CPE403 – Advanced Embedded Systems

# Design Assignment # 1

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Jeb Marinas

Email:marinj4@unlv.nevada.edu

Github Repository link (root): https://github.com/jebmarinas/Micro\_projects

Youtube Playlist link (root): https://youtube.com/channel/UC4UYhdLZJwQUP\_ce9kvbNMg

**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:

1. /\*
2. \* TIVAC Assignment 1 - Task 1
3. \* 1) Display device temperature every 0.5s
4. \* 2) Toggle RGB LEDs using PF0
5. \*/
6. **#include** <stdint.h>
7. **#include** <stdbool.h>
8. **#include** "inc/hw\_memmap.h"
9. **#include** "inc/hw\_types.h"
10. //#include "inc/hw\_ints.h"
11. **#include** "inc/tm4c123gh6pm.h"
12. **#include** "driverlib/gpio.h"
13. **#include** "driverlib/pin\_map.h"
14. **#include** "driverlib/sysctl.h"
15. **#include** "driverlib/uart.h"
16. **#include** "driverlib/interrupt.h"
17. **#include** "driverlib/timer.h"
18. **#include** "driverlib/debug.h"
19. **#include** "driverlib/adc.h"
20. **#include** "driverlib/sysctl.h"
21. **#include** "utils/uartstdio.h"
22. **#include** <string.h>
23. **#ifdef** DEBUG
24. void\_\_error\_\_(**char** \*pcFilename, uint32\_t ui32Line)
25. {
26. }
27. **#endif**
28. // Globals
29. uint32\_t ui32Period;
30. **char** buffer[4];
31. uint32\_t ui32ADC0Value[4];
32. **volatile** uint32\_t ui32TempAvg;
33. **volatile** uint32\_t ui32TempValueC;
34. **volatile** uint32\_t ui32TempValueF;
35. **void** **ConfigurePORT**(**void**);
36. // Timer 1 ISR
37. **void** **Timer1IntHandler**(**void**)
38. {
39. // Clear the timer interrupt
40. **TimerIntClear**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);
41. **ADCIntClear**(ADC0\_BASE, 2);
42. **ADCProcessorTrigger**(ADC0\_BASE, 2);
43. **ADCSequenceDataGet**(ADC0\_BASE, 2, ui32ADC0Value);
44. ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2)/4;
45. ui32TempValueC = (1475 - ((2475 \* ui32TempAvg)) / 4096)/10;
46. ui32TempValueF = ((ui32TempValueC \* 9) + 160) / 5;
47. **UARTprintf**("C %3d\t",ui32TempValueC );
48. **UARTprintf**("F %3d\t",ui32TempValueF );
49. **UARTprintf**("\n");
50. }
51. **int** **main**(**void**) {
52. // Configure Clock
53. **SysCtlClockSet**(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_OSC\_MAIN | SYSCTL\_XTAL\_16MHZ);
54. // Configure peripherals
55. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_UART0);
56. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOA);
57. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_TIMER1); // Enabling Timer 1
58. ConfigurePORT();
59. // Configure ADC
60. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_ADC0);
61. **ADCHardwareOversampleConfigure**(ADC0\_BASE, 32);
62. **ADCSequenceConfigure**(ADC0\_BASE, 2, ADC\_TRIGGER\_PROCESSOR, 0); // Changed to sequencer #2
63. **ADCSequenceStepConfigure**(ADC0\_BASE, 2, 0, ADC\_CTL\_TS);
64. **ADCSequenceStepConfigure**(ADC0\_BASE, 2, 1, ADC\_CTL\_TS);
65. **ADCSequenceStepConfigure**(ADC0\_BASE, 2, 2, ADC\_CTL\_TS);
66. **ADCSequenceStepConfigure**(ADC0\_BASE, 2, 3, ADC\_CTL\_TS|ADC\_CTL\_IE|ADC\_CTL\_END);
67. **ADCSequenceEnable**(ADC0\_BASE, 2);
68. //register the interrupt handler for PF0
69. // GPIOIntRegister(GPIO\_PORTF\_BASE, GPIOF0IntHandler);
70. //SW2 goes low when pressed
71. **GPIOIntTypeSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_FALLING\_EDGE);
72. // Configure Timer 1 module
73. **TimerConfigure**(TIMER1\_BASE, TIMER\_CFG\_PERIODIC);
74. ui32Period = **SysCtlClockGet**()/2; // Period of 0.5s 2Hz
75. **TimerLoadSet**(TIMER1\_BASE, TIMER\_A, ui32Period -1);
76. **IntEnable**(INT\_TIMER1A);
77. **TimerIntEnable**(TIMER1\_BASE, TIMER\_TIMA\_TIMEOUT);
78. // Configure pins for UART
79. //GPIOPinConfigure(GPIO\_PA0\_U0RX);
80. **GPIOPinConfigure**(GPIO\_PA1\_U0TX);
81. **GPIOPinTypeUART**(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);
82. **UARTClockSourceSet**(UART0\_BASE, UART\_CLOCK\_PIOSC);
83. **UARTStdioConfig**(0, 115200, 16000000);
84. // Enable interrupts
85. **IntMasterEnable**();
86. **GPIOIntEnable**(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);
87. **TimerEnable**(TIMER1\_BASE, TIMER\_A);
88. **ADCSequenceEnable**(ADC0\_BASE, 2);
89. // Initial message to terminal display
90. **UARTprintf**("Temperature:\n");
91. **while** (1){};
92. }
93. **void** **ConfigurePORT**(**void**)
94. {
95. //Port configuration (LEDS)
96. //Enable GPIOF port
97. **SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_GPIOF);
98. //set LEDS connected to pins as outputs
99. **GPIOPinTypeGPIOOutput**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3);
100. //Unlock Pin F0 to use an interrupt on SW2
101. SYSCTL\_RCGC2\_R |= 0x00000020; // activate clock for Port F
102. GPIO\_PORTF\_LOCK\_R = 0x4C4F434B; // unlock GPIO Port F
103. GPIO\_PORTF\_CR\_R = 0x1F; // allow changes to PF4-0
104. // only PF0 needs to be unlocked, other bits can't be locked
105. GPIO\_PORTF\_AMSEL\_R = 0x00; // disable analog on PF
106. GPIO\_PORTF\_PCTL\_R = 0x00000000; // PCTL GPIO on PF4-0
107. GPIO\_PORTF\_DIR\_R = 0x0E; // PF4,PF0 in, PF3-1 out
108. GPIO\_PORTF\_AFSEL\_R = 0x00; // disable alt funct on PF7-0
109. GPIO\_PORTF\_PUR\_R = 0x11; // enable pull-up on PF0 and PF4
110. GPIO\_PORTF\_DEN\_R = 0x1F; // enable digital I/O on PF4-0
111. }

**Task 2:**

**#include** <stdint.h>

**#include** <stdbool.h>

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

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

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

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

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

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

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

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

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

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

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

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

**typedef** **struct**{

uint32\_t adcBuffer[4];

uint32\_t AVG;

uint32\_t C;

uint32\_t F;

}Temp\_t;

**typedef** **enum**{

*WAIT*,

*RED*, *BLUE*, *GREEN*,

*red*, *blue*, *green*,

*TEMP*,

*temp*,

*STAT*

}State\_enum;

**typedef** **struct**{

**unsigned** **char** input;

}UART\_t;

**void** **check\_status**(uint32\_t status, **const** **char**\* string);

**void** **send\_status**(**const** **char**\* string, **bool** status);

**void** **calculate\_avg\_temperature**(Temp\_t \*temp\_t);

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

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

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

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

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

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

{

//Config System Clock

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

Temp\_t temp\_t;

**volatile** **unsigned** **char** status\_of\_leds;

**volatile** State\_enum state, nextState = *WAIT*;

**volatile** UART\_t uart\_t;

**volatile** uint32\_t period;

//Config UART

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

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

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

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

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

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

//

//Config ADC

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

**ADCSequenceConfigure**(ADC0\_BASE, 1, ADC\_TRIGGER\_PROCESSOR, 0);

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

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

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

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

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

//Config LEDs

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

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

**GPIOPadConfigSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

//Next State for UI

**while**(1)

{

**if**(state == *WAIT*)

{

**while**(!**UARTCharsAvail**(UART0\_BASE)){};

uart\_t.input = **UARTCharGet**(UART0\_BASE);

**if**(uart\_t.input == 'R') state = *RED*;

**else** **if**(uart\_t.input == 'B') state = *BLUE*;

**else** **if**(uart\_t.input == 'G') state = *GREEN*;

**else** **if**(uart\_t.input == 'r') state = *red*;

**else** **if**(uart\_t.input == 'b') state = *blue*;

**else** **if**(uart\_t.input == 'g') state = *green*;

**else** **if**(uart\_t.input == 'T') state = *TEMP*;

**else** **if**(uart\_t.input == 't') state = *temp*;

**else** **if**(uart\_t.input == 'S') state = *STAT*;

}

**else** **if**(state == *RED*){

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

}

**else** **if**(state == *red*){

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

}

**else** **if**(state == *BLUE*){

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

}

**else** **if**(state == *blue*){

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

}

**else** **if**(state == *GREEN*){

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

}

**else** **if**(state == *green*){

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

}

**else** **if**(state == *TEMP*){

calculate\_avg\_temperature(&temp\_t);

**UARTprintf**("C %3d\t", temp\_t.C);

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

}

**else** **if**(state == *temp*){

calculate\_avg\_temperature(&temp\_t);

**UARTprintf**("C %3d\t", temp\_t.F);

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

}

**else** **if**(state == *STAT*){

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

check\_status((status\_of\_leds&RED\_LED) >> 1, "Red: ");

check\_status((status\_of\_leds&BLUE\_LED) >> 2, "Blue: ");

check\_status((status\_of\_leds&GREEN\_LED) >> 3, "Green: ");

}

state = *WAIT*;

}

}

**void** **check\_status**(uint32\_t status, **const** **char**\* string){

**if**(status)

send\_status(string, 1);

**else**

send\_status(string, 0);

}

**void** **send\_status** (**const** **char**\* string, **bool** status){

**UARTprintf**(string);

**if**(status)

**UARTprintf**("ON");

**else**

**UARTprintf**("OFF");

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

}

//Calculate the average

**void** **calculate\_avg\_temperature**(Temp\_t \*temp\_t){

//Trigger ADC and wait for interrupt

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

**while**(!**ADCIntStatus**(ADC0\_BASE, 1, **false**)){};

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

**ADCSequenceDataGet**(ADC0\_BASE, 1, temp\_t->adcBuffer);

//Calculate Average

temp\_t->AVG = (temp\_t->adcBuffer[0] + temp\_t->adcBuffer[1] + temp\_t->adcBuffer[2] + temp\_t->adcBuffer[3])/4;

temp\_t->C = (1475 - ((2475 \* temp\_t->AVG)) / 4096)/10;

temp\_t->F = ((temp\_t->C \* 9) + 160) / 5;

}

**Task 3:**

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

**#include** <stdbool.h>

**#include** <stdint.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/adc.h"

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

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

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

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

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

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

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

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

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

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

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

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

**#define** ADC\_SAMPLE\_BUF\_SIZE 1

**#if** defined(ewarm)

**#pragma** data\_alignment=1024

uint8\_t pui8ControlTable[1024];

**#elif** defined(ccs)

**#pragma** DATA\_ALIGN(pui8ControlTable, 1024)

uint8\_t pui8ControlTable[1024];

**#else**

uint8\_t pui8ControlTable[1024] **\_\_attribute\_\_** ((aligned(1024)));

**#endif**

//Structs

**enum** BUFFER\_STATUS

{

*EMPTY*,

*FULL*

};

**typedef** **struct**{

uint16\_t adcBuffer[ADC\_SAMPLE\_BUF\_SIZE];

uint32\_t AVG;

uint32\_t C;

uint32\_t F;

}Temp\_t;

**typedef** **enum**{

*WAIT*,

*RED*, *BLUE*, *GREEN*,

*red*, *blue*, *green*,

*TEMP*,

*temp*,

*STAT*

}State\_enum;

**typedef** **struct**{

**unsigned** **char** input;

}UART\_t;

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

//Variables

**volatile** **static** **enum** BUFFER\_STATUS BufferStatus;

**volatile** Temp\_t temp\_t;

**volatile** uint32\_t Count;

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

//Prototype

**void** **CalculateTemperatureAvg**(**void**);

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

**void** **ConfigureADC**(**void**);

**void** **ConfigureUDMA**(**void**);

**void** **check\_status**(uint32\_t status, **const** **char**\* string);

**void** **send\_status**(**const** **char**\* string, **bool** status);

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

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

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

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

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

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

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

**volatile** **unsigned** **char** status\_of\_leds;

**volatile** State\_enum state, nextState = *WAIT*;

**volatile** UART\_t uart\_t;

BufferStatus = *EMPTY*;

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

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

//Config LEDs

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

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

**GPIOPadConfigSet**(GPIO\_PORTF\_BASE, GPIO\_PIN\_1 | GPIO\_PIN\_2 | GPIO\_PIN\_3, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

ConfigureUART();

ConfigureUDMA();

ConfigureADC();

**IntMasterEnable**();

**while**(1)

{

**if**(state == *WAIT*)

{

**while**(!**UARTCharsAvail**(UART0\_BASE)){};

uart\_t.input = **UARTCharGet**(UART0\_BASE);

**if**(uart\_t.input == 'R') state = *RED*;

**else** **if**(uart\_t.input == 'B') state = *BLUE*;

**else** **if**(uart\_t.input == 'G') state = *GREEN*;

**else** **if**(uart\_t.input == 'r') state = *red*;

**else** **if**(uart\_t.input == 'b') state = *blue*;

**else** **if**(uart\_t.input == 'g') state = *green*;

**else** **if**(uart\_t.input == 'T') state = *TEMP*;

**else** **if**(uart\_t.input == 't') state = *temp*;

**else** **if**(uart\_t.input == 'S') state = *STAT*;

}

**else** **if**(state == *RED*){

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

}

**else** **if**(state == *red*){

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

}

**else** **if**(state == *BLUE*){

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

}

**else** **if**(state == *blue*){

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

}

**else** **if**(state == *GREEN*){

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

}

**else** **if**(state == *green*){

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

}

**else** **if**(state == *TEMP*){

calculate\_avg\_temperature(&temp\_t);

**UARTprintf**("C %3d\t", temp\_t.C);

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

}

**else** **if**(state == *temp*){

calculate\_avg\_temperature(&temp\_t);

**UARTprintf**("C %3d\t", temp\_t.F);

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

}

**else** **if**(state == *STAT*){

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

check\_status((status\_of\_leds&RED\_LED) >> 1, "Red: ");

check\_status((status\_of\_leds&BLUE\_LED) >> 2, "Blue: ");

check\_status((status\_of\_leds&GREEN\_LED) >> 3, "Green: ");

}

state = *WAIT*;

}

}

}

**void** **check\_status**(uint32\_t status, **const** **char**\* string){

**if**(status)

send\_status(string, 1);

**else**

send\_status(string, 0);

}

**void** **send\_status** (**const** **char**\* string, **bool** status){

**UARTprintf**(string);

**if**(status)

**UARTprintf**("ON");

**else**

**UARTprintf**("OFF");

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

}

**void** **CalculateTemperatureAvg**(**void**){

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

**while**(BufferStatus == *EMPTY*){};

temp\_t.AVG = 0;

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

{

temp\_t.AVG += temp\_t.adcBuffer[Count];

temp\_t.adcBuffer[Count] = 0;

}

BufferStatus = *EMPTY*;

uDMAChannelTransferSet(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT,

UDMA\_MODE\_BASIC,

(**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0),

&temp\_t.adcBuffer, ADC\_SAMPLE\_BUF\_SIZE);

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

temp\_t.AVG = ((temp\_t.AVG +

(ADC\_SAMPLE\_BUF\_SIZE / 2)) /

ADC\_SAMPLE\_BUF\_SIZE);

temp\_t.C = (1475 - ((2475 \* temp\_t.AVG)) / 4096)/10;

temp\_t.F = ((temp\_t.C \* 9) + 160) / 5;

}

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

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

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

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

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

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

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

}

**void** **ConfigureADC**(**void**){

**IntDisable**(INT\_ADC0SS0);

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

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

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

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

ADC\_CTL\_IE);

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

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

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

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

**IntEnable**(INT\_ADC0SS0);

}

**void** **ConfigureUDMA**(**void**){

**uDMAEnable**();

**uDMAControlBaseSet**(pui8ControlTable);

**uDMAChannelAttributeDisable**(UDMA\_CHANNEL\_ADC0,

UDMA\_ATTR\_ALTSELECT | UDMA\_ATTR\_HIGH\_PRIORITY |

UDMA\_ATTR\_REQMASK);

**uDMAChannelControlSet**(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT, UDMA\_SIZE\_16 |

UDMA\_SRC\_INC\_NONE | UDMA\_DST\_INC\_16 | UDMA\_ARB\_64);

uDMAChannelTransferSet(UDMA\_CHANNEL\_ADC0 | UDMA\_PRI\_SELECT,

UDMA\_MODE\_BASIC,

(**void** \*)(ADC0\_BASE + ADC\_O\_SSFIFO0),

&temp\_t.adcBuffer, ADC\_SAMPLE\_BUF\_SIZE);

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

**uDMAChannelEnable**(UDMA\_CHANNEL\_ADC0);

}

**void** **ADCSeq0Handler**(**void**){

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

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

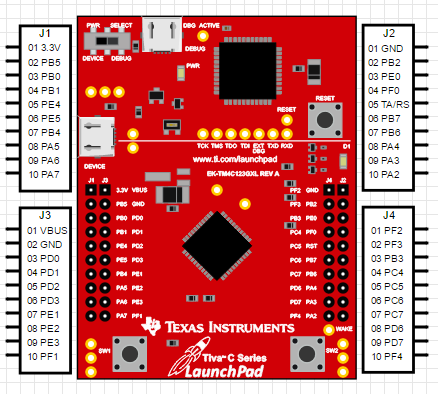
{

BufferStatus = *FULL*;

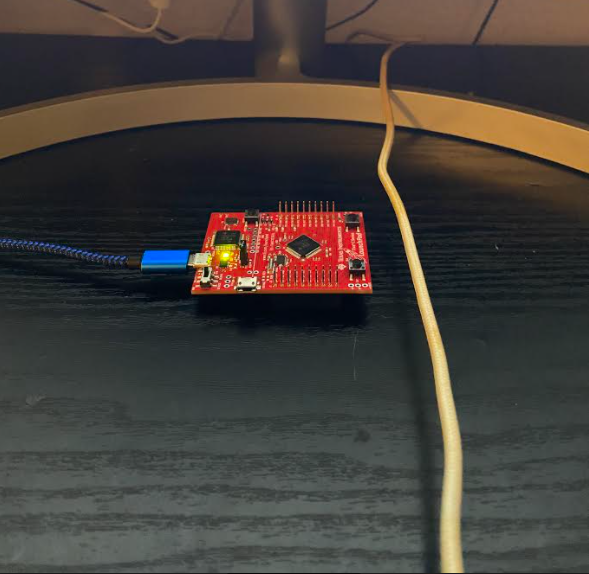
}

}

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

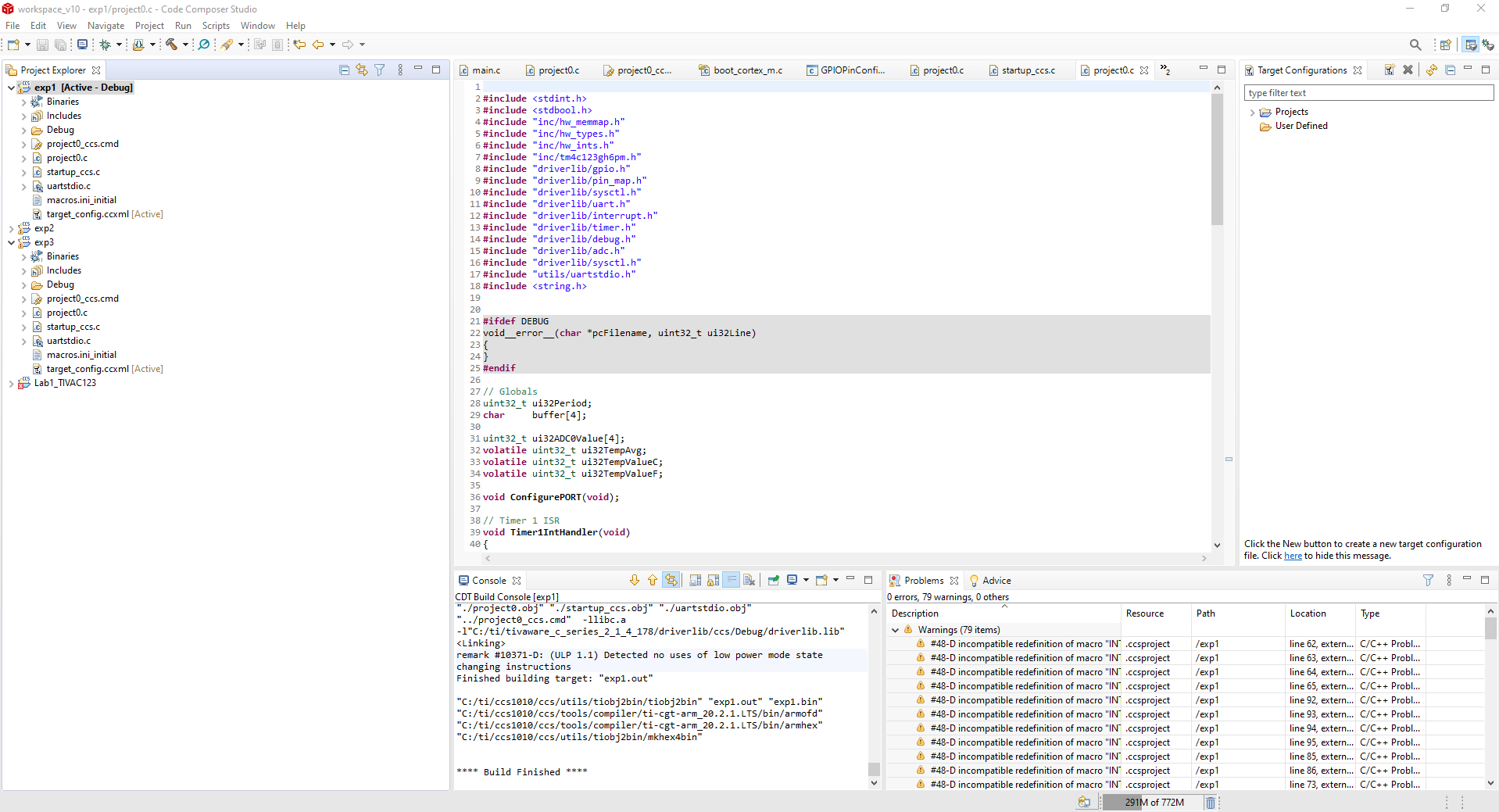


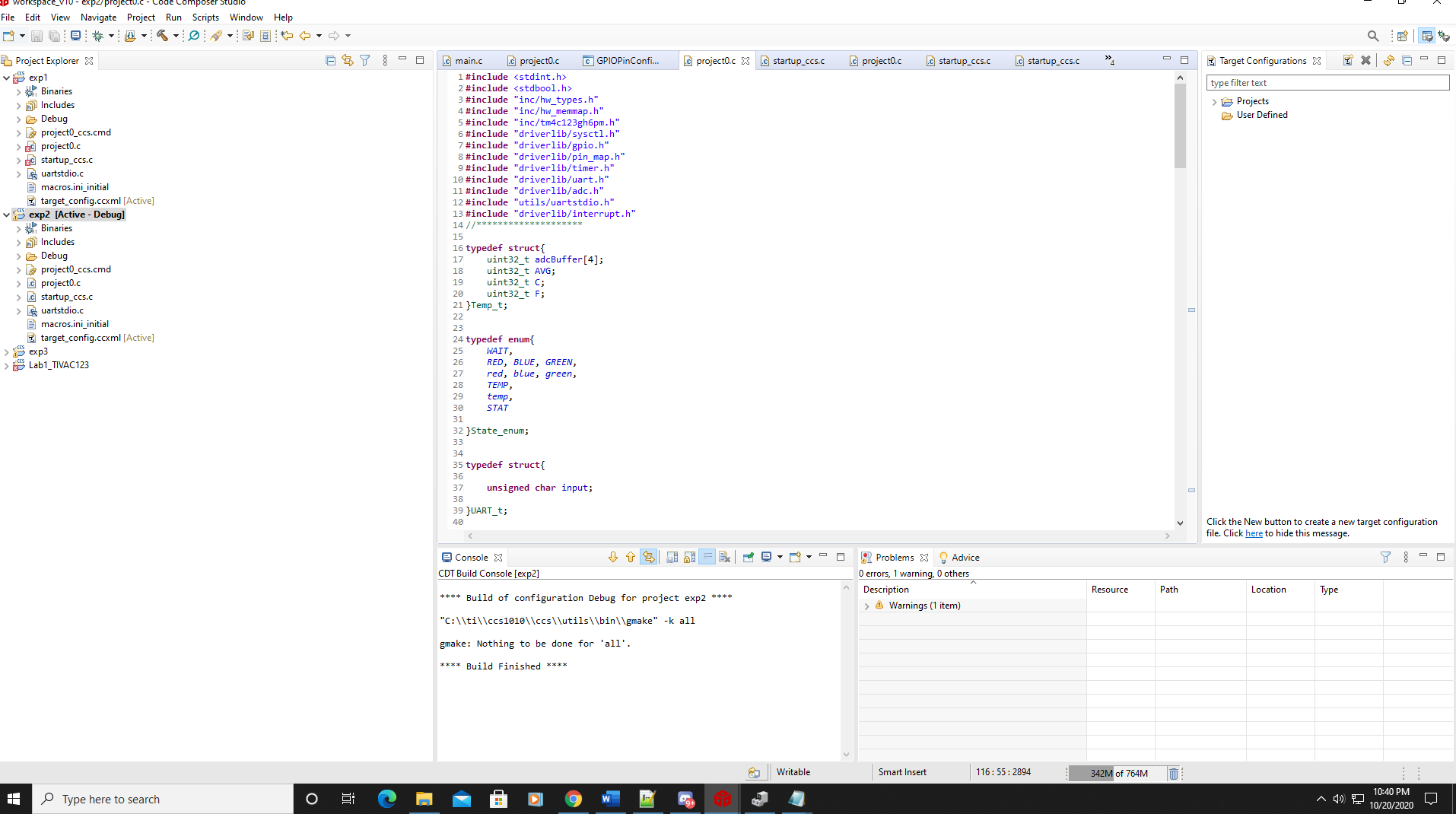
Set\_up:



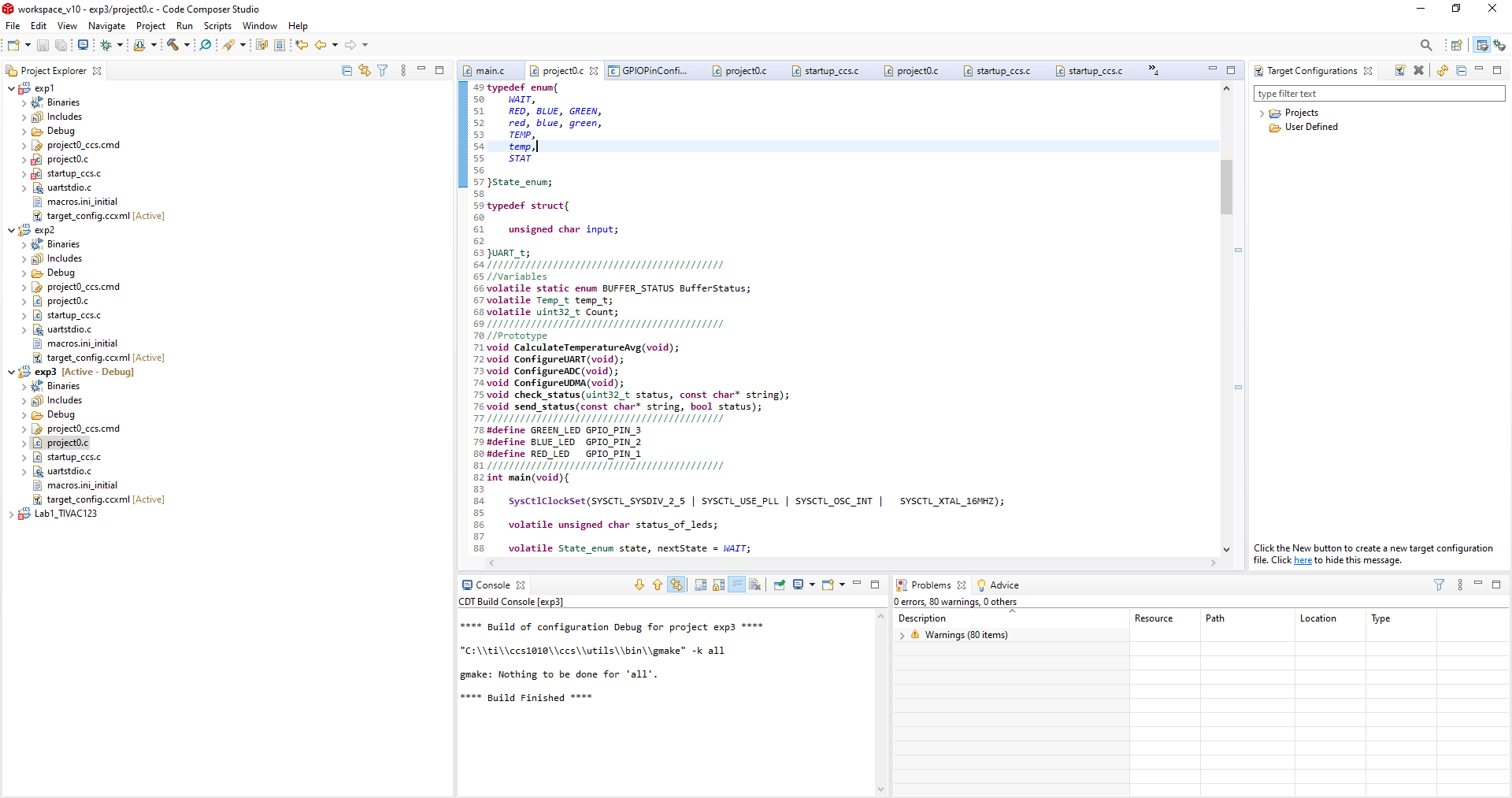
Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.

**Task1 compilation:**



**Task2 compilation:**

**Task3 compilation:**

****

1. Declaration

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

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

Name of the Student

Jeb Marinas