CPE403 – Advanced Embedded Systems

# Design Assignment 4

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.git>

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.

/\* XDC Module Headers \*/

**#include** <xdc/std.h>

**#include** <xdc/runtime/System.h>

**#include** <xdc/cfg/global.h>

/\* BIOS Module Headers \*/

**#include** <ti/sysbios/BIOS.h>

**#include** <ti/sysbios/knl/Task.h>

**#include** <ti/sysbios/knl/Clock.h>

**#include** <ti/sysbios/knl/Semaphore.h>

**#include** <ti/drivers/Board.h>

**#include** <ti/drivers/GPIO.h>

**#include** <ti/drivers/ADC.h>

**#include** <ti/display/Display.h>

**#include** <ti/drivers/PWM.h>

**#include** <ti/drivers/Timer.h>

/\* Driver configuration \*/

**#include** "ti\_drivers\_config.h"

uint16\_t timerCNT = 0;

uint16\_t adcValue;

ADC\_Handle adc;

**static** Display\_Handle display;

uint16\_t pwmPeriod = 4096;

uint16\_t duty = 0;

PWM\_Handle pwm1 = NULL;

**void** **timerCallback**(Timer\_Handle myHandle, int\_fast16\_t status)

{

timerCNT++;

**if**(timerCNT == 5)

{

Semaphore\_post(semHandle);

}

**if**(timerCNT == 10)

{

Semaphore\_post(semHandle1);

}

**if**(timerCNT == 15)

{

Semaphore\_post(semHandle2);

timerCNT = 0;

}

}

**void** **heartBeatFxN**(UArg arg0, UArg arg1)

{

**while**(1)

{

**GPIO\_write**(CONFIG\_GPIO\_LED\_0, CONFIG\_LED\_OFF);

Task\_sleep(500\*(1000/Clock\_tickPeriod));

**GPIO\_write**(CONFIG\_GPIO\_LED\_0, CONFIG\_LED\_ON);

Task\_sleep(500\*(1000/Clock\_tickPeriod));

}

}

**void** **adcFxN**(UArg arg0, UArg arg1)

{

**while**(1){

Semaphore\_pend(semHandle, BIOS\_WAIT\_FOREVER);

**ADC\_convert**(adc, &adcValue);

//Semaphore\_post(semHandle);

Task\_sleep(500\*(1000/Clock\_tickPeriod));

}

}

**void** **displayFxN**(UArg arg0, UArg arg1)

{

**while**(1){

Semaphore\_pend(semHandle1, BIOS\_WAIT\_FOREVER);

Display\_printf(display, 1, 0, "ADC Value: %d\n", adcValue);

// Semaphore\_post(semHandle1);

Task\_sleep(500\*(1000/Clock\_tickPeriod));

}

}

**void** **pwmFxN**(UArg arg0, UArg arg1)

{

**while**(1){

Semaphore\_pend(semHandle2, BIOS\_WAIT\_FOREVER);

duty = adcValue;

**PWM\_setDuty**(pwm1, duty);

Display\_printf(display, 1, 0, "PWM duty cycle: %d\n", duty);

//Semaphore\_post(semHandle);

Task\_sleep(500\*(1000/Clock\_tickPeriod));

}

}

/\*

\* ======== main ========

\*/

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

{

/\* Call driver init functions \*/

Board\_init();

**GPIO\_init**();

**ADC\_init**();

**PWM\_init**();

Display\_init();

**Timer\_init**();

Timer\_Handle timer0;

Timer\_Params timerParams;

/\*

\* Setting up the timer in continuous callback mode that calls the callback

\* function every 1000 microseconds or 1ms.

\*/

**Timer\_Params\_init**(&timerParams);

timerParams.period = 1000;

timerParams.periodUnits = *Timer\_PERIOD\_US*;

timerParams.timerMode = *Timer\_CONTINUOUS\_CALLBACK*;

timerParams.timerCallback = timerCallback;

timer0 = **Timer\_open**(CONFIG\_TIMER\_0, &timerParams);

**if** (timer0 == NULL) {

// Failed to initialized timer

Display\_printf(display, 0, 0, "Error initializing CONFIG\_TIMER\_0\n");

**while** (1);

}

**if** (**Timer\_start**(timer0) == Timer\_STATUS\_ERROR) {

// Failed to start timer

Display\_printf(display, 0, 0, "Error to start timer\n");

**while** (1);

}

ADC\_Params adcParams;

// int\_fast16\_t res;

**ADC\_Params\_init**(&adcParams);

adc = **ADC\_open**(CONFIG\_ADC\_0, &adcParams);

**if**(adc == NULL) {

Display\_printf(display, 0, 0, "Error initializing CONFIG\_ADC\_0\n");

**while**(1);

}

display = Display\_open(Display\_Type\_UART, NULL);

**if** (display == NULL) {

/\* Failed to open display driver \*/

**while** (1);

}

PWM\_Params pwmParams;

**PWM\_Params\_init**(&pwmParams);

pwmParams.dutyUnits = *PWM\_DUTY\_US*;

pwmParams.dutyValue = 0;

pwmParams.periodUnits = *PWM\_PERIOD\_US*;

pwmParams.periodValue = pwmPeriod;

pwm1 = **PWM\_open**(CONFIG\_PWM\_0, &pwmParams);

**if** (pwm1 == NULL) {

/\* CONFIG\_PWM\_0 did not open \*/

Display\_printf(display, 0, 0, "Error initializing CONFIG\_PWM\_0\n");

**while** (1);

}

**PWM\_start**(pwm1);

/\*

\* normal BIOS programs, would call BIOS\_start() to enable interrupts

\* and start the scheduler and kick BIOS into gear. But, this program

\* is a simple sanity test and calls BIOS\_exit() instead.

\*/

BIOS\_start(); /\* terminates program and dumps SysMin output \*/

**ADC\_close**(adc);

**return**(0);

}

/\*

\* ======== myDelay ========

\* Assembly function to delay. Decrements the count until it is zero

\* The exact duration depends on the processor speed.

\*/

**\_\_asm**(" .sect \".text:myDelay\"\n"

" .clink\n"

" .thumbfunc myDelay\n"

" .thumb\n"

" .global myDelay\n"

"myDelay:\n"

" subs r0, #1\n"

" bne.n myDelay\n"

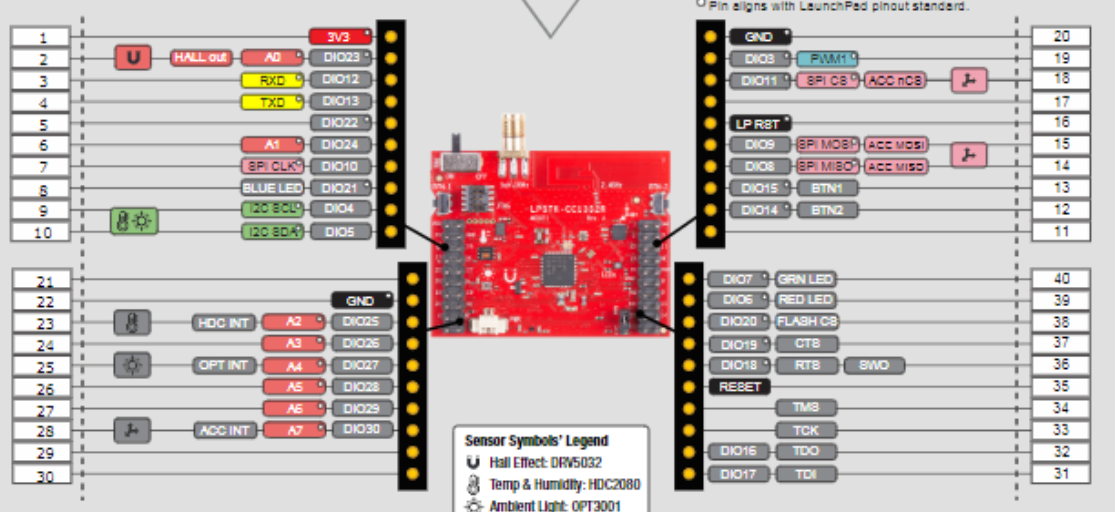
" bx lr\n");

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

Components used: CC1352R, BoosterPack MKII, jump wires, code composer studio.

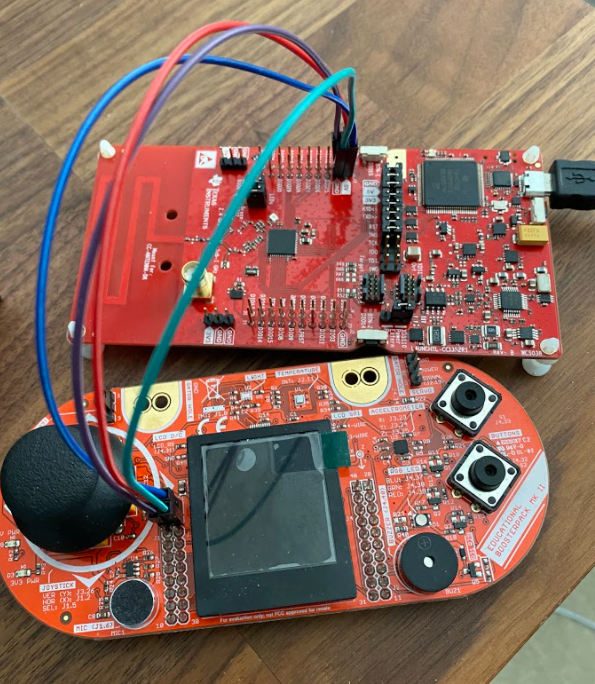
Pins used: 3.3V, GND, DIO23 (ADC), DIO7 (green LED), DIO6 (red LED), DIO12 and DIO13 (UART).

Interface: UART, ADC, PWM.

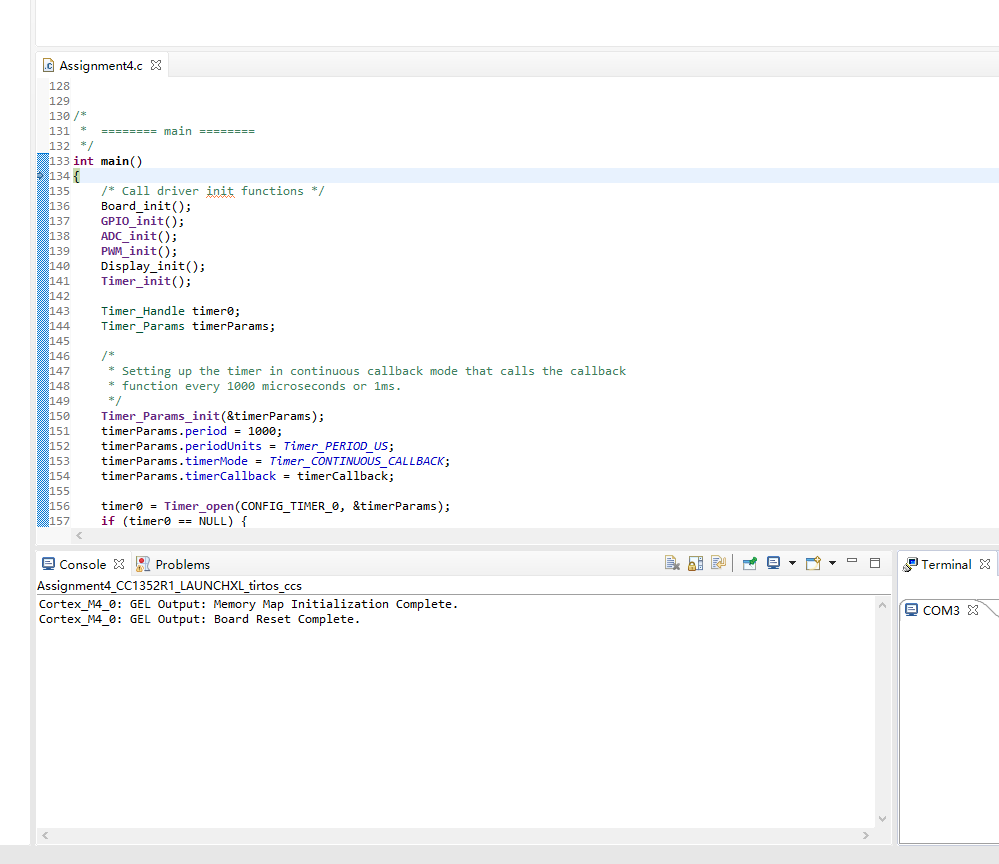


1. Screenshots of the IDE, physical setup, debugging process - Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.

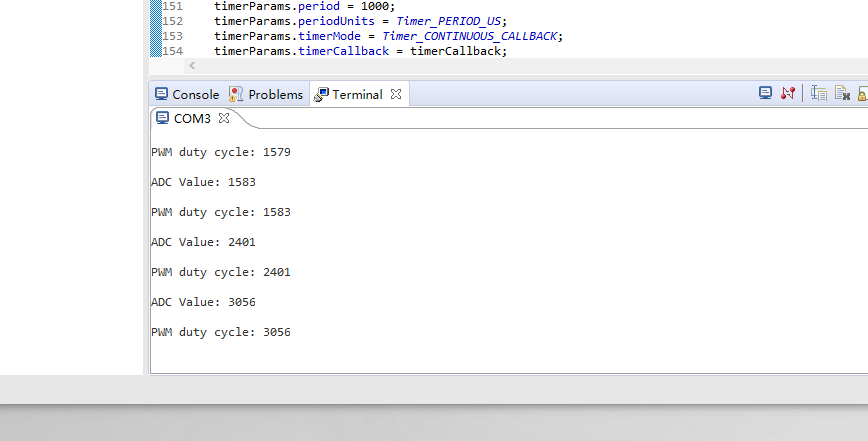
Board setup:

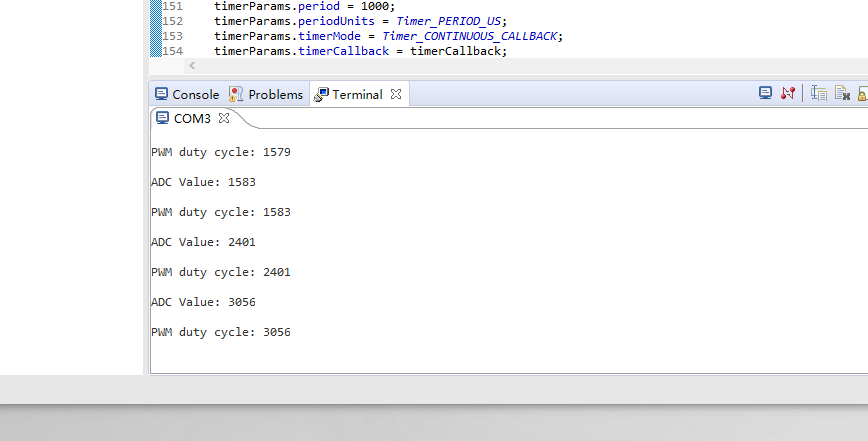


Successful compilation:



Output:





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