

Lab Report

|  |  |
| --- | --- |
| Name: Alexis Steven Garcia | Date: December 3, 2018 |
| Course: EGCP-450 | Lab #: 7 |

Grading Criteria:

|  |  |  |
| --- | --- | --- |
| **Section** | **Earned Points** | **Possible Points** |
| Program Code: |  | 34 |
| Part 1 Demo: |  | 33 |
| Part 2 Demo: |  | 33 |
| Total: | 0 | 100 |

**PLEASE UPLOAD YOUR REPORT IN TITANIUM. NO PAPER REPORTS.**

Professor Comments:

# Program Code

Copy your code here. Please provide comments in your code. This will help me analyze your code and remove any ambiguity. **Provide your code as text, not as a screenshot/image**.

## Part 1: LCD Driver

**Main.c:**

**#include** <stdint.h>

**#include** "LCD.h"

**#include** "SysTick.h"

**void** **main**()

{

**uint32\_t** wait\_1s = 3000000;

**uint32\_t** wait\_halfs = 1500000;

**LCD\_Init**();

// A

**LCD\_OutChar**('A');

**SysTick\_Wait**(wait\_1s); // Wait 1s

// B

**LCD\_OutChar**('B');

**SysTick\_Wait**(wait\_1s); // Wait 1s

// C

**LCD\_OutChar**('C');

**SysTick\_Wait**(wait\_1s); // Wait 1s

// Move cursor to the 2nd line

**LCD\_OutCmd**(0xC0);

// 1

**LCD\_OutChar**('1');

**SysTick\_Wait**(wait\_1s); // Wait 1s

// 2

**LCD\_OutChar**('2');

**SysTick\_Wait**(wait\_1s); // Wait 1s

// 3

**LCD\_OutChar**('3');

**SysTick\_Wait**(wait\_1s); // Wait 1s

//Clear LCD

// LCD\_OutCmd(0x01);

**LCD\_Clear**();

/\*

\* TEST LCD

\* 0123456789

\*/

**LCD\_OutString**("TEST LCD");

**LCD\_OutCmd**(0xC0); // Move cursor to the 2nd line

**for**(**int** i = 0; i < 10; i++){

**LCD\_OutChar**(i+'0'); //1-9

**SysTick\_Wait**(wait\_halfs); // Wait 1s

}

**LCD\_Clear**();

**LCD\_OutUFix**(0);

**SysTick\_Wait**(wait\_halfs);

**LCD\_Clear**();

**LCD\_OutUFix**(3333);

**SysTick\_Wait**(wait\_halfs);

**LCD\_Clear**();

**LCD\_OutUFix**(6666);

**SysTick\_Wait**(wait\_halfs);

**LCD\_Clear**();

**LCD\_OutUFix**(9999);

**SysTick\_Wait**(wait\_halfs);

**LCD\_Clear**();

**LCD\_OutUFix**(10000);

**SysTick\_Wait**(wait\_halfs);

**while**(1); // Main loop

}

**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 depending on the device

ground = pin 3 Vlc grounded for highest contrast

P9.6 = pin 4 RS (1 for data, 0 for control/status)

ground = pin 5 R/W (1 for read, 0 for write)

P9.7 = pin 6 E (enable)

P7.0 = pin 7 DB0 (8-bit data)

P7.1 = pin 8 DB1

P7.2 = pin 9 DB2

P7.3 = pin 10 DB3

P7.4 = pin 11 DB4

P7.5 = pin 12 DB5

P7.6 = pin 13 DB6

P7.7 = pin 14 DB7

16 characters are configured as 1 row of 16

addr 00 01 02 03 04 05 ... 0F

\*/

// 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** \*pt);

//-----------------------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);

**void** **LCD\_asterick**(**uint32\_t** status);

**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 depending on the device

ground = pin 3 Vlc grounded for highest contrast

P6.4 = pin 4 RS (1 for data, 0 for control/status)

ground = pin 5 R/W (1 for read, 0 for write)

P6.5 = pin 6 E (enable)

P4.4 = pin 11 DB4 (4-bit data)

P4.5 = pin 12 DB5

P4.6 = pin 13 DB6

P4.7 = pin 14 DB7

16 characters are configured as 1 row of 16

addr 00 01 02 03 04 05 ... 0F

\*/

**#include** <stdint.h>

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** "LCD.h"

**#include** "SysTick.h"

**#include** "msp.h"

// Marcros

**#define** **BusFreq** 3 // assuming a 3 MHz bus 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

**uint32\_t** input, previous;

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

**void** **OutPort6**() {

P6OUT = (LCD\_RS<<4) | (LCD\_E<<5);

}

**void** **SendPulse**() {

**OutPort6**();

**SysTick\_Wait**(T6us); // wait 6us

LCD\_E = 1; // E=1, R/W=0, RS=1

**OutPort6**();

**SysTick\_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**();

**SysTick\_Wait**(T1600us); // wait 1.6ms

}

**void** **SendCmd**() {

LCD\_E = 0;

LCD\_RS = 0; // E=0, R/W=0, RS=0

**SendPulse**();

**SysTick\_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

//Clock\_Init48MHz(); // set system clock to 48 MHz

**SysTick\_Init**(); // Volume 1 Program 4.7, Volume 2 Program 2.12

LCD\_E = 0;

LCD\_RS = 0; // E=0, R/W=0, RS=0

**OutPort6**();

**LCD\_OutCmd**(0x30); // command 0x30 = Wake up

**SysTick\_Wait**(T5ms); // must wait 5ms, busy flag not available

**LCD\_OutCmd**(0x30); // command 0x30 = Wake up #2

**SysTick\_Wait**(T160us); // must wait 160us, busy flag not available

**LCD\_OutCmd**(0x30); // command 0x30 = Wake up #3

**SysTick\_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

input = 0;

previous = 1;

}

// 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;

P4OUT = let\_high;

**SendChar**();

P4OUT = let\_low;

**SendChar**();

**SysTick\_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;

P4OUT = com\_high;

**SendCmd**();

P4OUT = com\_low;

**SendCmd**();

**SysTick\_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** \*pt) {

**while**(\*pt){

**LCD\_OutChar**(\*pt);

pt++;

}

}

//-----------------------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

**if**(n >= 10){

**LCD\_OutUDec**(n/10);

n = n%10;

}

**LCD\_OutChar**(n+'0'); /\* n is between 0 and 9 \*/

}

//--------------------------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

**if**(number >= 0x10){

**LCD\_OutUHex**(number/0x10);

**LCD\_OutUHex**(number%0x10);

}

**else**{

**if**(number < 0xA){

**LCD\_OutChar**(number+'0');

}

**else**{

**LCD\_OutChar**((number-0x0A)+'A');

}

}

}

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

input = number;

**if**(number < 10000){

**uint32\_t** thousands, hundreds, tens, ones;

thousands = number/1000;

**LCD\_OutChar**(thousands+'0');

**LCD\_OutChar**('.');

hundreds = (number%1000)/100;

**LCD\_OutChar**(hundreds+'0');

tens = (number%100)/10;

**LCD\_OutChar**(tens+'0');

ones = number%10;

**LCD\_OutChar**(ones+'0');

previous = input;

}

**else**{

**LCD\_asterick**(1);

}

}

**void** **LCD\_asterick**(**uint32\_t** status){

/\*

\* \*.\*\*\*\*

\* DONE :)

\*/

**LCD\_OutString**("\*.\*\*\*");

**SysTick\_Wait**(3000000);

**LCD\_OutCmd**(0xC0); // Move cursor to the 2nd line

**if**(status == 1){

**LCD\_OutString**("DONE :)");

}

}

## Part 2: Position Measurement System

**Main.c:**

**#include** <stdint.h>

**#include** "SysTickInts.h"

**#include** "LCD.h"

**#include** "msp432p401r.h"

**#include** "ADC14.h"

**void** **main**() {

**uint32\_t** wait\_1s = 3000000;

**uint32\_t** value;

**ADC0\_InitSWTriggerCh0**();

**LCD\_Init**();

**LCD\_OutString**("EGCP-450 Lab 8");

**while**(1) {

value = **SysTick\_Mailbox**();

**LCD\_OutCmd**(0xC0);

**LCD\_OutUFix**(value);

**SysTick\_Wait**(wait\_1s);

**LCD\_OutString**(" cm");

}

}

**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 depending on the device

ground = pin 3 Vlc grounded for highest contrast

P9.6 = pin 4 RS (1 for data, 0 for control/status)

ground = pin 5 R/W (1 for read, 0 for write)

P9.7 = pin 6 E (enable)

P7.0 = pin 7 DB0 (8-bit data)

P7.1 = pin 8 DB1

P7.2 = pin 9 DB2

P7.3 = pin 10 DB3

P7.4 = pin 11 DB4

P7.5 = pin 12 DB5

P7.6 = pin 13 DB6

P7.7 = pin 14 DB7

16 characters are configured as 1 row of 16

addr 00 01 02 03 04 05 ... 0F

\*/

// 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** \*pt);

//-----------------------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);

**void** **LCD\_asterick**(**uint32\_t** status);

**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 depending on the device

ground = pin 3 Vlc grounded for highest contrast

P6.4 = pin 4 RS (1 for data, 0 for control/status)

ground = pin 5 R/W (1 for read, 0 for write)

P6.5 = pin 6 E (enable)

P4.4 = pin 11 DB4 (4-bit data)

P4.5 = pin 12 DB5

P4.6 = pin 13 DB6

P4.7 = 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** "SysTickInts.h"

**#include** "msp.h"

**#include** "ADC14.h"

// Marcros

**#define** **BusFreq** 3 // assuming a 3 MHz bus 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

**uint32\_t** input, previous;

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

**void** **OutPort6**() {

P6OUT = (LCD\_RS<<4) | (LCD\_E<<5);

}

**void** **SendPulse**() {

**OutPort6**();

**SysTick\_Wait**(T6us); // wait 6us

LCD\_E = 1; // E=1, R/W=0, RS=1

**OutPort6**();

**SysTick\_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**();

**SysTick\_Wait**(T1600us); // wait 1.6ms

}

**void** **SendCmd**() {

LCD\_E = 0;

LCD\_RS = 0; // E=0, R/W=0, RS=0

**SendPulse**();

**SysTick\_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

//Clock\_Init48MHz(); // set system clock to 48 MHz

**SysTick\_Init**(**ADC\_In**()); // Volume 1 Program 4.7, Volume 2 Program 2.12

LCD\_E = 0;

LCD\_RS = 0; // E=0, R/W=0, RS=0

**OutPort6**();

**LCD\_OutCmd**(0x30); // command 0x30 = Wake up

**SysTick\_Wait**(T5ms); // must wait 5ms, busy flag not available

**LCD\_OutCmd**(0x30); // command 0x30 = Wake up #2

**SysTick\_Wait**(T160us); // must wait 160us, busy flag not available

**LCD\_OutCmd**(0x30); // command 0x30 = Wake up #3

**SysTick\_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

input = 0;

previous = 1;

}

// 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;

P4OUT = let\_high;

**SendChar**();

P4OUT = let\_low;

**SendChar**();

**SysTick\_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;

P4OUT = com\_high;

**SendCmd**();

P4OUT = com\_low;

**SendCmd**();

**SysTick\_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** \*pt) {

**while**(\*pt){

**LCD\_OutChar**(\*pt);

pt++;

}

}

//-----------------------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

**if**(n >= 10){

**LCD\_OutUDec**(n/10);

n = n%10;

}

**LCD\_OutChar**(n+'0'); /\* n is between 0 and 9 \*/

}

//--------------------------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

**if**(number >= 0x10){

**LCD\_OutUHex**(number/0x10);

**LCD\_OutUHex**(number%0x10);

}

**else**{

**if**(number < 0xA){

**LCD\_OutChar**(number+'0');

}

**else**{

**LCD\_OutChar**((number-0x0A)+'A');

}

}

}

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

input = number;

**if**(input != previous){

**uint32\_t** ten\_thous, thousands, hundreds, tens, ones;

ten\_thous = number/10000;

// LCD\_OutChar(ten\_thous+'0');

thousands = (number%10000)/1000;

**LCD\_OutChar**(thousands+'0');

**LCD\_OutChar**('.');

hundreds = (number%1000)/100;

**LCD\_OutChar**(hundreds+'0');

tens = (number%100)/10;

**LCD\_OutChar**(tens+'0');

ones = number%10;

**LCD\_OutChar**(ones+'0');

previous = input;

}

**else**{

// LCD\_asterick(0);

}

}

**void** **LCD\_asterick**(**uint32\_t** status){

/\*

\* \*.\*\*\*\*

\* DONE :)

\*/

**LCD\_OutString**("\*.\*\*\*");

**LCD\_OutCmd**(0xC0); // Move cursor to the 2nd line

**if**(status == 1){

**LCD\_OutString**("DONE :)");

}

}

**SysTick.c:**

**#include** <stdint.h>

**#include** "ADC14.h"

**#include** "msp432p401r.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;

**volatile** **uint32\_t** ADCflag;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*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**() {

ADCvalue = **ADC\_In**();

}

**uint32\_t** **SysTick\_Mailbox**() {

**return** (10000 \* ADCvalue) / 16383;

}

// Time delay using busy wait.

// The delay parameter is in units of the core clock. (units of 333 nsec for 3 MHz clock)

**void** **SysTick\_Wait**(**uint32\_t** delay) {

**long** sr = **StartCritical**();

// 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

}

SysTick->LOAD = (delay - 1); // count down to zero

SysTick->VAL = 0; // any write to CVR clears it and COUNTFLAG in CSR

**while**((SysTick->CTRL&0x00010000) == 0);

SysTick->LOAD = wait\_per - 1; // maximum reload value

**EndCritical**(sr);

}

// Time delay using busy wait.

// This assumes 3 MHz system clock.

**void** **SysTick\_Wait10ms**(**uint32\_t** delay) {

**long** sr = **StartCritical**();

**uint32\_t** i;

**for**(i=0; i<delay; i++){

**SysTick\_Wait**(30000); // wait 10ms (assumes 3 MHz clock)

}

**EndCritical**(sr);

}