California State University

EGCP 450

Lab 5

Professor

Zaid Frayeh

Code :

Main.c :

```
#include <stdint.h>
#include "LCD.h"
#include "Timer32.h"
#include "ADC14.h"
#include "SysTickInts.h"

void main()
{
    LCD_Init();

    ADC0_InitSWTriggerCh0();

    SysTick_Init(2344);
    LCD_OutString("ZAID LAB 5");
    while(1){

        if(mailbox == 1){

    uint32_t ADC = SysTick_Mailbox();

    ADC = ADC * 0.12208;

    LCD_OutCmd(0xC0);
    LCD_OutUFix(ADC);
    LCD_OutString(" CM");
    mailbox = 0;

        }
        Timer32_Wait10ms(30);

    }

}
```

LCD.h :

```
#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
```

LCD.c :

```
#include <stdint.h>
#include "LCD.h"
#include "Timer32.h"
#include "msp.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 48                              // assuming a 48 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 OutPort6() {
      P6OUT = (LCD_RS<<4) | (LCD_E<<5);
}

void SendPulse() {
      OutPort6();
      Timer32_Wait(T6us);                       // wait 6us
      LCD_E = 1;                                      // E=1, R/W=0, RS=1
      OutPort6();
      Timer32_Wait(T6us);                       // wait 6us
      LCD_E = 0;                                      // E=0, R/W=0, RS=1
      OutPort6();
}

void SendChar() {
      LCD_E = 0;
      LCD_RS = 1;                                     // E=0, R/W=0, RS=1
      SendPulse();
      Timer32_Wait(T1600us);                    // wait 1.6ms
}

void SendCmd() {
      LCD_E = 0;
      LCD_RS = 0;                                     // E=0, R/W=0, RS=0
      SendPulse();
      Timer32_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() {
      P4SEL0 &= ~0xF0;
      P4SEL1 &= ~0xF0;                          // configure upper nibble of P4 as GPIO
      P4DIR |= 0xF0;                                  // make upper nibble of P4 out

      P6SEL0 &= ~0x30;
      P6SEL1 &= ~0x30;                          // configure P6.4 and P6.5 as GPIO
      P6DIR |= 0x30;                                  // make P6.4 and P6.5 out

      LCD_E = 0;
      LCD_RS = 0;                                     // E=0, R/W=0, RS=0
      OutPort6();
```

```
      LCD_OutCmd(0x30);                     // command 0x30 = Wake up
      Timer32_Wait(T5ms);                   // must wait 5ms, busy flag not
available
      LCD_OutCmd(0x30);                     // command 0x30 = Wake up #2
      Timer32_Wait(T160us);                 // must wait 160us, busy flag not
available
      LCD_OutCmd(0x30);                     // command 0x30 = Wake up #3
      Timer32_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(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)<<4;
      unsigned char let_high = 0xF0&letter;
    long intstatus = StartCritical();

      P4OUT = let_high;
      SendChar();
      P4OUT = let_low;
      SendChar();
      EndCritical( intstatus );
      Timer32_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)<<4;
      unsigned char com_high = 0xF0&command;
    long intstatus = StartCritical();

      P4OUT = com_high;
      SendCmd();
      P4OUT = com_low;
      SendCmd();
      EndCritical( intstatus );
      Timer32_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 x;
    for(x = 0 ; x<=18 ; x++)
    {
        if(ptr[x] == 0){
            return 0;
                    }
```

```
        LCD_OutChar(ptr[x]);
    }

                                }

//-----------------------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
    static char s[16];

        static int32_t i;

        static uint32_t count ;


           if(n/10){
               count++;
               LCD_OutUDec(n/10);

           }


           s[i++] = abs(n) % 10 + '0';

           s[i] = '\0';


           if(count == i - 1){

           LCD_OutString(s);
           i = 0;
           count = 0;
           }



    }
void hextodec(uint32_t hexa) {

     if(hexa == 0 ) LCD_OutChar(0x30);
     else if(hexa == 1 )LCD_OutChar(0x31);
     else if(hexa == 2 )LCD_OutChar(0x32);
     else if(hexa == 3 )LCD_OutChar(0x33);
     else if(hexa == 4 )LCD_OutChar(0x34);
     else if(hexa == 5 )LCD_OutChar(0x35);
     else if(hexa == 6 )LCD_OutChar(0x36);
     else if(hexa == 7 )LCD_OutChar(0x37);
     else if(hexa == 8 )LCD_OutChar(0x38);
     else if(hexa == 9 )LCD_OutChar(0x39);
```

```
}


// ----------------------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 num) {

           static char array[16];
           static int32_t i;
           static uint32_t count ;

           if(count == 0)
       {
           if(num < 10){

               array[i++] = '0';
               array[i++] = '.';
               array[i++] = '0';
               array[i++] = '0';

                        }
           if(num >= 10 && num < 99){
               array[i++] = '0';
               array[i++] = '.';
               array[i++] = '0';
                                    }
           if(num >= 100 && num < 999){
               array[i++] = '0';
               array[i++] = '.';
                                       }
     }


              if(num/10){
                  count++;
                  LCD_OutUFix(num/10);

                          }

              array[i++] = abs(num) % 10 + '0';

              if(count == 3){

                  if(i == 1){
                      array[i++] = '.';
                            }
                            }

              array[i] = '\0';

              if(num < 10){
```

```
            if(count == i - 5){

                      LCD_OutString(array);
                      i = 0;
                      count = 0;
                              }

                    }
        if(num >= 10 && num < 99){
            if(count == i - 4){

                      LCD_OutString(array);
                      i = 0;
                      count = 0;
                              }
                                 }
        if(num >= 100 && num < 999){
            if(count == i - 3){

                      LCD_OutString(array);
                      i = 0;
                      count = 0;
                               }
                                   }

        if(num >= 1000){

            if(count + 2 == i){

                LCD_OutString(array);
                i = 0;
                count = 0;
                        }
                        }


    }
```

Timer32.h :

```
#ifndef __TIMER32_H__
#define __TIMER32_H__

// Time delay using busy wait
// The delay parameter is in units of the core clock (units of 333 nsec for 3 MHz
clock)
void Timer32_Wait( uint32_t delay );

// Time delay using busy wait
// This assumes 3 MHz system clock
void Timer32_Wait10ms( uint32_t delay );

#endif
```

Timer32.c :

```c
#include <stdint.h>
#include "msp432p401r.h"

// Time delay using busy wait
// The delay parameter is in units of the core clock (units of 333 nsec for 3 MHz clock)
void Timer32_Wait( uint32_t delay ){

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

  TIMER32_1->CONTROL = (TIMER32_1->CONTROL & ~(0x000000EF)) | 0x000000C3;
    // Timer32 in periodic mode, no interrupts, no prescale, 32-bit, and one-shot
  TIMER32_1->LOAD = delay - 1; // Will count down to 0
  while( TIMER32_1->VALUE ){} // Wait until timer expires (value equals 0)
  // Or, clear interrupt and wait for raw interrupt flag to be set
  //TIMER32_1->INTCLR = 0;
  //while( !(TIMER32_1->RIS & 0x1) ){}
  return;
}

// Time delay using busy wait
// This assumes 3 MHz system clock
void Timer32_Wait10ms( uint32_t delay ){
  uint32_t i;
  for( i = 0; i < delay; i++ ){
    Timer32_Wait(30000);  // wait 10ms (assumes 3 MHz clock)
  }
  return;
}
```

SysTickints.h :

```c
#ifndef __SYSTICKINTS_H__
#define __SYSTICKINTS_H__

// **************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 333ns (assuming 3 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();

int mailbox;

#endif
```

SysTickints.c :

```c
#include <stdint.h>
#include "ADC14.h"
#include "msp432p401r.h"
#include "SysTickInts.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;

// **************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 333ns (assuming 3 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_InitSWTriggerCh0();                // initialize ADC sample P5.5/A0

      Counts = 0;

      SysTick->CTRL = 0;                   // disable SysTick during setup
      SysTick->LOAD = period - 1;          // maximum reload value
      SysTick->VAL = 0;                    // any write to current clears it
      SCB->SHP[3] = (SCB->SHP[3]&0x00FFFFFF)|0x40000000;   // priority 2
      SysTick->CTRL = 0x00000007;          // enable SysTick with interrupts

      EnableInterrupts();

      EndCritical(sr);
}

void SysTick_Handler() {

    mailbox = 1;

      ADCvalue = ADC_In();
}

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

Question 2:
Part 1 :

sampling an ADC at a higher frequency than necessary may not be beneficial, as it can increase the complexity of the system and may introduce noise and other issues. However, if the ADC is being used to sample a signal with high-frequency components that need to be accurately captured, then a higher sampling frequency may be required. But in our case with the hardware and software we have so going to a higher sampling rate, the system might not be fast enough to display the data

PART 2:

if you sample an ADC at a higher frequency, using a FIFO (First In First Out) queue can be beneficial for storing data, we can store it much faster Since we have a first in first out and our array can be as big as they system requires it to be. But displaying the data might be a little bit slower since we need higher requirements to be able to process the speed of the FIFO

to implement a FIFO:

we need to declare and initialize a FIFO as well as an array that would have the size of the samples that we need
we enqueue the ADC data to the FIFO array from the tail and then dequeue and process it from the head and display it, it would be in a while loop so that it keeps repeating this process