CPE403 – Advanced Embedded Systems

# Design Assignment 01

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Rishawn Peppers Johnson

Email: Peppersj@unlv.nevada.edu

Github Repository link (root): <https://github.com/PeppersJ/v4e0nk_i3>

Youtube Playlist link (root): <https://drive.google.com/drive/folders/1fJ029-AAWjTnN-QrRqNLd0iLwKGm6A08?usp=sharing>

**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.

**Task 1**

/\* Created By: Rishawn Peppers Johnson

\* Date Created: 01 October 2020

\* Device: TivaC123GH6PM

\* CpE 403 Assignment 01 Task 01

\*

\* Purpose: Constantly display temperature through the terminal using a timer interrupt every 0.5s.

\* The PF4 button will toggle all three (R,G,B) LEDs using an interrupt.

\*

\* Inputs: None

\*

\* Outputs: Current temperature in F and C to terminal through UART.

\*

\* \*/**#include** <stdint.h> // Variable definitions for C99 Standard

**#include** <stdbool.h> // Boolean definitions for the C99 Standard

//#define TARGET\_IS\_BLIZZARD\_RB1 // For rom.h, defines TivaC type

**#include** "inc/tm4c123gh6pm.h" // def. for the interrupt and register assignments on the Tiva C Series device on the launchPad board

**#include** "inc/hw\_memmap.h" // Macros defining the memory map

**#include** "inc/hw\_types.h" // Common macros for TivaC

//#include "inc/hw\_ints.h" // Macros for TivaC interrupts

//#include "driverlib/rom.h" // Macros for calling functions in ROM

**#include** "driverlib/rom\_map.h" // Macros for default calling functions in ROM otherwise in Flash

**#include** "utils/uartstdio.h" // UART Console Driver Functions

**#include** "driverlib/interrupt.h"// Macros for NVIC Controller API of DriverLib

**#include** "driverlib/gpio.h" // API GPIO ports

**#include** "driverlib/debug.h" // Macros for debugging driverlib

**#include** "driverlib/sysctl.h" // Driver for SysTick

**#include** "driverlib/systick.h" // Driver for SysTick timer in NVIC

**#include** "driverlib/adc.h" // Drivers for ADC

**#include** "driverlib/timer.h" // Drivers for timers

**#include** "driverlib/uart.h" // Macros for UART

**#include** "driverlib/pin\_map.h" // Mapping of peripherals and pins

// Library error routine

**#ifdef** DEBUG

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

}

**#endif**

// Pins

**#define** RED\_LED GPIO\_PIN\_1

**#define** BLUE\_LED GPIO\_PIN\_2

**#define** GREEN\_LED GPIO\_PIN\_3

// Variables

uint32\_t ui32Period;

uint32\_t ui32ADC0Value[4]; // Stores ADC value (size of ADC sequencer)

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Prototypes

// Initialize Functions

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

**void** **init\_GPIO**();

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

**void** **init\_UART**();

// Interrupt Handlers

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

**void** **GPIOF0IntHandler**(**void**);

// Program Entry

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

//System clock to 40Mhz (PLL= 400Mhz / 10 = 40Mhz)

MAP\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

// Enable Peripherals

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0); // Running at default rate of 1Msps

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

// Configure Peripherals

init\_ADC();

init\_GPIO();

init\_TIMER();

init\_UART();

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

**IntMasterEnable**(); // Enable Master Interrupt

**TimerEnable**(TIMER0\_BASE, TIMER\_A); // Enable Timer 0

**ADCSequenceEnable**(ADC0\_BASE, 1); // Enable Sequencer 1

**while**(1) {

}

}

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

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0); // Using ADC0, sequencer 1, processor triggered, highest priority

// All 4 sequencers sampling internal temperature sensor

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

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

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

// Samples temperature, sets interrupt flag when done, tell ADC last conversion on sequencer 1

**ADCSequenceStepConfigure**(ADC0\_BASE, 1, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);

}

**void** **init\_GPIO**() {

// Initialize GPIO pins

GPIO\_PORTF\_LOCK\_R = 0x4C4F434B; // unlock GPIO Port F

GPIO\_PORTF\_CR\_R = 0x1F; // allow changes to PF4-0

// Set LED pins as outputs

**GPIOPinTypeGPIOOutput**( GPIO\_PORTF\_BASE, RED\_LED | GREEN\_LED | BLUE\_LED );

// Set PF0 as input

**GPIOPinTypeGPIOInput**( GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

//register the interrupt handler for PF0

**GPIOIntRegister**(GPIO\_PORTF\_BASE, GPIOF0IntHandler);

//SW2 goes low when pressed

**GPIOIntTypeSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_FALLING\_EDGE);

//enable interrupts on PF0

**GPIOIntEnable**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

}

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

/\*

Configure the timer as periodic, by omission it's in count down mode.

It counts from the load value to 0 and then resets back to the load value.

\*/

MAP\_TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_PERIODIC);

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

MAP\_TimerLoadSet(TIMER0\_BASE, TIMER\_A, ui32Period - 1);

MAP\_IntEnable(INT\_TIMER0A);

MAP\_TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

}

**void** **init\_UART**() {

// GPIO Setup

// Enable UART GPIO pins

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

// 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 internal 16MHz as clock

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

// Initialize for I/O console

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

}

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

// Clear the interrupt

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

// Clear status flag before writing to ADC

**ADCIntClear**(ADC0\_BASE, 1);

// Set ADC to trigger with software

**ADCProcessorTrigger**(ADC0\_BASE, 1);

// Copy from ADC FIFO to buffer

**ADCSequenceDataGet**(ADC0\_BASE, 1, ui32ADC0Value);

// Calculate average temperature

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

// Convert to C (datasheet's equation)

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

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

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

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

}

**void** **GPIOF0IntHandler**(**void**) { //interrupt handler for GPIO pin F0

// Clear interrupt flag on pin F0

MAP\_GPIOIntClear(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

// Toggle all LEDs

int32\_t status = **GPIOPinRead**(GPIO\_PORTF\_BASE, RED\_LED|BLUE\_LED|GREEN\_LED);

MAP\_GPIOPinWrite(GPIO\_PORTF\_BASE, RED\_LED|BLUE\_LED|GREEN\_LED, 0xFFFFFFFF ^ status);

}

**Task 2**

/\* Created By: Rishawn Peppers Johnson

\* Date Created: 01 October 2020

\* Device: TivaC123GH6PM

\* CpE 403 Assignment 01 Task 02

\*

\* Purpose: Using a user interface, execute the expected command. The possible commands are to

\* individually turn ON or OFF all three (R,G,B) LEDs, display the temperature in F or C,

\* and display the status of all the LEDs (ON or OFF). The PF4 button will toggle all

\* three (R,G,B) LEDs using an interrupt. The temperature value will be read and then

\* displayed in terminal through UART.

\*

\* Inputs: Single character menu choice in terminal:

\* T - Temperature in C

\* t - Temperature in F

\* R/G/B - Turn ON (R,G,B) LED (only one char at a time)

\* r/g/b - Turn OFF (R,G,B) LED (only one char at a time)

\* S - Display the status (ON or OFF) of LEDs

\*

\* Outputs: Requested value or confirmation of command to terminal through UART.

\*

\* \*/

**#include** <stdint.h> // Variable definitions for C99 Standard

**#include** <stdbool.h> // Boolean definitions for the C99 Standard

//#define TARGET\_IS\_BLIZZARD\_RB1 // For rom.h, defines TivaC type

**#include** "inc/tm4c123gh6pm.h" // def. for the interrupt and register assignments on the Tiva C Series device on the launchPad board

**#include** "inc/hw\_memmap.h" // Macros defining the memory map

**#include** "inc/hw\_types.h" // Common macros for TivaC

//#include "inc/hw\_ints.h" // Macros for TivaC interrupts

//#include "driverlib/rom.h" // Macros for calling functions in ROM

**#include** "driverlib/rom\_map.h" // Macros for default calling functions in ROM otherwise in Flash

**#include** "utils/uartstdio.h" // UART Console Driver Functions

**#include** "driverlib/interrupt.h"// Macros for NVIC Controller API of DriverLib

**#include** "driverlib/gpio.h" // API GPIO ports

**#include** "driverlib/debug.h" // Macros for debugging driverlib

**#include** "driverlib/sysctl.h" // Driver for SysTick

**#include** "driverlib/systick.h" // Driver for SysTick timer in NVIC

**#include** "driverlib/adc.h" // Drivers for ADC

**#include** "driverlib/timer.h" // Drivers for timers

**#include** "driverlib/uart.h" // Macros for UART

**#include** "driverlib/pin\_map.h" // Mapping of peripherals and pins

// Library error routine

**#ifdef** DEBUG

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

}

**#endif**

// Pins

**#define** RED\_LED GPIO\_PIN\_1

**#define** BLUE\_LED GPIO\_PIN\_2

**#define** GREEN\_LED GPIO\_PIN\_3

// Buffers

**char** cBuff[5];

// Variable0073

uint32\_t ui32Period;

uint32\_t ui32ADC0Value[4]; // Stores ADC value (size of ADC sequencer)

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// Prototypes

// Custom Functions

**void** **UARTprintLedStatus**(uint32\_t, uint8\_t);

// Initialize Functions

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

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

**void** **init\_GPIO**();

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

// Interrupt Handlers

void Timer0AIntHandler(void);

**void** **GPIOF0IntHandler**(**void**);

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

// Program Entry

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

// System clock to 40Mhz (PLL= 400Mhz / 10 = 40Mhz)

MAP\_SysCtlClockSet(SYSCTL\_SYSDIV\_5|SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_16MHZ);

// Enable Peripherals

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

MAP\_SysCtlDelay(30u);

// Configure Peripherals

ConfigureUART();

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

init\_GPIO();

init\_ADC();

init\_TIMER();

MAP\_IntMasterEnable(); // Enable Global Interrupts

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

**while**(1) {

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

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

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

// Read in from UART Buffer

**UARTgets**(cBuff, 2);

// Controls and UART Menu Display

**if** (cBuff[0] == 'R') { // Red LEDs

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, RED\_LED, RED\_LED);

} **else** **if**( cBuff[0] == 'r') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, RED\_LED, 0);

}

**if** (cBuff[0] == 'G') { // Green LEDs

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GREEN\_LED, GREEN\_LED);

} **else** **if**( cBuff[0] == 'g') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GREEN\_LED, 0);

}

**if** (cBuff[0] == 'B') { // Blue LEDs

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, BLUE\_LED, BLUE\_LED);

} **else** **if**( cBuff[0] == 'b') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, BLUE\_LED, 0);

}

**if** (cBuff[0] == 'T') // Temperature

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

**else** **if**( cBuff[0] == 't')

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

**if** (cBuff[0] == 'S') { // LED Status

**UARTprintf**("LED Status:\n");

**UARTprintf**("Red LED:\t");

UARTprintLedStatus(GPIO\_PORTF\_BASE, RED\_LED);

**UARTprintf**("Green LED:\t");

UARTprintLedStatus(GPIO\_PORTF\_BASE, GREEN\_LED);

**UARTprintf**("Blue LED:\t");

UARTprintLedStatus(GPIO\_PORTF\_BASE, BLUE\_LED);

}

}

}

**void** **UARTprintLedStatus**(uint32\_t ui32Port, uint8\_t ui8Pins) {

**volatile** int32\_t status = **GPIOPinRead**(ui32Port, ui8Pins);

**SysCtlDelay**(1000);

**if** (status == 0)

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

**else**

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

}

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

// GPIO setup for UART

// Enable GPIO Peripheral used by UART

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

// Enable UART0

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

// Configure GPIO pins for UART mode

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

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

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

// Use internal 16MHz as clock

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

// Initialize for I/O console

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

}

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

// Disable interrupts on sequencer 0

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

// Disable sample sequencer 0

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

// Set interrupt function

**ADCIntRegister**( ADC0\_BASE , 0, ADCseq0Handler );

// Using ADC0, sequencer 1, processor triggered, highest priority

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

// Configuration reading from internal temperature sensor as last in sequence.

// Causes interrupt when complete

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

// Re-enable sequencer 0

// Re-enable sequencer 0

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

// Clear status flag before writing to ADC

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

// Enable DMA for sequencer 0

//ADCSequenceDMAEnable( ADC0\_BASE, 0 );

// Re-enable interrupts for sequencer 0

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

}

**void** **init\_GPIO**() {

// Initialize GPIO pins

GPIO\_PORTF\_LOCK\_R = 0x4C4F434B; // unlock GPIO Port F

GPIO\_PORTF\_CR\_R = 0x1F; // allow changes to PF4-0

// Set LED pins as outputs

**GPIOPinTypeGPIOOutput**( GPIO\_PORTF\_BASE, RED\_LED | GREEN\_LED | BLUE\_LED );

// Set PF0 as input

**GPIOPinTypeGPIOInput**( GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

//register the interrupt handler for PF0

**GPIOIntRegister**(GPIO\_PORTF\_BASE, GPIOF0IntHandler);

//SW2 goes low when pressed

**GPIOIntTypeSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_FALLING\_EDGE);

//enable interrupts on PF0

**GPIOIntEnable**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

}

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

**TimerConfigure**(TIMER0\_BASE, TIMER\_CFG\_SPLIT\_PAIR | TIMER\_CFG\_A\_PERIODIC);

**TimerLoadSet**(TIMER0\_BASE, TIMER\_A, **SysCtlClockGet**()/16000 - 1);

// Enable ADC trigger output

**TimerControlTrigger**(TIMER0\_BASE, TIMER\_A, **true**);

// Timer stops when in debug mode

**TimerControlStall**(TIMER0\_BASE, TIMER\_A, **true**);

// Enable interrupt on Timer 0 A

//TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

**TimerEnable**(TIMER0\_BASE, TIMER\_A);

}

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

// Clear the interrupt

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

// Clear status flag before writing to ADC

**ADCIntClear**(ADC0\_BASE, 1);

// Copy from ADC FIFO to buffer

**ADCSequenceDataGet**(ADC0\_BASE, 0, ui32ADC0Value);

// Calculate average temperature

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

// Convert to C (datasheet's equation)

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

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

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

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

}

**void** **GPIOF0IntHandler**(**void**) { //interrupt handler for GPIO pin F0

// Clear interrupt flag on pin F0

MAP\_GPIOIntClear(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

// Toggle all LEDs

int32\_t status = **GPIOPinRead**(GPIO\_PORTF\_BASE, RED\_LED|BLUE\_LED|GREEN\_LED);

MAP\_GPIOPinWrite(GPIO\_PORTF\_BASE, RED\_LED|BLUE\_LED|GREEN\_LED, 0xFFFFFFFF ^ status);

}

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

**ADCSequenceDataGet**(ADC0\_BASE, 0, ui32ADC0Value);

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

}

**Task 3**

/\* Created By: Rishawn Peppers Johnson

\* Date Created: 01 October 2020

\* Device: TivaC123GH6PM

\* CpE 403 Assignment 01 Task 03

\*

\* Purpose: Using a user interface, execute the expected command. The possible commands are to

\* individually turn ON or OFF all three (R,G,B) LEDs, display the temperature in F or C,

\* and display the status of all the LEDs (ON or OFF). The PF4 button will toggle all

\* three (R,G,B) LEDs using an interrupt. The temperature value read from the ADC will

\* be stored in memory using uDMA. It will then be transfered to UART from memory to UART

\* using uDMA to be displayed in the terminal.

\*

\* Inputs: Single character menu choice in terminal:

\* T - Temperature in C

\* t - Temperature in F

\* R/G/B - Turn ON (R,G,B) LED (only one char at a time)

\* r/g/b - Turn OFF (R,G,B) LED (only one char at a time)

\* S - Display the status (ON or OFF) of LEDs

\*

\* Outputs: Requested value or confirmation of command to terminal through UART.

\*

\* \*/

**#include** <stdint.h> // Variable definitions for C99 Standard

**#include** <stdbool.h> // Boolean definitions for the C99 Standard

//#define TARGET\_IS\_BLIZZARD\_RB1 // For rom.h, defines TivaC type

**#include** "inc/tm4c123gh6pm.h" // def. for the interrupt and register assignments on the Tiva C Series device on the launchPad board

**#include** "inc/hw\_memmap.h" // Macros defining the memory map

**#include** "inc/hw\_types.h" // Common macros for TivaC

**#include** "inc/hw\_adc.h" // Macros for ADC hardware

**#include** "inc/hw\_udma.h" // Macros for uDMA registers

//#include "inc/hw\_ints.h" // Macros for TivaC interrupts

//#include "driverlib/rom.h" // Macros for calling functions in ROM

**#include** "driverlib/rom\_map.h" // Macros for default calling functions in ROM otherwise in Flash

**#include** "utils/uartstdio.h" // UART Console Driver Functions

**#include** "driverlib/interrupt.h"// Macros for NVIC Controller API of DriverLib

**#include** "driverlib/gpio.h" // API GPIO ports

**#include** "driverlib/debug.h" // Macros for debugging driverlib

**#include** "driverlib/sysctl.h" // Driver for SysTick

**#include** "driverlib/systick.h" // Driver for SysTick timer in NVIC

**#include** "driverlib/adc.h" // Drivers for ADC

**#include** "driverlib/timer.h" // Drivers for timers

**#include** "driverlib/uart.h" // Macros for UART

**#include** "driverlib/udma.h" // Drivers and macros for uDMA Controller

**#include** "driverlib/pin\_map.h" // Mapping of peripherals and pins

// Library error routine

**#ifdef** DEBUG

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

}

**#endif**

// uDMA Types

**enum** BUFFERSTATUS { *EMPTY*, *FILLING*, *FULL* };

// Pins

**#define** RED\_LED GPIO\_PIN\_1

**#define** BLUE\_LED GPIO\_PIN\_2

**#define** GREEN\_LED GPIO\_PIN\_3

// Buffers

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

**#define** MEM\_BUFFER\_SIZE 1024

**char** cBuff[5]; // UART Input buffer

**static** uint32\_t g\_ui32SrcBuf[MEM\_BUFFER\_SIZE];

**static** uint32\_t g\_ui32DstBuf[MEM\_BUFFER\_SIZE];

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

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

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

// uDMA control table aligned to 1024-byte boundary

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

uint8\_t ucControlTable[1024];

// Variables

// Define transfer counter

**static** uint32\_t g\_ui32MemXferCount = 0;

// Define errors counters

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

**static** uint32\_t g\_ui32BadISR = 0u;

// Accumulator for system ticks

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

// Prototypes

// Custom Functions

**void** **UARTprintLedStatus**(uint32\_t, uint8\_t);

// Initialize Functions

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

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

**void** **init\_GPIO**();

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

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

// Interrupt Handlers

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

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

**void** **GPIOF0IntHandler**(**void**);

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

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

// Program Entry

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

// Variables

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

**volatile** uint32\_t ui32TempAvg;

**volatile** uint32\_t ui32TempValueC;

**volatile** uint32\_t ui32TempValueF;

// System clock to 160Mhz (PLL= 400Mhz / 2.5 = 160Mhz)

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

// Enable Peripherals

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UDMA);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

MAP\_SysCtlDelay(30u);

// Enable System Clock

MAP\_SysTickPeriodSet( **SysCtlClockGet**() / 100000u );

//SysTickIntRegister( SysTickIntHandler );

MAP\_SysTickIntEnable();

MAP\_SysTickEnable();

// Initialize

BufferStatus[0] = *FILLING*;

BufferStatus[1] = *EMPTY*;

samples\_taken = 0u;

// Configure Peripherals

ConfigureUART();

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

init\_DMA();

init\_GPIO();

init\_ADC();

init\_TIMER();

MAP\_IntMasterEnable(); // Enable Global Interrupts

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

**while**(1) {

// Read into Buffer1 or Buffer2 if the other is full

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

average1 = 0u; // Reset average to accumulate

// Sum and clear values from buffer

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

average1 += ADC\_Out1[i];

ADC\_Out1[i] = 0u;

}

BufferStatus[0u] = *EMPTY*; // Clear buffer

// Mark and enable to be transfered into ADC\_Out1

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 );

// Update sample count and average

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

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

}

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

average2 = 0u; // Reset average to accumulate

// Sum and clear values from buffer

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

average2 += ADC\_Out2[i];

ADC\_Out2[i] = 0u;

}

BufferStatus[1u] = *EMPTY*; // Clear buffer

// Mark and enable to be transfered into ADC\_Out2

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 );

// Update sample count and average

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

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

// UARTprintf("\t%d\t\t%d\t\t%d\r", ui32TempValueC, ui32TempValueF, samples\_taken);

}

// Calculate Temperature in C and F

ui32TempAvg = (average1 + average2) / 2;

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

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

// Read in from UART Buffer

**UARTgets**(cBuff, 2);

// Controls and UART Menu Display

**if** (cBuff[0] == 'R') { // Red LEDs

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, RED\_LED, RED\_LED);

} **else** **if**( cBuff[0] == 'r') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, RED\_LED, 0);

}

**if** (cBuff[0] == 'G') { // Green LEDs

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GREEN\_LED, GREEN\_LED);

} **else** **if**( cBuff[0] == 'g') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, GREEN\_LED, 0);

}

**if** (cBuff[0] == 'B') { // Blue LEDs

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, BLUE\_LED, BLUE\_LED);

} **else** **if**( cBuff[0] == 'b') {

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

**GPIOPinWrite**(GPIO\_PORTF\_BASE, BLUE\_LED, 0);

}

**if** (cBuff[0] == 'T') // Temperature

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

**else** **if**( cBuff[0] == 't')

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

**if** (cBuff[0] == 'S') { // LED Staus

**UARTprintf**("LED Status:\n");

**SysCtlDelay**(1000);

**UARTprintf**("Red LED:\t");

UARTprintLedStatus(GPIO\_PORTF\_BASE, RED\_LED);

**UARTprintf**("Green LED:\t");

UARTprintLedStatus(GPIO\_PORTF\_BASE, GREEN\_LED);

**UARTprintf**("Blue LED:\t");

UARTprintLedStatus(GPIO\_PORTF\_BASE, BLUE\_LED);

}

}

}

**void** **UARTprintLedStatus**(uint32\_t ui32Port, uint8\_t ui8Pins) {

// Display ON or OFF to UART

**volatile** int32\_t status = **GPIOPinRead**(ui32Port, ui8Pins);

**SysCtlDelay**(1000);

**if** (status == 0)

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

**else**

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

}

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

// GPIO setup for UART

// Enable UART GPIO pins

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

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 internal 16MHz as clock

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

// Initialize for I/O console

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

}

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

// Initialize uDMA on ADC CH0 in Ping-Pong Mode

**uDMAEnable**();

**uDMAControlBaseSet**(ucControlTable);

uDMAChannelAttributeDisable( UDMA\_CHANNEL\_ADC0, UDMA\_ATTR\_ALTSELECT |

UDMA\_ATTR\_HIGH\_PRIORITY| UDMA\_ATTR\_REQMASK);

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

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);

}

**void** **init\_GPIO**() {

// Initialize GPIO pins

GPIO\_PORTF\_LOCK\_R = 0x4C4F434B; // unlock GPIO Port F

GPIO\_PORTF\_CR\_R = 0x1F; // allow changes to PF4-0

// Set LED pins as outputs

**GPIOPinTypeGPIOOutput**( GPIO\_PORTF\_BASE, RED\_LED | GREEN\_LED | BLUE\_LED );

// Set PF0 as input

**GPIOPinTypeGPIOInput**( GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

//register the interrupt handler for PF0

**GPIOIntRegister**(GPIO\_PORTF\_BASE, GPIOF0IntHandler);

//SW2 goes low when pressed

**GPIOIntTypeSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_FALLING\_EDGE);

//enable interrupts on PF0

**GPIOIntEnable**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

}

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

// Initialize Timer 0

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

MAP\_TimerLoadSet(TIMER0\_BASE, TIMER\_A, **SysCtlClockGet**()/16000 - 1);

// Enable ADC trigger output

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

// Timer stops when in debug mode

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

// Enable interrupt on Timer 0 A

**TimerIntRegister**(TIMER0\_BASE, TIMER\_A, Timer0AIntHandler);

MAP\_TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

}

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

// ADC CH0 Timer Triggered for uDMA

// Set GPIO PIN to read in ADC

**GPIOPinTypeADC**(GPIO\_PORTE\_BASE, GPIO\_PIN\_2);

**SysCtlDelay**(80);

// Set ADC clock as PIOSC / 1 at half rate (16MHz/1) / 2 = 8 MHz

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

// Wait for configuration to finish

**SysCtlDelay**(10);

// Disable interrupts on ADC0

**IntDisable**( INT\_ADC0SS0 );

// Disable interrupts on sequencer 0

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

// Disable sample sequencer 0

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

// Set interrupt function

//ADCIntRegister( ADC0\_BASE , 0, ADCseq0Handler );

// Using ADC0, sequencer 1, processor triggered, highest priority

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

// Configuration reading from internal temperature sensor as last in sequence.

// Causes interrupt when complete

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

// Re-enable sequencer 0

// Re-enable sequencer 0

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

// Clear status flag before writing to ADC

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

// Enable DMA for sequencer 0

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

// Re-enable interrupts for sequencer 0

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

// Re-enable interrupts on ADC 0

**IntEnable**( INT\_ADC0SS0 );

}

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

// ADCSequenceDataGet(ADC0\_BASE, 0, ui32ADC0Value);

// UARTprintf("Seq Interrupt");

**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\_PRI\_SELECT) == UDMA\_MODE\_STOP)

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

{

BufferStatus[0] = *FILLING*;

BufferStatus[1] = *FULL*;

}

}

// uDMA error handler

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

uint32\_t ui32Status;

ui32Status = **uDMAErrorStatusGet**();

**if**(ui32Status)

{

**uDMAErrorStatusClear**();

g\_ui32DMAErrCount++;

}

}

**void** **GPIOF0IntHandler**(**void**) {

// Clear interrupt flag on pin F0

**GPIOIntClear**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

// Toggle all LEDs

int32\_t status = **GPIOPinRead**(GPIO\_PORTF\_BASE, RED\_LED|BLUE\_LED|GREEN\_LED);

**GPIOPinWrite**(GPIO\_PORTF\_BASE, RED\_LED|BLUE\_LED|GREEN\_LED, 0xFFFFFFFF ^ status);

}

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

g\_ui32SysTickCount++;

// UARTprintf("Sys Tick ");

}

// For debugging purposes

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

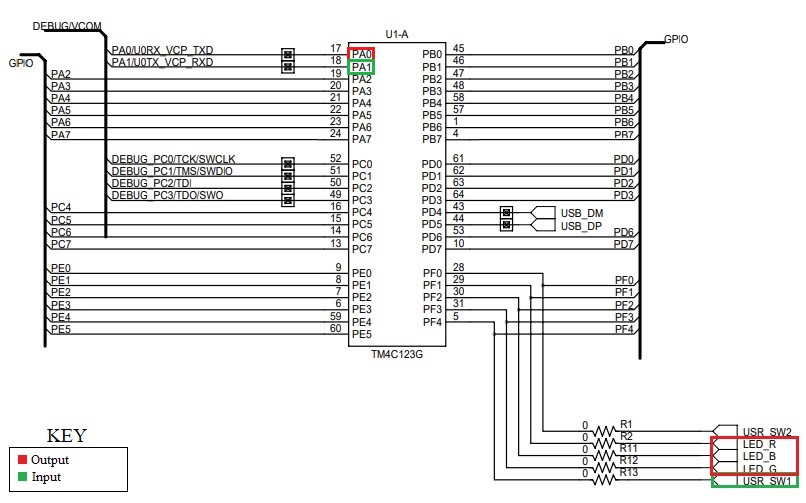
// Clear the interrupt

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

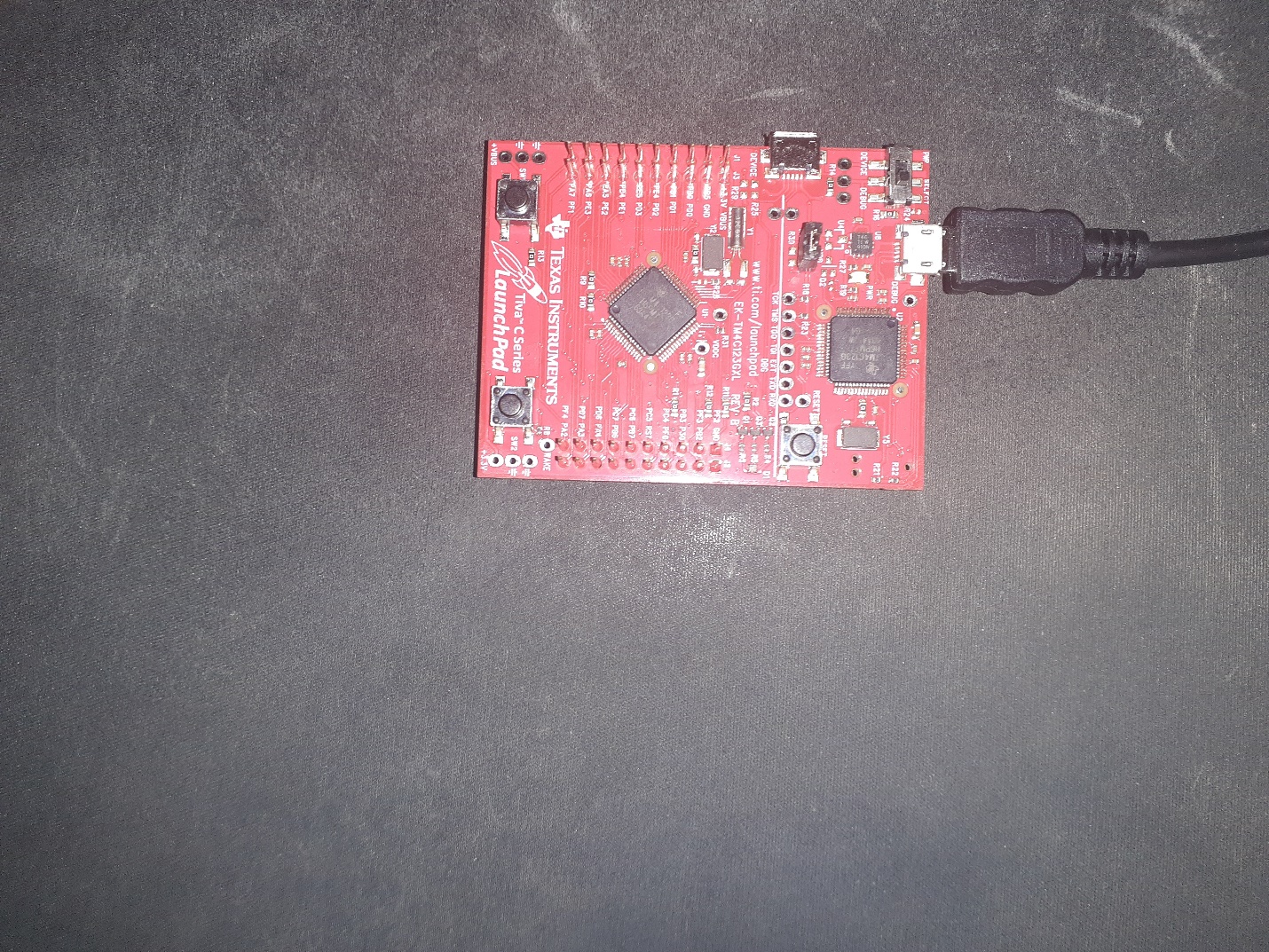
// UARTprintf("Timer Int ");

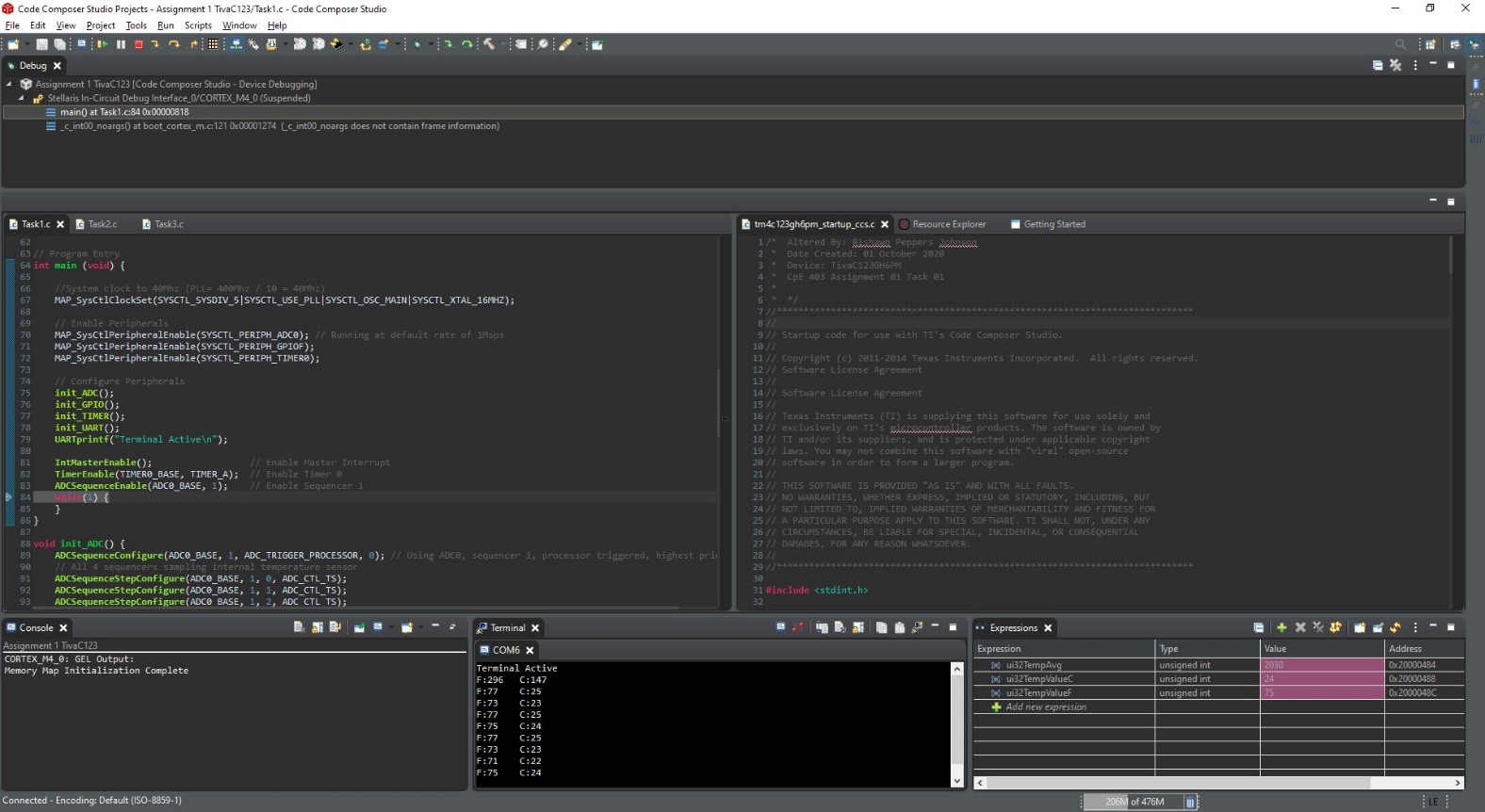
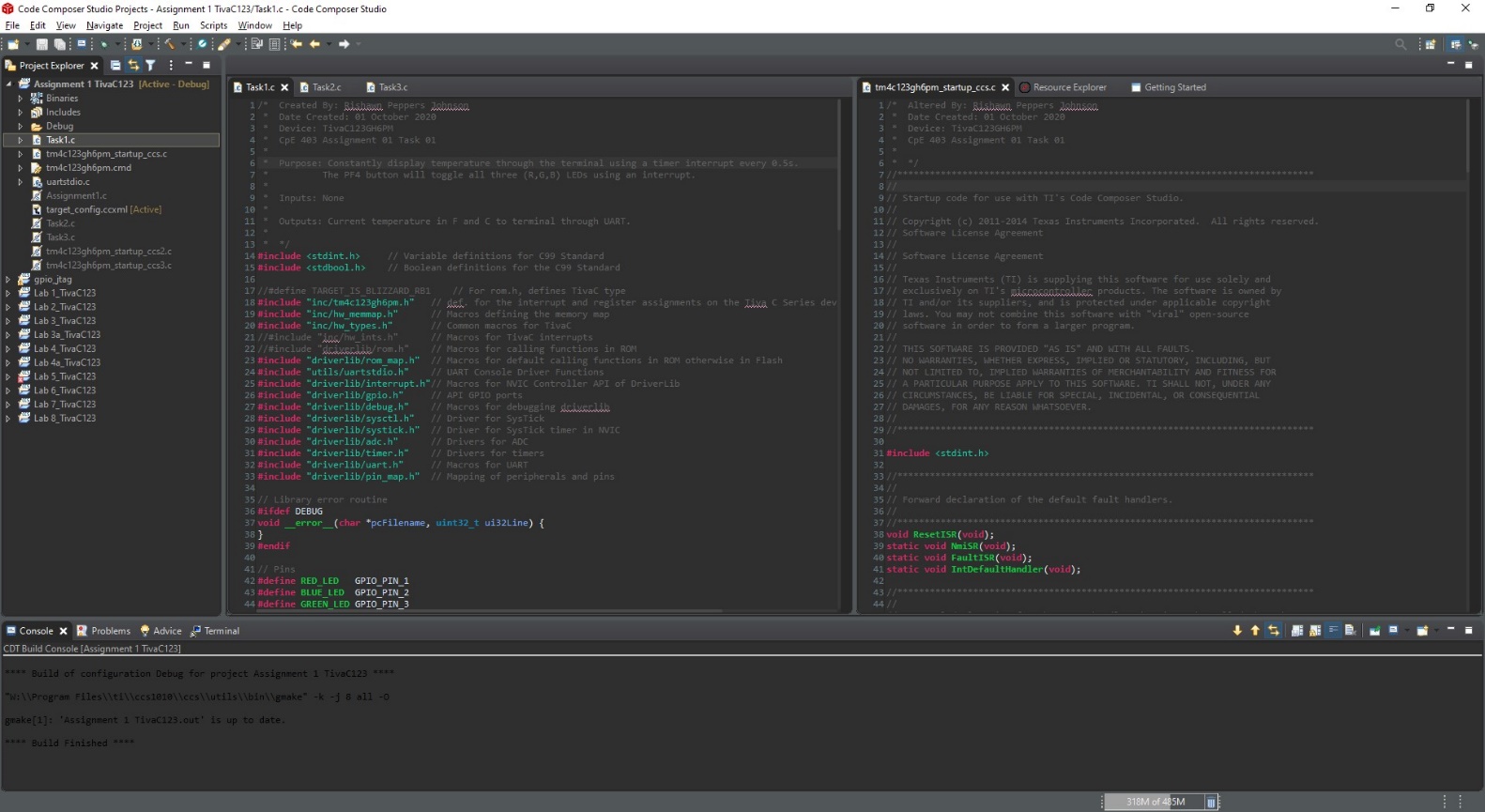
}

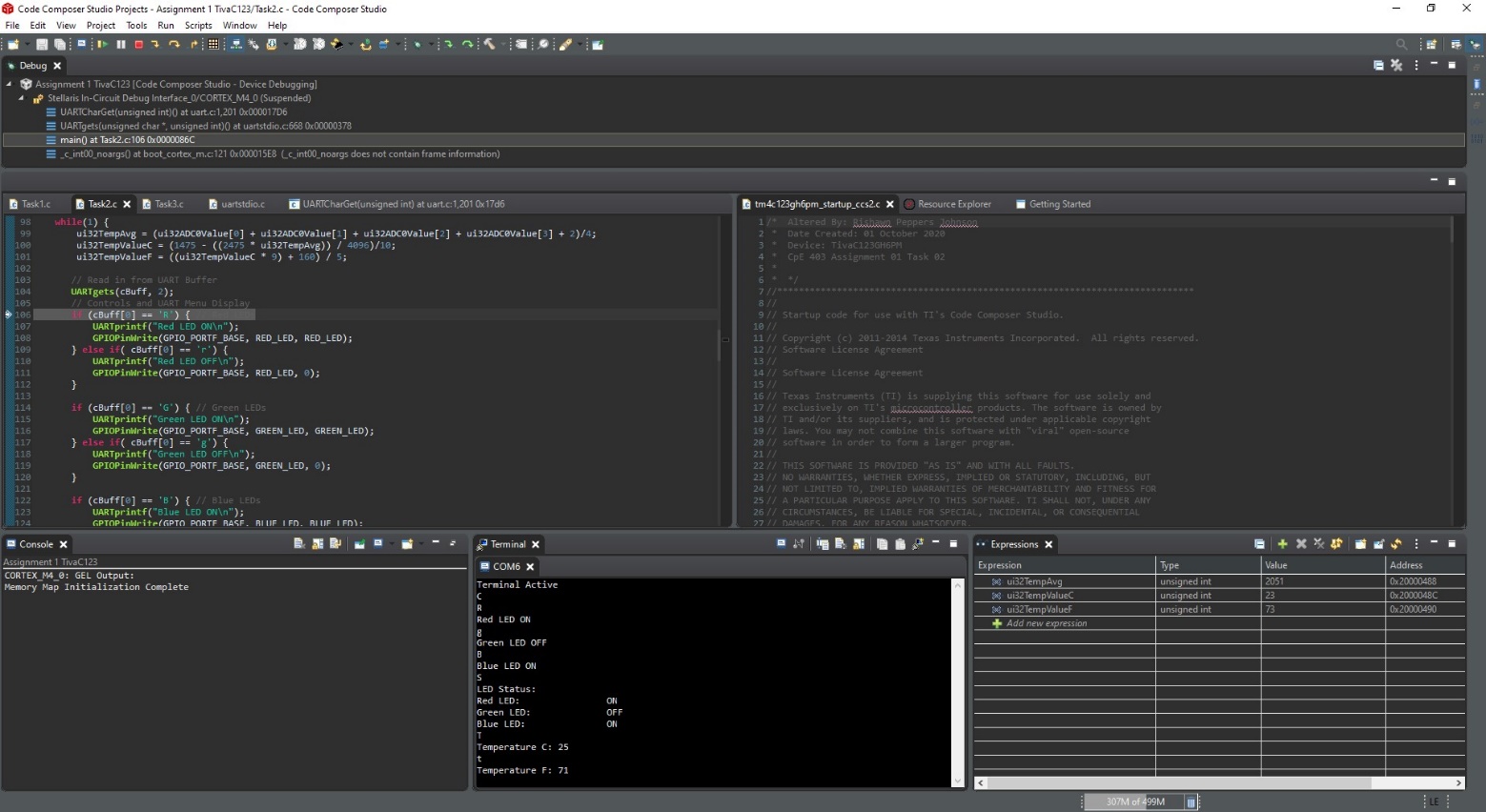
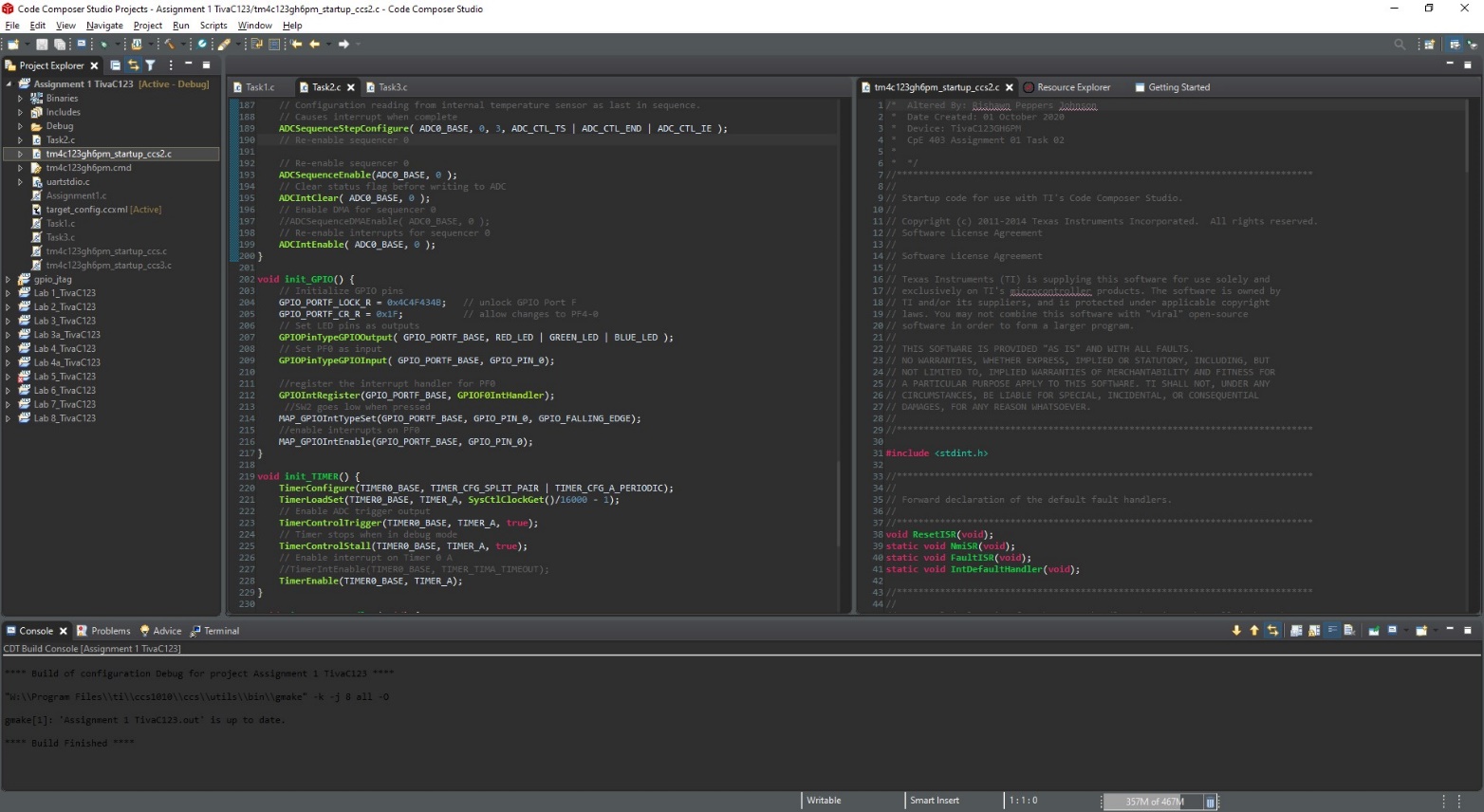
1. Block diagram and/or Schematics showing the components, pins used, and interface.

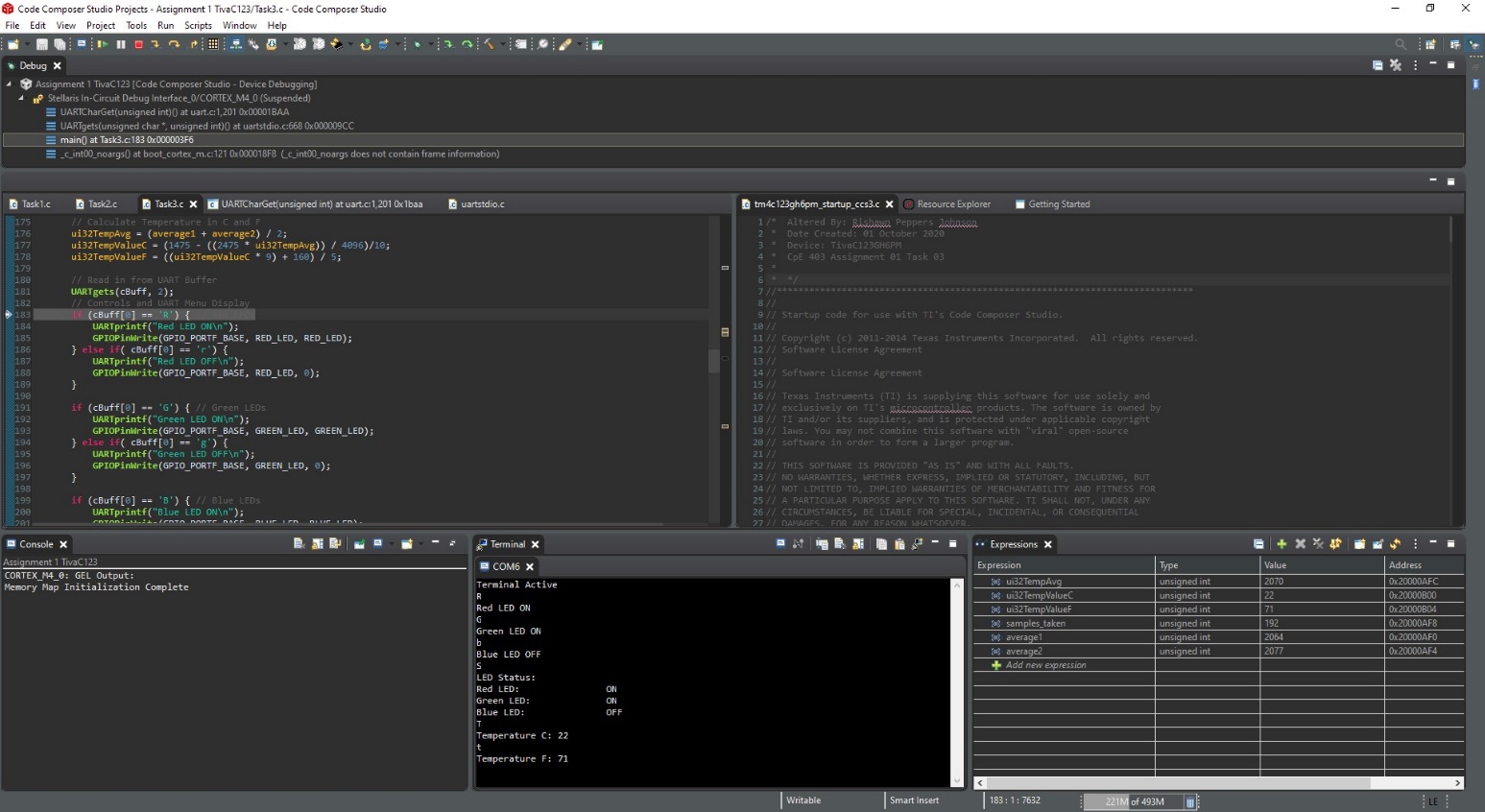
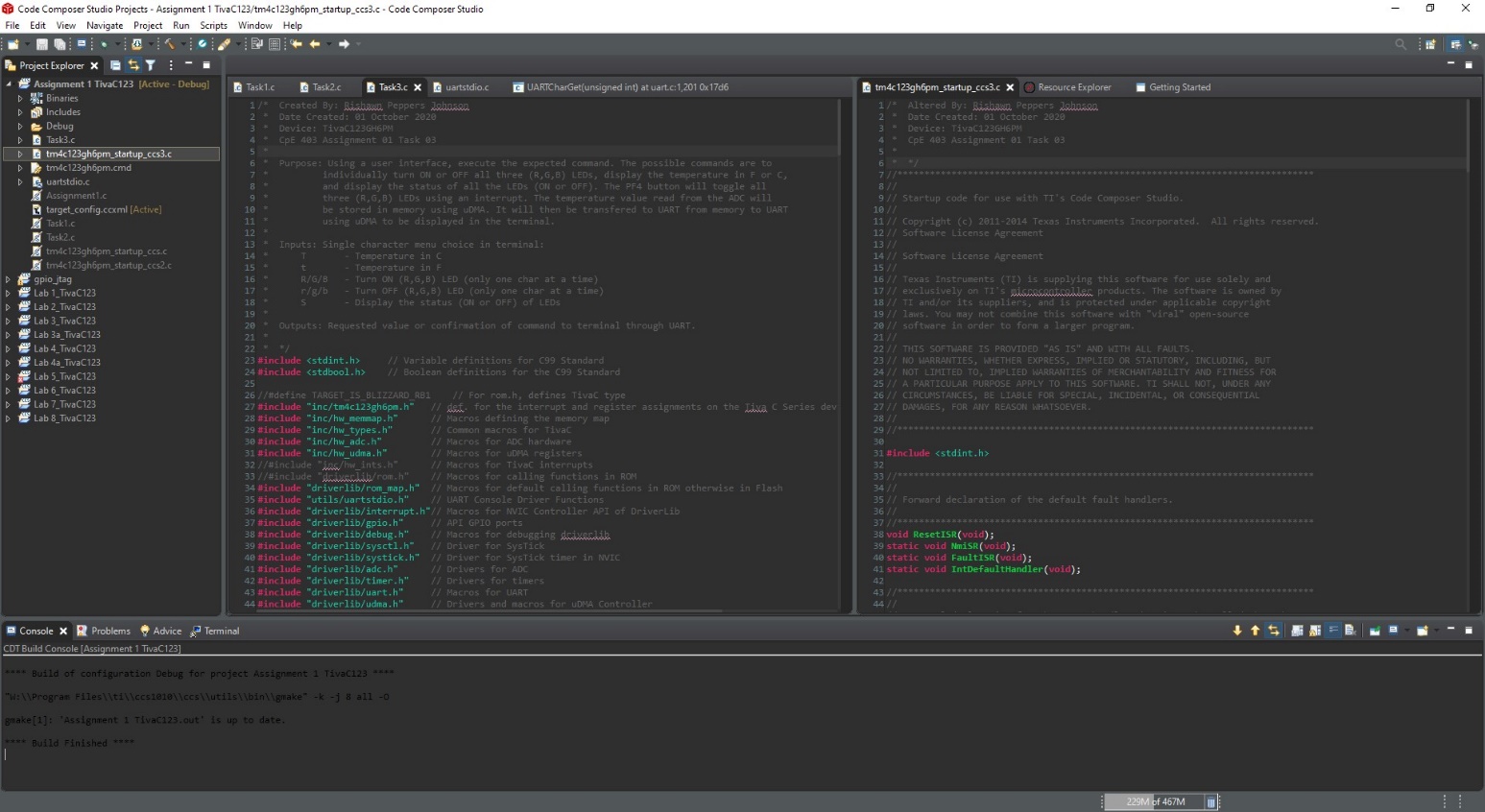


1. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.









1. Declaration

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

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

Rishawn Peppers Johnson