CPE403 – Advanced Embedded Systems

# Design Assignment 1

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Xianjie Cao

Email: [caox2@unlv.nevada.edu](mailto:caox2@unlv.nevada.edu)

Github Repository link (root): <https://github.com/c1029324620/Flat-White>

Youtube Playlist link (root): <https://www.youtube.com/playlist?list=PLY90fbcjLcrnosJGw9U__oC1jxRqwp8w8>

**Follow the submission guideline to be awarded points for this Assignment.**

Submit the following for all Assignments:

1. In the document, for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only.
2. Create a private Github repository with a random name (no CPE/403, Lastname, Firstname). Place all labs under the root folder TIVAC, sub-folder named Assignment1, with one document and one video link file for each lab, place modified c files named as asng\_taskxx.c.
3. If multiple c files or other libraries are used, create a folder asng1\_t01 and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup\_ccs.c and other include files, c) text file with youtube video links (see template).
5. Submit the doc file in canvas before the due date. The root folder of the github assignment directory should have the documentation and the text file with youtube video links.
6. Organize your youtube videos as playlist under the name “cpe403”. The playlist should have the video sequence arranged as submission or due dates.
7. Only submit pdf documents. Do not forget to upload this document in the github repository and in the canvas submission portal.
8. Code for Tasks. for each task submit the modified or included code (from the base code) with highlights and justifications of the modifications. Also include the comments. If no base code is provided, submit the base code for the first task only. Use separate page for each task.

**Task1:**

**#include** <stdint.h>

**#include** <stdbool.h>

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

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

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

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

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

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

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

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

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

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

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

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

**#include** <string.h>

**#ifdef** DEBUG

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

{

}

**#endif**

// Globals

uint32\_t ui32Period;

**char** buffer[4];

uint8\_t status = 0x0E;

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Timer 1 ISR

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

{

// Clear the timer interrupt

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

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

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

**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;

**UARTprintf**("C %3d\t",ui32TempValueC );

**UARTprintf**("F %3d\t",ui32TempValueF );

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

}

**void** **GPIOF4IntHandler**(**void**)

{

//Toggle the RGB LED.

status = status ^ 14;

//clear GPIO interrupt

**GPIOIntClear**(GPIO\_PORTF\_BASE, GPIO\_INT\_PIN\_4);

**GPIOPinWrite**(GPIO\_PORTF\_BASE,GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, status);

}

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

// Configure Clock

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

// Configure peripherals

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

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

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF); //enable GPIOF port.

//Configure GPIO port F

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

**GPIOPinTypeGPIOInput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_4);

**GPIOPadConfigSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_4, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

**GPIOIntRegister**(GPIO\_PORTF\_BASE, GPIOF4IntHandler); //register hanndler function for port F

**GPIOIntTypeSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_4, GPIO\_FALLING\_EDGE); //PF4 for falling edge trigger.

**GPIOIntEnable**(GPIO\_PORTF\_BASE, GPIO\_INT\_PIN\_4); //enable interrupt for PF4.

// Configure ADC

**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);

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

////////////////////

// Configure Timer 1 module

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

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

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1);

**IntEnable**(INT\_TIMER1A);

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

// Configure pins for UART

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

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

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

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

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

// Enable interrupts

**IntMasterEnable**();

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

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

// Initial message to terminal display

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

**while** (1) // Wait forever

{

}

}

Task 2:

**#include** <stdint.h>

**#include** <stdbool.h>

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

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

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

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

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

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

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

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

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

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

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

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

**#include** <string.h>

**#ifdef** DEBUG

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

{

}

**#endif**

// Globals

uint32\_t ui32Period;

**char** buffer[4];

uint8\_t status = 0x0E;

uint32\_t ui32ADC0Value[4];

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Timer 1 ISR

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

{

// Clear the timer interrupt

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

//ADC keep running.

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

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

**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;

}

**void** **display\_tem\_c**()

{

**UARTprintf**("Temperature in C: %3d\n", ui32TempValueC);

}

**void** **display\_tem\_f**()

{

**UARTprintf**("Temperature in F: %3d\n", ui32TempValueF);

}

**void** **red\_on**()

{

//turn RED LED on

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, GPIO\_PIN\_1);

}

**void** **blue\_on**()

{

//turn BLUE LED on

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

}

**void** **green\_on**()

{

//Turn GREEN LED on

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, GPIO\_PIN\_3);

}

**void** **red\_off**()

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1, 0x0);

}

**void** **blue\_off**()

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0x0);

}

**void** **green\_off**()

{

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3, 0x0);

}

**void** **get\_status**()

{

uint8\_t red, green, blue;

red = **GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

blue = **GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

green = **GPIOPinRead**(GPIO\_PORTF\_BASE, GPIO\_PIN\_3);

**if**(red == 0x0)

**UARTprintf**("Red LED is OFF\n");

**else**

**UARTprintf**("Red LED is ON\n");

**if**(green == 0x0)

**UARTprintf**("Green LED is OFF\n");

**else**

**UARTprintf**("Green LED is ON\n");

**if**(blue == 0x0)

**UARTprintf**("Blue LED is OFF\n");

**else**

**UARTprintf**("Blue LED is ON\n");

}

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

{

//40MHz

**SysCtlClockSet**(SYSCTL\_SYSDIV\_5 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0); //Enabling UART

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA); //Enabling GPIO port A

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

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF); //enable GPIOF port.

// Configure ADC

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

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

**GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);

**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);

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

// Configure Timer 1 module

**TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);

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

**TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1);

**IntEnable**(INT\_TIMER1A);

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

// Configure pins for UART

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

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

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

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

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

// Enable interrupts

**IntMasterEnable**();

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

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

**UARTprintf**("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Enter the command\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

**UARTprintf**("\*\*R|r => Red LED B|b => Blue LED G|g => Green LED\*\*\n");

**UARTprintf**("\*\*\*T => Temp in C | t => Temp in F | S => RGB Status\*\*\*\n");

**while**(1)

{

**switch**(**UARTCharGet**(UART0\_BASE))

{

**case** 'T':

display\_tem\_c();

**break**;

**case** 't':

display\_tem\_f();

**break**;

**case** 'R':

red\_on();

**break**;

**case** 'r':

red\_off();

**break**;

**case** 'B':

blue\_on();

**break**;

**case** 'b':

blue\_off();

**break**;

**case** 'G':

green\_on();

**break**;

**case** 'g':

green\_off();

**break**;

**case** 'S':

get\_status();

**break**;

**default**:

**break**;

}

}

}

**Task 3:**

**#include** <stdbool.h>

**#include** <stdint.h>

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**#define** ADC\_SAMPLE\_BUF\_SIZE 64

**enum** BUFFERSTATUS

{ *EMPTY*,

*FILLING*,

*FULL*

};

**#pragma** DATA\_ALIGN(ucControlTable, 1024)

uint8\_t ucControlTable[1024];

**static** uint16\_t ADC\_Out1[ADC\_SAMPLE\_BUF\_SIZE];

**static** uint16\_t ADC\_Out2[ADC\_SAMPLE\_BUF\_SIZE];

**static** **enum** BUFFERSTATUS BufferStatus[2];

**volatile** uint32\_t ui32Temp;

**void** **ConfigureUART**(**void**);

**void** **init\_ADC**(**void**);

**void** **init\_TIMER**(**void**);

**void** **init\_DMA**(**void**);

**static** uint32\_t g\_ui32DMAErrCount = 0u;

**static** uint32\_t g\_ui32SysTickCount;

**void** **uDMAErrorHandler**(**void**)

{

uint32\_t ui32Status;

ui32Status = MAP\_uDMAErrorStatusGet();

**if**(ui32Status)

{

MAP\_uDMAErrorStatusClear();

g\_ui32DMAErrCount++;

}

}

// Not used in this example, but used to debug to make sure timer interrupts happen

**void** **Timer0AIntHandler**(**void**)

{

//

// Clear the timer interrupt flag.

//

**TimerIntClear**(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

}

**void** **SysTickIntHandler**(**void**)

{

// Update our system tick counter.

g\_ui32SysTickCount++;

}

**void** **ADCseq0Handler**()

{

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

**if** ((**uDMAChannelModeGet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT) == UDMA\_MODE\_STOP)

&& (BufferStatus[0] == *FILLING*))

{

BufferStatus[0] = *FULL*;

BufferStatus[1] = *FILLING*;

}

**else** **if** ((**uDMAChannelModeGet**(UDMA\_CHANNEL\_ADC0 | UDMA\_ALT\_SELECT) == UDMA\_MODE\_STOP)

&& (BufferStatus[1] == *FILLING*))

{

BufferStatus[0] = *FILLING*;

BufferStatus[1] = *FULL*;

}

}

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

{

uint32\_t i, average1, average2, samples\_taken;

// Set the system clock to run at 80MHz from the PLL.

**SysCtlClockSet**(SYSCTL\_SYSDIV\_2\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

**SysCtlDelay**(20u);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0); //Enable the clock to TIMER0

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0); //Enable the clock to ADC module

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UDMA); //Enable the clock to uDMA

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE); //Enables the clock to PORT E

MAP\_SysCtlDelay(30u);

MAP\_SysTickPeriodSet(**SysCtlClockGet**() / 100000u); //Sets the period of the SysTic counter to 10us

MAP\_SysTickIntEnable();

MAP\_SysTickEnable();

BufferStatus[0] = *FILLING*;

BufferStatus[1] = *EMPTY*;

samples\_taken = 0u;

ConfigureUART();

**UARTprintf**("ADC->uDMA->UART\n");

**UARTprintf**("\tTempC\tTempF\tTotal Samples\n");

init\_DMA();

init\_ADC();

init\_TIMER();

MAP\_IntMasterEnable();

MAP\_TimerEnable(TIMER0\_BASE, TIMER\_A); // Start everything

**while**(1)

{

**if**(BufferStatus[0u] == *FULL*)

{

// Do something with data in ADC\_OUT1

average1 = 0u;

**for**(i =0u; i < ADC\_SAMPLE\_BUF\_SIZE; i++)

{

average1 += ADC\_Out1[i];

ADC\_Out1[i] = 0u;

}

BufferStatus[0u] = *EMPTY*;

// Enable for another uDMA block transfer

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0), &ADC\_Out1, ADC\_SAMPLE\_BUF\_SIZE);

**uDMAChannelEnable**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT); // Enables DMA channel so it can perform transfers

samples\_taken += ADC\_SAMPLE\_BUF\_SIZE;

average1 = (average1 + (ADC\_SAMPLE\_BUF\_SIZE / 2u)) / ADC\_SAMPLE\_BUF\_SIZE;

average1 = (1475 - ((2475 \* average1)) / 4096)/10; //temp in C

}

**if**(BufferStatus[1u] == *FULL*)

{

// Do something with data in ADC\_OUT2

average2 = 0u;

**for**(i =0u; i < ADC\_SAMPLE\_BUF\_SIZE; i++)

{

average2 += ADC\_Out2[i];

ADC\_Out2[i] = 0u;

}

BufferStatus[1u] = *EMPTY*;

// Enable for another uDMA block transfer

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_ALT\_SELECT, UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0), &ADC\_Out2, ADC\_SAMPLE\_BUF\_SIZE);

**uDMAChannelEnable**(UDMA\_CHANNEL\_ADC0 | UDMA\_ALT\_SELECT);

samples\_taken += ADC\_SAMPLE\_BUF\_SIZE;

average2 = (average2 + (ADC\_SAMPLE\_BUF\_SIZE / 2u)) / ADC\_SAMPLE\_BUF\_SIZE;

average2 = ((average1 \* 9) + 160) / 5; //temp in F

**UARTprintf**("\t%d\t%d\t%d\r", average1,average2,samples\_taken);

}

}

}

**void** **init\_TIMER**()

{

MAP\_TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_SPLIT\_PAIR | TIMER\_CFG\_A\_PERIODIC);

// Set sample frequency to 16KHz (every 62.5uS)

MAP\_TimerLoadSet(TIMER0\_BASE, TIMER\_A, MAP\_SysCtlClockGet()/16000 -1); //**TODO**: Timer Load Value is set here

MAP\_TimerControlTrigger(TIMER0\_BASE, TIMER\_A, **true**);

MAP\_TimerControlStall(TIMER0\_BASE, TIMER\_A, **true**); //Assist in debug by stalling timer at breakpoints

}

**void** **init\_ADC**()

{

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

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

**SysCtlDelay**(80u);

// Use ADC0 sequence 0 to sample channel 0 once for each timer period

**ADCClockConfigSet**(ADC0\_BASE, ADC\_CLOCK\_SRC\_PIOSC | ADC\_CLOCK\_RATE\_HALF, 1);

**SysCtlDelay**(10); // Time for the clock configuration to set

**IntDisable**(INT\_ADC0SS0);

**ADCIntDisable**(ADC0\_BASE, 0u);

**ADCSequenceDisable**(ADC0\_BASE,0u);

// With sequence disabled, it is now safe to load the new configuration parameters

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

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

**ADCSequenceEnable**(ADC0\_BASE,0u); //Once configuration is set, re-enable the sequencer

**ADCIntClear**(ADC0\_BASE,0u);

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

**IntEnable**(INT\_ADC0SS0);

}

**void** **init\_DMA**()

{

**uDMAEnable**(); // Enables uDMA

**uDMAControlBaseSet**(ucControlTable);

**uDMAChannelAttributeDisable**(UDMA\_CHANNEL\_ADC0, UDMA\_ATTR\_ALTSELECT | UDMA\_ATTR\_HIGH\_PRIORITY | UDMA\_ATTR\_REQMASK);

**uDMAChannelAttributeEnable**(UDMA\_CHANNEL\_ADC0, UDMA\_ATTR\_USEBURST);

// Only allow burst transfers

**uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_SIZE\_16 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_16 | UDMA\_ARB\_1);

**uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_ALT\_SELECT, UDMA\_SIZE\_16 | UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_16 | UDMA\_ARB\_1);

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0), &ADC\_Out1, ADC\_SAMPLE\_BUF\_SIZE);

**uDMAChannelTransferSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_ALT\_SELECT, UDMA\_MODE\_PINGPONG, (**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0), &ADC\_Out2, ADC\_SAMPLE\_BUF\_SIZE);

**uDMAChannelEnable**(UDMA\_CHANNEL\_ADC0); // Enables DMA channel so it can perform transfers

}

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

//

// Configure the UART and its pins. This must be called before UARTprintf().

//

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

**void**

**ConfigureUART**(**void**)

{

//

// Enable the GPIO Peripheral used by the UART.

//

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//

// Enable UART0

//

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

//

// Configure GPIO Pins for UART mode.

//

MAP\_GPIOPinConfigure(GPIO\_PA0\_U0RX);

MAP\_GPIOPinConfigure(GPIO\_PA1\_U0TX);

MAP\_GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//

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

//

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

//

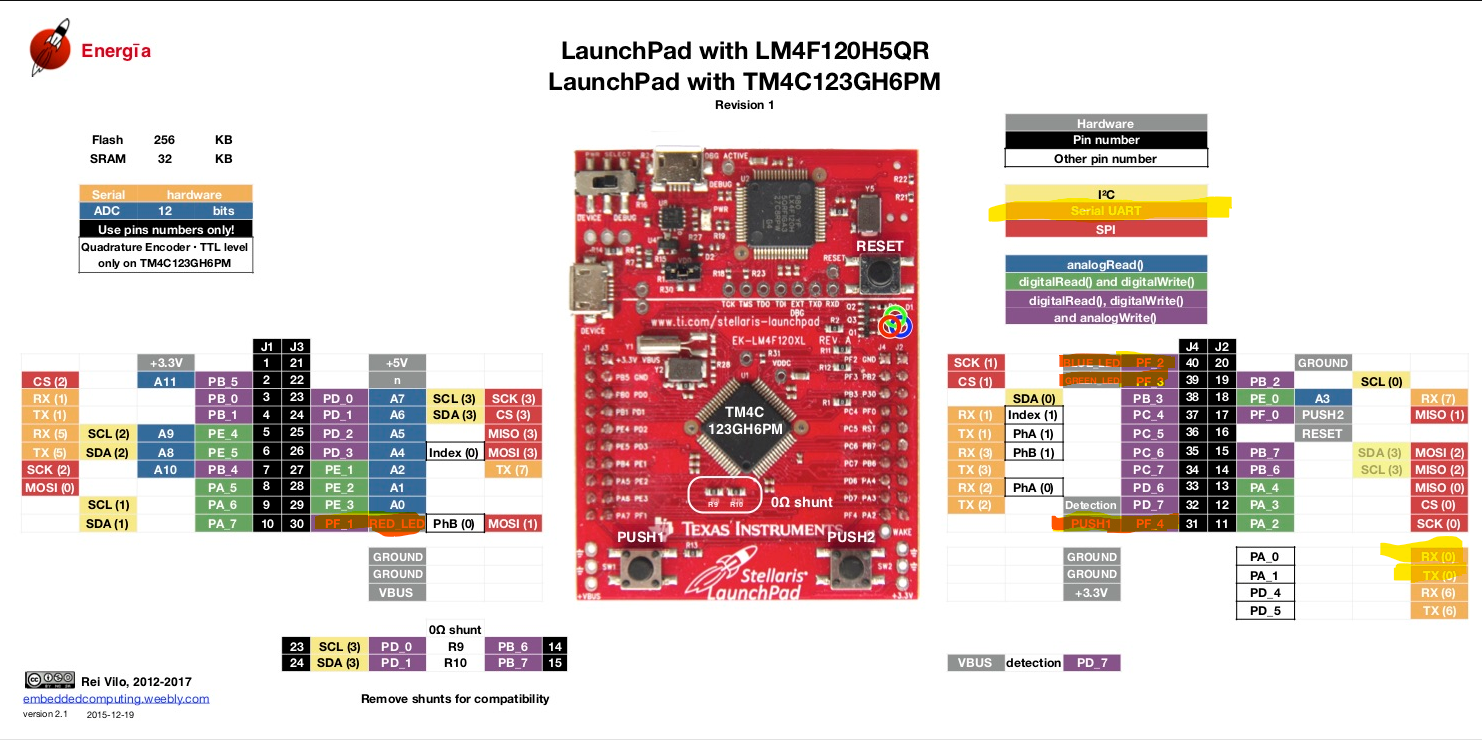
// Initialize the UART for console I/O.

//

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

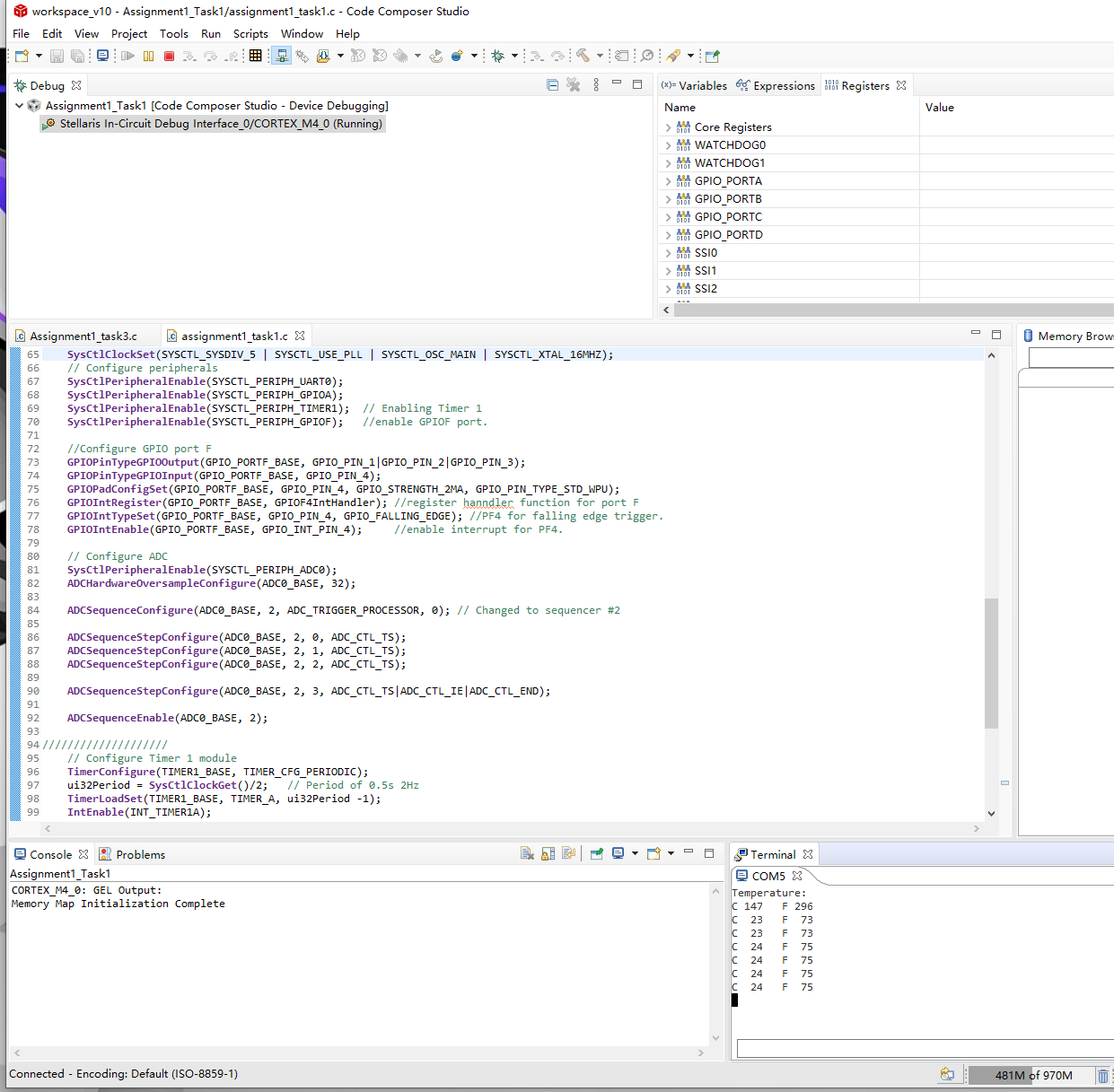
}

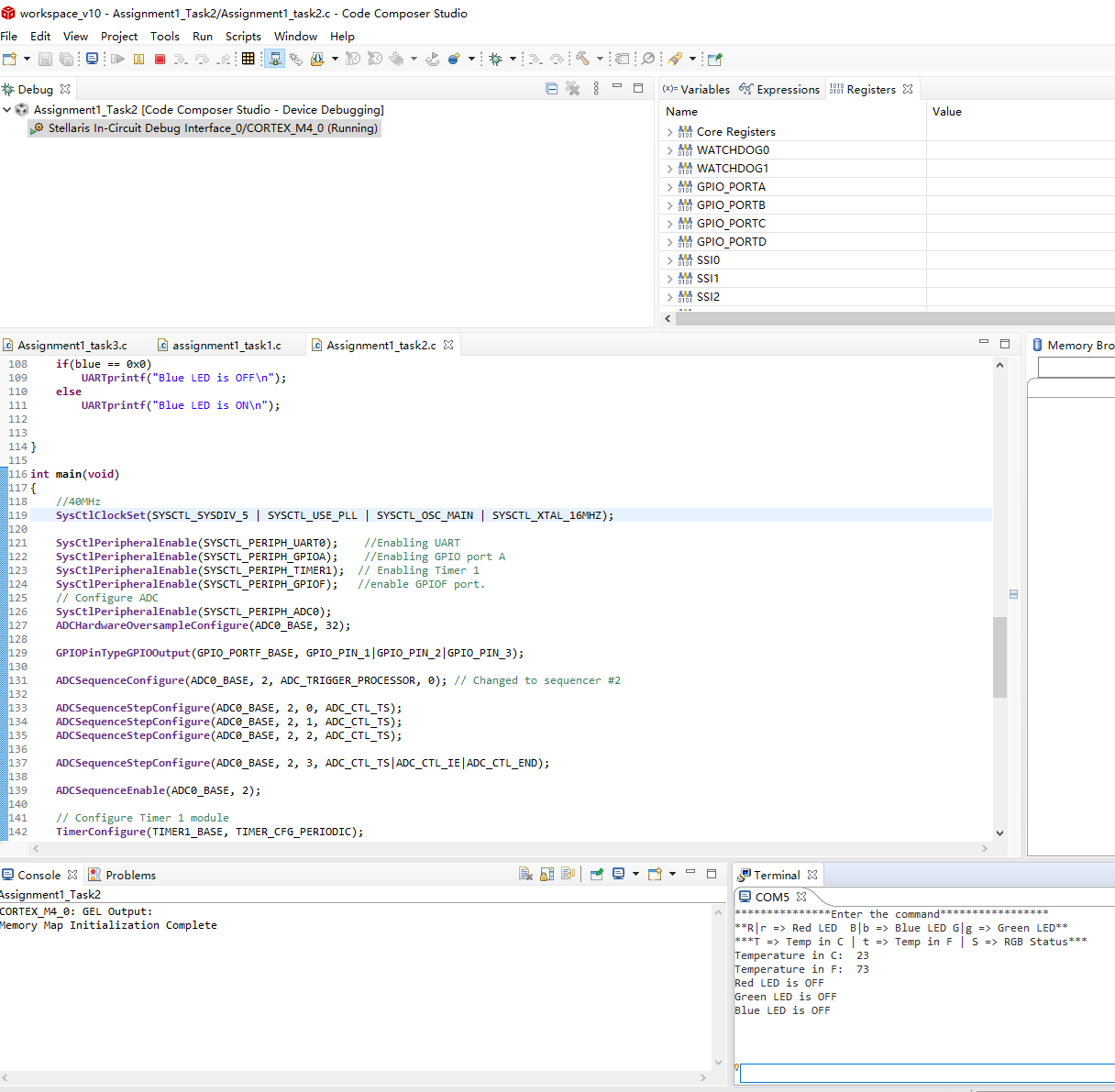
1. Block diagram and/or Schematics showing the components, pins used, and interface.

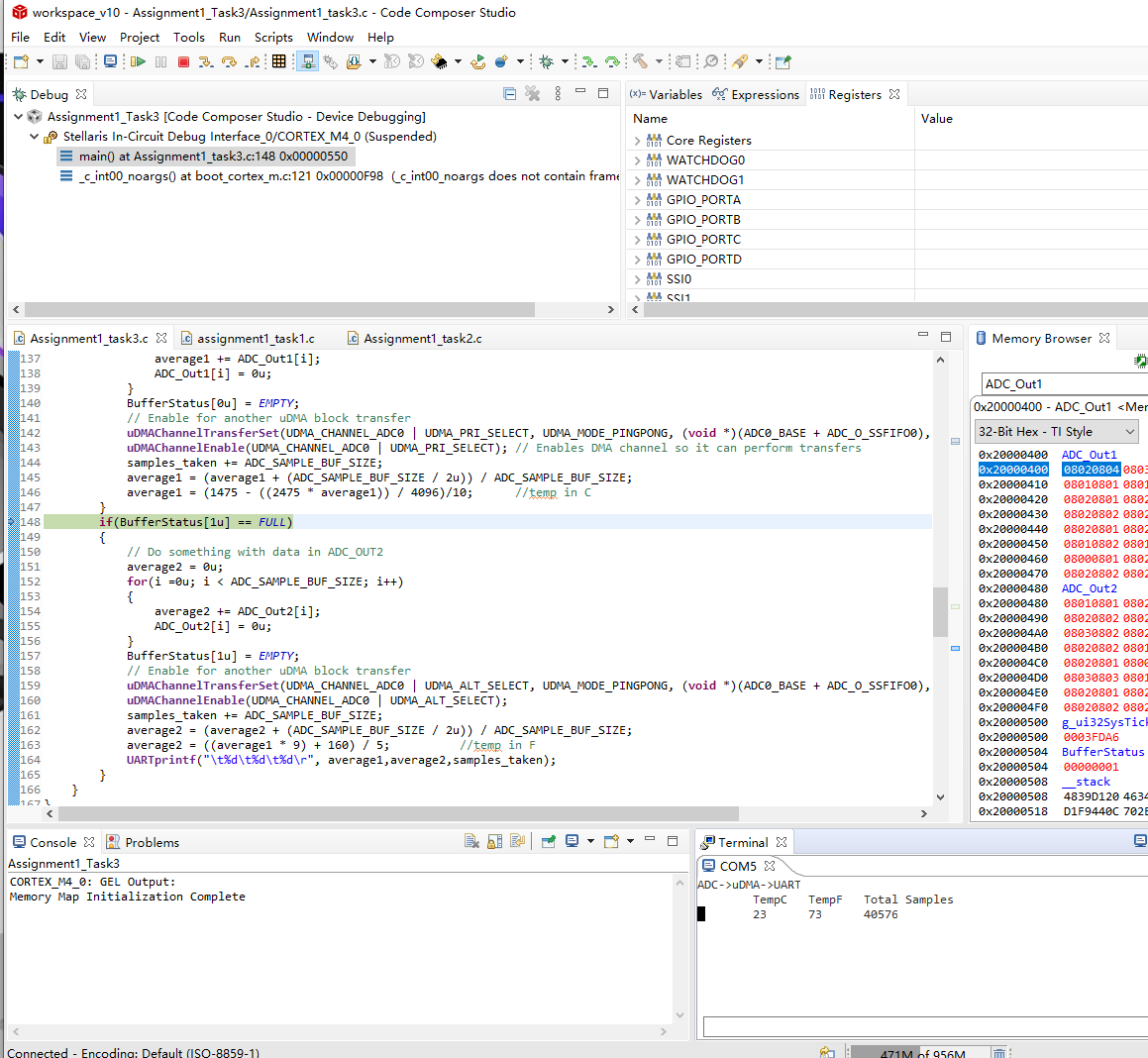


Pin used are highlighted.

1. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.









1. Declaration

I understand the Student Academic Misconduct Policy - http://studentconduct.unlv.edu/misconduct/policy.html

“This assignment submission is my own, original work”.

Xianjie Cao