**10/16/2018**

**Task 00: Execute provided code**

**------------------------------------------------------------------------------------**

Modifications highlighted in Yellow

**Task 01:**

Youtube Link: https://youtu.be/Lbu4uktovUg

**Modified Code:**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_ssi.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/ssi.h"

**#include** "driverlib/gpio.h"

**#include** "driverlib/sysctl.h"

**#include** "utils/uartstdio.h"

**#include** "utils/uartstdio.c"

**#include** "driverlib/uart.h"

**#include** "driverlib/pin\_map.h"

**#include** "driverlib/adc.h"

**#include** "driverlib/timer.h"

**#include** "driverlib/interrupt.h"

**#define** NUM\_SSI\_DATA 3

// Global variables

uint32\_t ui32ADC0Value[3];

uint32\_t ui32Period;

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**volatile** **char** buffer[3];

**volatile** uint32\_t pui32DataTx[NUM\_SSI\_DATA];

**volatile** uint32\_t pui32DataRx[NUM\_SSI\_DATA];

**volatile** uint32\_t ui32Index;

// Prototype function to convert int to char for UARTCharPut function

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base);

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

{

// Setting clock to 50Mz

**SysCtlClockSet**(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC |SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

// The SSI0 peripheral and port A must be enabled for use.

// Enable the SSI0 peripheral

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_SSI0);

// The SSI0 peripheral is on Port A and pins 2,3,4 and 5.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);

// This function/s configures the pin muxing on port A pins 2,3,4 and 5

**GPIOPinConfigure**(GPIO\_PA2\_SSI0CLK);

**GPIOPinConfigure**(GPIO\_PA3\_SSI0FSS);

**GPIOPinConfigure**(GPIO\_PA4\_SSI0RX);

**GPIOPinConfigure**(GPIO\_PA5\_SSI0TX);

**GPIOPinTypeSSI**(GPIO\_PORTA\_BASE, GPIO\_PIN\_5 | GPIO\_PIN\_4 | GPIO\_PIN\_3 |GPIO\_PIN\_2);

**GPIOPinWrite**(GPIO\_PORTA\_BASE,GPIO\_PIN\_4,GPIO\_PIN\_4);

// Configure and enable the SSI port for SPI master mode.

**SSIClockSourceSet**(SSI0\_BASE,SSI\_CLOCK\_SYSTEM);

**SSIConfigSetExpClk**(SSI0\_BASE, **SysCtlClockGet**(),SSI\_FRF\_MOTO\_MODE\_0,SSI\_MODE\_MASTER, 1000000, 8);

// ADC Configuration

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

**ADCSequenceConfigure**(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0); // Changed to sequencer #2

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

// Timer 1 configuration

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); // Timer 1 enabled

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC); // Timer in periodic mode

ui32Period = **SysCtlClockGet**()/2; // Period of 0.5s 2Hz

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1); // setting timer to expire at 0.5 seconds

**IntEnable**(INT\_TIMER1A); // Interrupt for Timer1A enabled

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

// Initial message

**UARTprintf**("\nSSI ->\n");

**UARTprintf**(" Mode: SPI\n");

**UARTprintf**(" Data: 8-bit\n\n");

// Enable interrupts

**IntMasterEnable**();

// Enable SSI module

**SSIEnable**(SSI0\_BASE);

**ADCSequenceEnable**(ADC0\_BASE, 2);

**TimerEnable**(TIMER1\_BASE, TIMER\_A);

// Enable UART0 so that we can configure the clock.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);

// Use the internal 16MHz oscillator as the UART clock source.

**UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);

// Select the alternate (UART) function for these pins.

**GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// Initialize the UART for console I/O.

**UARTStdioConfig**(0, 115200, 16000000);

}

// To convert integer to character. This function is not in stdlib.h

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base)

{

// check that the base if valid

**if** (base < 2 || base > 36) { \*result = '\0';}

**char**\* ptr = result, \*ptr1 = result, tmp\_char;

**int** tmp\_value;

**do** {

tmp\_value = value;

value /= base;

\*ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35 + (tmp\_value - value \* base)];

} **while** ( value );

// Apply negative sign

**if** (tmp\_value < 0) \*ptr++ = '-';

\*ptr-- = '\0';

**while**(ptr1 < ptr) {

tmp\_char = \*ptr;

\*ptr--= \*ptr1;

\*ptr1++ = tmp\_char;

}

}

**void** **Timer1IntHandler**(**void**)

{

// Clear the timer interrupt

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

**ADCIntClear**(ADC0\_BASE, 2);

**ADCProcessorTrigger**(ADC0\_BASE, 2);

**while**(!**ADCIntStatus**(ADC0\_BASE, 2, false))

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

itoa(ui32TempValueF, &buffer, 10); // converting integer to char

**UARTprintf**("\nTemperature Sent:\n ");

**for**(ui32Index = 0; ui32Index < NUM\_SSI\_DATA; ui32Index++)

{

pui32DataTx[ui32Index] = buffer[ui32Index];

**if**( ui32Index == 2)

{

pui32DataTx[ui32Index] = 'F';

}

// Display the data that SSI is transferring.

**UARTprintf**("'%c' ", pui32DataTx[ui32Index]);

**SSIDataPut**(SSI0\_BASE, pui32DataTx[ui32Index]);

}

// Wait until SSIO is done transferring all data in FIFO

**while**(**SSIBusy**(SSI0\_BASE))

{

}

// Display indication that SSI receives data

**UARTprintf**("\nTemperature Received:\n ");

**for**(ui32Index = 0; ui32Index < NUM\_SSI\_DATA; ui32Index++)

{

**SSIDataGet**(SSI0\_BASE, &pui32DataRx[ui32Index]);

// Since we are using 8-bit data, mask off the MSB.

pui32DataRx[ui32Index] &= 0x00FF;

// Display the data that SSI0 received.

**UARTprintf**("'%c' ", pui32DataRx[ui32Index]);

}

}

**------------------------------------------------------------------------------------**

**Task 02:**

Youtube Link: <https://youtu.be/zRXKPTrMDBI>

**Modified Code:**

**#include** <stdint.h>

**#include** <stdbool.h>

**#include** "inc/hw\_memmap.h"

**#include** "inc/hw\_ssi.h"

**#include** "inc/hw\_types.h"

**#include** "driverlib/ssi.h"

**#include** "driverlib/sysctl.h"

**#include** "driverlib/pin\_map.h"

**#include** "driverlib/adc.h"

**#include** "driverlib/timer.h"

**#include** "driverlib/interrupt.h"

**#include** "Nokia5110.h"

**#define** NUM\_SSI\_DATA 4

// Global variables

uint32\_t ui32ADC0Value[4];

uint32\_t ui32Period;

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

**volatile** **char** Fbuffer[4];

**volatile** **char** Cbuffer[4];

**volatile** uint32\_t pui32DataTx[NUM\_SSI\_DATA];

**volatile** uint32\_t pui32DataRx[NUM\_SSI\_DATA];

**volatile** uint32\_t ui32Index;

**unsigned** **char** menu\_elements[12][25];

// Prototype function to convert int to char for UARTCharPut function

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base);

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

{

startSSI0(); // Initializing SSI0 module, definition in Nokia5510.c file

initialize\_screen(BACKLIGHT\_ON,SSI0);

// Setting clock to 50Mz

**SysCtlClockSet**(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC |SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

// ADC Configuration

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);

**ADCSequenceConfigure**(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0); // Changed to sequencer #2

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

// Timer 1 configuration

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); // Timer 1 enabled

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC); // Timer in periodic mode

ui32Period = **SysCtlClockGet**(); // Period of 1s frequency 1Hz

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1); // setting timer to expire at 0.5 seconds

**IntEnable**(INT\_TIMER1A); // Interrupt for Timer1A enabled

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

// Enable interrupts

**IntMasterEnable**();

**ADCSequenceEnable**(ADC0\_BASE, 2);

**TimerEnable**(TIMER1\_BASE, TIMER\_A);

}

// To convert integer to character. This function is not in stdlib.h

**void** **itoa**(**long** **unsigned** **int** value, **char**\* result, **int** base)

{

// check that the base if valid

**if** (base < 2 || base > 36) { \*result = '\0';}

**char**\* ptr = result, \*ptr1 = result, tmp\_char;

**int** tmp\_value;

**do** {

tmp\_value = value;

value /= base;

\*ptr++ = "zyxwvutsrqponmlkjihgfedcba9876543210123456789abcdefghijklmnopqrstuvwxyz" [35 + (tmp\_value - value \* base)];

} **while** ( value );

// Apply negative sign

**if** (tmp\_value < 0) \*ptr++ = '-';

\*ptr-- = '\0';

**while**(ptr1 < ptr) {

tmp\_char = \*ptr;

\*ptr--= \*ptr1;

\*ptr1++ = tmp\_char;

}

}

**void** **Timer1IntHandler**(**void**)

{

// Clear the timer interrupt

**TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);

**ADCIntClear**(ADC0\_BASE, 2);

**ADCProcessorTrigger**(ADC0\_BASE, 2);

**while**(!**ADCIntStatus**(ADC0\_BASE, 2, false))

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);

// Calculating temperatures

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;

ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;

ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;

// Converting integer value to character for buffer

itoa(ui32TempValueF, &Fbuffer, 10); // converting integer to char

itoa(ui32TempValueC, &Cbuffer, 10);

clear\_screen(SSI0);

// Displaying temperature values on LCD

screen\_write("Temperature",ALIGN\_CENTRE\_TOP,SSI0);

screen\_write(Fbuffer,ALIGN\_CENTRE\_CENTRE,SSI0);

screen\_write("F ",ALIGN\_RIGHT\_CENTRE,SSI0);

screen\_write(Cbuffer,ALIGN\_CENTRE\_BOTTOM,SSI0);

screen\_write("C ",ALIGN\_RIGHT\_BOTTOM,SSI0);

}

**Code used from Nokia5110.c to initialize SSI0 module**

**void** **startSSI0**()//This one works

{

**volatile** **unsigned** **long** delay;

SYSCTL\_RCGC2\_R |= 0x00000001; // activate clock for Port A

SYSCTL\_RCGCSSI\_R|=SYSCTL\_RCGCSSI\_R0; // activate clock for SSI0

delay = SYSCTL\_RCGC2\_R; // allow time for clock to stabilize

GPIO\_PORTA\_DIR\_R |= 0xD0; // make PA4,6,7 out

GPIO\_PORTA\_AFSEL\_R |= 0x2C; // enable alt funct on PA2,3,5

GPIO\_PORTA\_AFSEL\_R &= ~0xC0; // disable alt funct on PA6,7

GPIO\_PORTA\_DEN\_R |= 0xFC; // enable digital I/O on PA2,3,4,5,6,7

// configure PA2,3,5 as SSI

GPIO\_PORTA\_PCTL\_R = (GPIO\_PORTA\_PCTL\_R&0xFF0F00FF)+0x00202200;

// configure PA6,7 as GPIO

GPIO\_PORTA\_PCTL\_R = (GPIO\_PORTA\_PCTL\_R&0x00FFFFFF)+0x00000000;

GPIO\_PORTA\_AMSEL\_R &= ~0xFC; // disable analog functionality on PA2,3,4,5,6,7

SSI0\_CR1\_R&=~SSI\_CR1\_SSE; // 3)Disable SSI while configuring it

SSI0\_CR1\_R&=~SSI\_CR1\_MS; // 4) Set as Master

SSI0\_CC\_R|=SSI\_CC\_CS\_M; // 5) Configure clock source

SSI0\_CC\_R|=SSI\_CC\_CS\_SYSPLL; // 5) Configure clock source

SSI0\_CC\_R|=SSI\_CPSR\_CPSDVSR\_M;// 6) Configure prescale divisor

SSI0\_CPSR\_R = (SSI0\_CPSR\_R&~SSI\_CPSR\_CPSDVSR\_M)+2; // must be even number

SSI0\_CR0\_R |=0x7000;

SSI0\_CR0\_R &= ~(SSI\_CR0\_SPH | SSI\_CR0\_SPO);

//SSI0\_CPSR\_R = (SSI0\_CPSR\_R&~SSI\_CR0\_SCR\_M)+7; // must be even number

SSI0\_CR0\_R = (SSI0\_CR0\_R&~SSI\_CR0\_FRF\_M)+SSI\_CR0\_FRF\_MOTO;

// DSS = 8-bit data

SSI0\_CR0\_R = (SSI0\_CR0\_R&~SSI\_CR0\_DSS\_M)+SSI\_CR0\_DSS\_8;

SSI0\_CR1\_R|=SSI\_CR1\_SSE; // 3)Enable SSI

RESET = RESET\_LOW; // reset the LCD to a known state

**for**(delay=0; delay<10; delay=delay+1);// delay minimum 100 ns

RESET = RESET\_HIGH; // negative logic

}