**Date Submitted: 11/01/2018**

**Assignment Youtube Playlist:** **https://www.youtube.com/playlist?list=PL4oTyvRrubXf28k1HpLJkt31GBqTGDdho**

# Task 01: Modify the supplied code to transmit and receive the Internal Temperature and verify the results.

For Task 01, I have modified the included code to measure the internal temperature using our ADC, calculated it into two 8-bit values for Celsius and Fahrenheit, and sent the values through SPI loopback to be displayed in UART. To do so, I modified our clock settings as the SPI\_Master default settings were causing a fault during ADC initialization. To connect the loopback, we connected ports A4 and A5, our SPI transmit and receive pins, to have our board communicate with itself.

**Youtube Link:** **https://youtu.be/WYpkyyCXMdw**

**Original Code (added comments):**

**#include** <stdbool.h>

**#include** <stdint.h>

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

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

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

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

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

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

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

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

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

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

**#include** "inc/tm4c123gh6pm.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//! \addtogroup ssi\_examples\_list

//! <h1>SPI Master (spi\_master)</h1>

//!

//! This example shows how to configure the SSI0 as SPI Master. The code will

//! send three characters on the master Tx then polls the receive FIFO until

//! 3 characters are received on the master Rx.

//!

//! This example uses the following peripherals and I/O signals. You must

//! review these and change as needed for your own board:

//! - SSI0 peripheral

//! - GPIO Port A peripheral (for SSI0 pins)

//! - SSI0Clk - PA2

//! - SSI0Fss - PA3

//! - SSI0Rx - PA4

//! - SSI0Tx - PA5

//!

//! The following UART signals are configured only for displaying console

//! messages for this example. These are not required for operation of SSI0.

//! - UART0 peripheral

//! - GPIO Port A peripheral (for UART0 pins)

//! - UART0RX - PA0

//! - UART0TX - PA1

//!

//! This example uses the following interrupt handlers. To use this example

//! in your own application you must add these interrupt handlers to your

//! vector table.

//! - None.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Number of bytes to send and receive.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#define** NUM\_SSI\_DATA 2

**#define** TARGET\_IS\_BLIZZARD\_RB1

//Declaring global temperature variables. To be made internal to function in future iterations.

uint32\_t ui32ADC0Value[8];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint8\_t ui8TempValueFSend;

**volatile** uint8\_t ui8TempValueFReceive;

**volatile** uint8\_t ui8TempValueCSend;

**volatile** uint8\_t ui8TempValueCReceive;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// This function sets up UART0 to be used for a console to display information

// as the example is running.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void**

**InitConsole**(**void**)

{

//

// Enable GPIO port A which is used for UART0 pins.

// **TODO**: change this to whichever GPIO port you are using.

//

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

//

// Configure the pin muxing for UART0 functions on port A0 and A1.

// This step is not necessary if your part does not support pin muxing.

// **TODO**: change this to select the port/pin you are using.

//

**GPIOPinConfigure**(GPIO\_PA0\_U0RX);

**GPIOPinConfigure**(GPIO\_PA1\_U0TX);

//

// 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.

// **TODO**: change this to select the port/pin you are using.

//

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

//

// Initialize the UART for console I/O.

//

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

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Configure SSI0 in master Freescale (SPI) mode. This example will send out

// 3 bytes of data, then wait for 3 bytes of data to come in. This will all be

// done using the polling method.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**int**

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

{

**#if** defined(TARGET\_IS\_TM4C129\_RA0) || \

defined(TARGET\_IS\_TM4C129\_RA1) || \

defined(TARGET\_IS\_TM4C129\_RA2)

uint32\_t ui32SysClock;

**#endif**

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

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

uint32\_t ui32Index;

//

// Set the clocking to run directly from the external crystal/oscillator.

// **TODO**: The SYSCTL\_XTAL\_ value must be changed to match the value of the

// crystal on your board.

//

**#if** defined(TARGET\_IS\_TM4C129\_RA0) || \

defined(TARGET\_IS\_TM4C129\_RA1) || \

defined(TARGET\_IS\_TM4C129\_RA2)

ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |

SYSCTL\_OSC\_MAIN |

SYSCTL\_USE\_OSC), 25000000);

**#else**

//Modifying clock frequency for our device.

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ); //40MHz clock total.

// SysCtlClockSet(SYSCTL\_SYSDIV\_1 | SYSCTL\_USE\_OSC | SYSCTL\_OSC\_MAIN |

// SYSCTL\_XTAL\_16MHZ);

**#endif**

//

// Set up the serial console to use for displaying messages. This is

// just for this example program and is not needed for SSI operation.

//

InitConsole();

//

// Display the setup on the console.

//

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

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

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

//

// The SSI0 peripheral must be enabled for use.

//

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

//

// For this example SSI0 is used with PortA[5:2]. The actual port and pins

// used may be different on your part, consult the data sheet for more

// information. GPIO port A needs to be enabled so these pins can be used.

// **TODO**: change this to whichever GPIO port you are using.

//

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

//

// Configure the pin muxing for SSI0 functions on port A2, A3, A4, and A5.

// This step is not necessary if your part does not support pin muxing.

// **TODO**: change this to select the port/pin you are using.

//

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

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

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

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

//

// Configure the GPIO settings for the SSI pins. This function also gives

// control of these pins to the SSI hardware. Consult the data sheet to

// see which functions are allocated per pin.

// The pins are assigned as follows:

// PA5 - SSI0Tx

// PA4 - SSI0Rx

// PA3 - SSI0Fss

// PA2 - SSI0CLK

// **TODO**: change this to select the port/pin you are using.

//

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

GPIO\_PIN\_2);

//

// Configure and enable the SSI port for SPI master mode. Use SSI0,

// system clock supply, idle clock level low and active low clock in

// freescale SPI mode, master mode, 1MHz SSI frequency, and 8-bit data.

// For SPI mode, you can set the polarity of the SSI clock when the SSI

// unit is idle. You can also configure what clock edge you want to

// capture data on. Please reference the datasheet for more information on

// the different SPI modes.

//

**#if** defined(TARGET\_IS\_TM4C129\_RA0) || \

defined(TARGET\_IS\_TM4C129\_RA1) || \

defined(TARGET\_IS\_TM4C129\_RA2)

SSIConfigSetExpClk(SSI0\_BASE, ui32SysClock, SSI\_FRF\_MOTO\_MODE\_0,

SSI\_MODE\_MASTER, 1000000, 8);

**#else**

**SSIConfigSetExpClk**(SSI0\_BASE, **SysCtlClockGet**(), SSI\_FRF\_MOTO\_MODE\_0,

SSI\_MODE\_MASTER, 1000000, 8);

**#endif**

//

// Enable the SSI0 module.

//

**SSIEnable**(SSI0\_BASE);

//Enable ADC

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //Enable clock to ADC0.

//ADCHardwareOversampleConfigure(ADC0\_BASE, 32);

//Configure hardware averaging of ADC0, 32 samples.

**ADCSequenceConfigure**(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

//ADC0, Sample Sequencer 0, triggered by processor, highest priority (0).

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

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 1, ADC\_CTL\_TS);

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

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 3, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 4, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 5, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 6, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 7, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 0);

//Configure the eight steps of our sequencer. All using ADC0 and sequencer 0, measuring the internal temperature

//sensor, then triggering the interrupt flag when finished and signal the last conversion on sequencer 0.

//Enable ADC after all eight steps are set.

**IntMasterEnable**();

**while**(1){

//

// Read any residual data from the SSI port. This makes sure the receive

// FIFOs are empty, so we don't read any unwanted junk. This is done here

// because the SPI SSI mode is full-duplex, which allows you to send and

// receive at the same time. The SSIDataGetNonBlocking function returns

// "true" when data was returned, and "false" when no data was returned.

// The "non-blocking" function checks if there is any data in the receive

// FIFO and does not "hang" if there isn't.

//

**while**(**SSIDataGetNonBlocking**(SSI0\_BASE, &pui32DataRx[0]))

{

}

//Sense temperature.

**ADCProcessorTrigger**(ADC0\_BASE, 0); //Trigger ADC conversion with the processor.

**while**(!**ADCIntStatus**(ADC0\_BASE, 0, false)) //Wait for ADC conversion to complete.

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 0, ui32ADC0Value); //After conversion is complete, pull the data into the array.

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + ui32ADC0Value[4] + ui32ADC0Value[5] + ui32ADC0Value[6] + ui32ADC0Value[7] + 4) /8;

//Average the eight array temperatures, adding +4 to compensate for rounding due to integer math.

ui8TempValueCSend = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10; //Calculate degrees C.

ui8TempValueFSend = ((ui8TempValueCSend \* 9) + 160) / 5; //Calculate degrees F.

//

// Initialize the data to send.

//

pui32DataTx[0] = ui8TempValueFSend;

pui32DataTx[1] = ui8TempValueCSend;

//

// Display indication that the SSI is transmitting data.

//

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

//

// Send 2 bytes of data.

//

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

{

//

// Display the data that SSI is transferring.

//

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

//

// Send the data using the "blocking" put function. This function

// will wait until there is room in the send FIFO before returning.

// This allows you to assure that all the data you send makes it into

// the send FIFO.

//

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

}

//

// Wait until SSI0 is done transferring all the data in the transmit FIFO.

//

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

{

}

//

// Display indication that the SSI is receiving data.

//

**UARTprintf**("\nReceived:\n ");

//

// Receive 1 byte of data.

//

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

{

//

// Receive the data using the "blocking" Get function. This function

// will wait until there is data in the receive FIFO before returning.

//

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

//

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

//

pui32DataRx[ui32Index] &= 0x00FF;

//

// Save data received into the array.

//

ui8TempValueFReceive = pui32DataRx[0];

ui8TempValueCReceive = pui32DataRx[1];

}

//Display the temperatures received through the SPI loopback.

**UARTprintf**("Current Device Temperature in Fahrenheit: %i\n", ui8TempValueFReceive);

**UARTprintf**("Current Device Temperature in Celsius: %i\n", ui8TempValueCReceive);

}

//

// Return no errors

//

**return**(0);

}

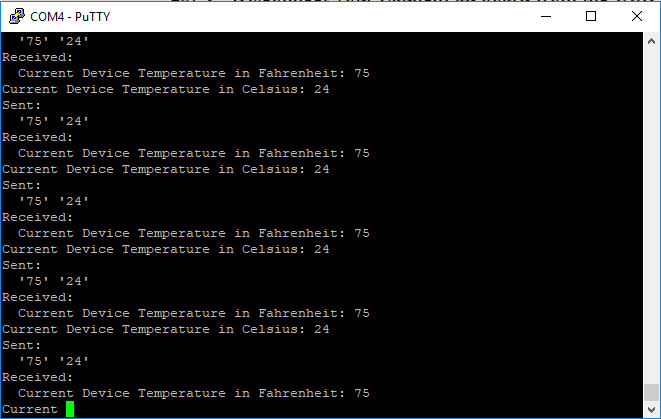


Figure 1 - PuTTY UART output, showing sent and received data.

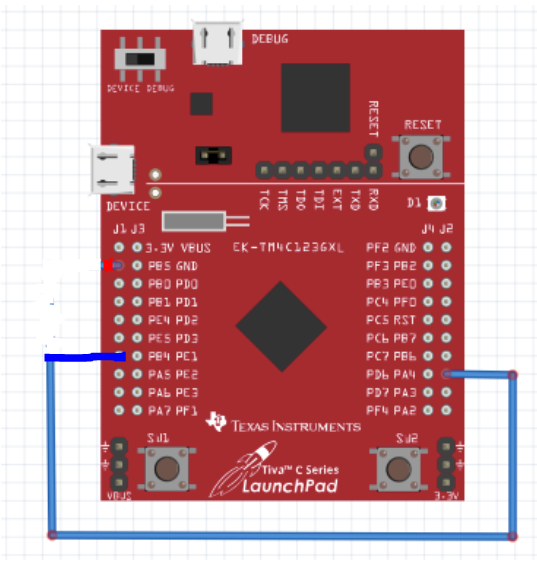


Figure - Loopback wiring diagram

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

# Task 02: Display the z-axis results in Nokia5110 GLCD. If task is not working, display the Lab 5 – Temperature on the LCD as: “Temperature: 72.92 F, 20.34 F”. Update every sec. using the timer.

**Youtube Link:** **https://youtu.be/0fykmgprYAk**

For Task 02 I modified the provided main.c file used for interacting with the Nokia 5110 GLCD. The ADC is taking temperature sensor readings once per second on a timer. The resulting reading is stored in two float variables for Fahrenheit and Celsius. I then used code to convert these floats to 5-large CHAR arrays (credit to aBUGSworstnightmare of 43oh.com for the “ftoa” function). The temperature is then sent to the LCD to be displayed, once per second.

**#include** <stdint.h>

**#include** <stdbool.h>

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

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

**#include** "inc/tm4c123gh6pm.h"

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

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

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

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

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

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

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

**#include** "Nokia5110.h"

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

//Include all required files and drivers for the lab.

**#define** TARGET\_IS\_BLIZZARD\_RB1

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

//Enable calling the Peripheral Driver Library from ROM, saving flash space.

**#ifdef** DEBUG //Record file name and line number of any errors from a library API with incorrect parameters.

void\_error\_(**char** \*pcFilename, uint32\_t ui32Line)

{

}

**#endif**

uint32\_t ui32ADC0Value[8];

**volatile** uint32\_t ui32TempAvg;

**volatile** **float** TempValueC;

**volatile** **float** TempValueF;

**char** CharF[5];

**char** CharC[5];

//Function to convert a float character to a character array.

//Credit to "aBUGSworstnightmare"

**void** **ftoa**(**float** f,**char** \*buf)

{

**int** pos=0,ix,dp,num;

**if** (f<0)

{

buf[pos++]='-';

f = -f;

}

dp=0;

**while** (f>=10.0)

{

f=f/10.0;

dp++;

}

**for** (ix=1;ix<8;ix++)

{

num = (**int**)f;

f=f-num;

**if** (num>9)

buf[pos++]='#';

**else**

buf[pos++]='0'+num;

**if** (dp==0) buf[pos++]='.';

f=f\*10.0;

dp--;

}

}

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

{

uint32\_t ui32TimerDelay; //Set timer delay for use in ADC triggering.

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ); //40MHz clock total.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0); //Enable clock to ADC0.

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); //Enable Timer 1 peripheral.

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC); //Configure timer 1 to be periodic.

uint32\_t ui32Clock = **SysCtlClockGet**(); //Retrieve system clock and store in variable.

ui32TimerDelay = (ui32Clock); //Trigger at 1Hz, 1 second period.

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32TimerDelay -1); //Load on-period into timer 1.

**IntEnable**(INT\_TIMER1A); //Enable vector associated with Timer 1A.

**TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT); //Enable interrupt from Timer 1A.

**IntMasterEnable**();

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

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

//Enable GPIO Port F and set Pin 2 as output.

**ADCHardwareOversampleConfigure**(ADC0\_BASE, 32); //Configure hardware averaging of ADC0, 32 samples.

**ADCSequenceConfigure**(ADC0\_BASE, 0, ADC\_TRIGGER\_PROCESSOR, 0);

//ADC0, Sample Sequencer 0, triggered by processor, highest priority (0).

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

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 1, ADC\_CTL\_TS);

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

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 3, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 4, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 5, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 6, ADC\_CTL\_TS);

**ADCSequenceStepConfigure**(ADC0\_BASE, 0, 7, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

**ADCSequenceEnable**(ADC0\_BASE, 0);

//Configure the eight steps of our sequencer. All using ADC0 and sequencer 0, measuring the internal temperature

//sensor, then triggering the interrupt flag when finished and signal the last conversion on sequencer 0.

//Enable ADC after all eight steps are set.

//Enable lazy stacking and the FPU using the ROM functions.

**FPULazyStackingEnable**();

**FPUEnable**();

startSSI0(); //Enable SSI0 and allow some time to process.

initialize\_screen(BACKLIGHT\_ON,SSI0); //Initialize LCD screen connected with SSI0.

clear\_screen(SSI0); //Clear previous screen contents.

screen\_write("ECG\n603!\nTemp\nSensor",ALIGN\_CENTRE\_CENTRE,SSI0); //Write enabling message.

**SysCtlDelay**(ui32Clock \* 2); //Pause for 2 seconds.

**TimerEnable**(TIMER1\_BASE, TIMER\_A); //Enable and start timer 1A.

//Endless loop to wait in while timer is running.

**while**(1)

{

}

}

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

{

// Clear the timer interrupt

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

**ADCIntClear**(ADC0\_BASE, 0); //Clear ADC interrupt flag.

**ADCProcessorTrigger**(ADC0\_BASE, 0); //Trigger ADC conversion with the processor.

**while**(!**ADCIntStatus**(ADC0\_BASE, 0, false)) //Wait for ADC conversion to complete.

{

}

**ADCSequenceDataGet**(ADC0\_BASE, 0, ui32ADC0Value); //After conversion is complete, pull the data into the array.

ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + ui32ADC0Value[4] + ui32ADC0Value[5] + ui32ADC0Value[6] + ui32ADC0Value[7] + 4) /8;

TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10; //Calculate degrees C.

TempValueF = ((TempValueC \* 9) + 160) / 5; //Calculate degrees F.

//Convert float values into character arrays.

ftoa(TempValueF, CharF);

ftoa(TempValueC, CharC);

clear\_screen(SSI0); //Clear previous screen contents.

screen\_write("Temperature:\n",ALIGN\_LEFT\_CENTRE,SSI0); //Write temperature string, then character translations of characters.

char\_write(CharF[0],SSI0);

char\_write(CharF[1],SSI0);

char\_write(CharF[2],SSI0);

char\_write(CharF[3],SSI0);

char\_write(CharF[4],SSI0);

char\_write('F',SSI0);

char\_write(',',SSI0);

char\_write(' ',SSI0);

char\_write(CharC[0],SSI0);

char\_write(CharC[1],SSI0);

char\_write(CharC[2],SSI0);

char\_write(CharC[3],SSI0);

char\_write(CharC[4],SSI0);

char\_write('C',SSI0);

}

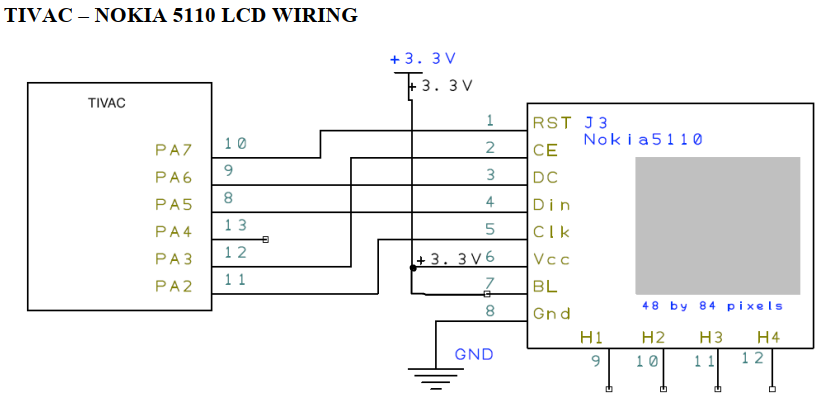


Figure - Nokia 5110 Wiring Diagram

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