoted to a controller interrupt.  Software triggers the
// ADC0 conversion and waits for the conversion to finish.  If
// somewhat precise periodic measurements are required, the
// software trigger can occur in a periodic interrupt.  This
// approach has the advantage of being simple.  However, it does
// not guarantee real-time.
//
// A better approach would be to use a hardware timer to trigger
// the ADC0 conversion independently from software and generate
// an interrupt when the conversion is finished.  Then, the
// software can transfer the conversion result to memory and
// process it after all measurements are complete.

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain9 (PE4)
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitSWTriggerSeq3_Ch9(void);

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
```

```
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain8 (PE5)
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitSWTriggerSeq3_Ch8(void);

//------------ADC0_InSeq3------------
// Busy-wait Analog to digital conversion
// Input: none
// Output: 12-bit result of ADC conversion
uint32_t ADC0_InSeq3(void);

#endif
```

**Question 2:**

The temperature sensor's full range of temperatures is -20℃ to 100℃, which is a range of -4℉ to 212℉. As for the voltages, it ranges from 4.0V to 10.0V for supply voltage and will output a voltage of around -200mV to 1000mV. Since the ADC supplies a voltage between 0 to 3.3V, it would not be able to measure the temperatures after 0℃. The relations between temperature and voltage output is 10mv/℃ * Temp ℃.

**Question 3:**

Modular design is a design principle where a system is created by various smaller pieces. It is helpful in designing a large system since you can focus on creating small parts of the system that will then ultimately work together in the end. Additionally, since you created smaller systems you can see specifically which parts of the larger system do and don't work based on the smaller system. For this lab, there were smaller parts for reading the values of the temperature sensor, printing out a number to the display, and converting the ADC number of temperature to fahrenheit. Since we focused on each part one by one we were able to better test if the system worked, such as first testing if the LCD display was working before moving to the next step.

**Question 4:**

How I would change the data flow graph from the lecture to better fit this lab would be first changing the position sensor to a temperature sensor. From there the rest of the graph would be the same with the values of each changing. For the temperature it would be from -4℉ to 212℉, the voltage from 4.0V to 10.0V, and the fixed point being from 0 to 99.9. I would change the call graph to be the SysTick_Mailbox in the main and the ADC is in the SysTick. From the main it would call the mailbox then go to the LCD driver then to the LCD hardware. The fixed point numbers are used in the data acquisition by making the system be able to convert the number provided by the ADC once converted to fahrenheit to be displayed onto the LCD. The variable integer being the fahrenheit number, the range being from 0 to 99.9 and the fixed constant being to the nearest tenth.

**Question 5:**

The mailbox will not work properly when the ACD is sampled at a higher frequency. The reason is that if the sampling goes too fast then there is a chance that an interrupt could be missed. A FIFO would work better at a high frequency since it would do each interrupt one by one. How I would change my code to support a FIFO queue would be by having an interrupt flag at each point in the sampling.

**Question 6:**

[1] Mechatronics Engineer, "Interfacing LM35 Temperature Sensor with TM4C123 Microcontroller," YouTube, https://www.youtube.com/watch?v=O6KhYW6NjvQ (accessed Nov. 18, 2024).