

Design Assignment 3

Name: Samuel McCormick

Email: samuel.mccormick@gmail.com

Github Repository link (root): <https://github.com/brokenboredom/tech-muffin>

Youtube Playlist link (root):

https://www.youtube.com/playlist?list=PLCSfNPhZUD_NZxU7Tsm_bMTwxiQqxqpBr

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 CPE403/603/710, Lastname, Firstname). Place all labs under the root folder MSP432E4/CC1352, 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 “ADVEMBSYS”. 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.

I understand the Student Academic Misconduct Policy -

<http://studentconduct.unlv.edu/misconduct/policy.html>

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

Samuel McCormick

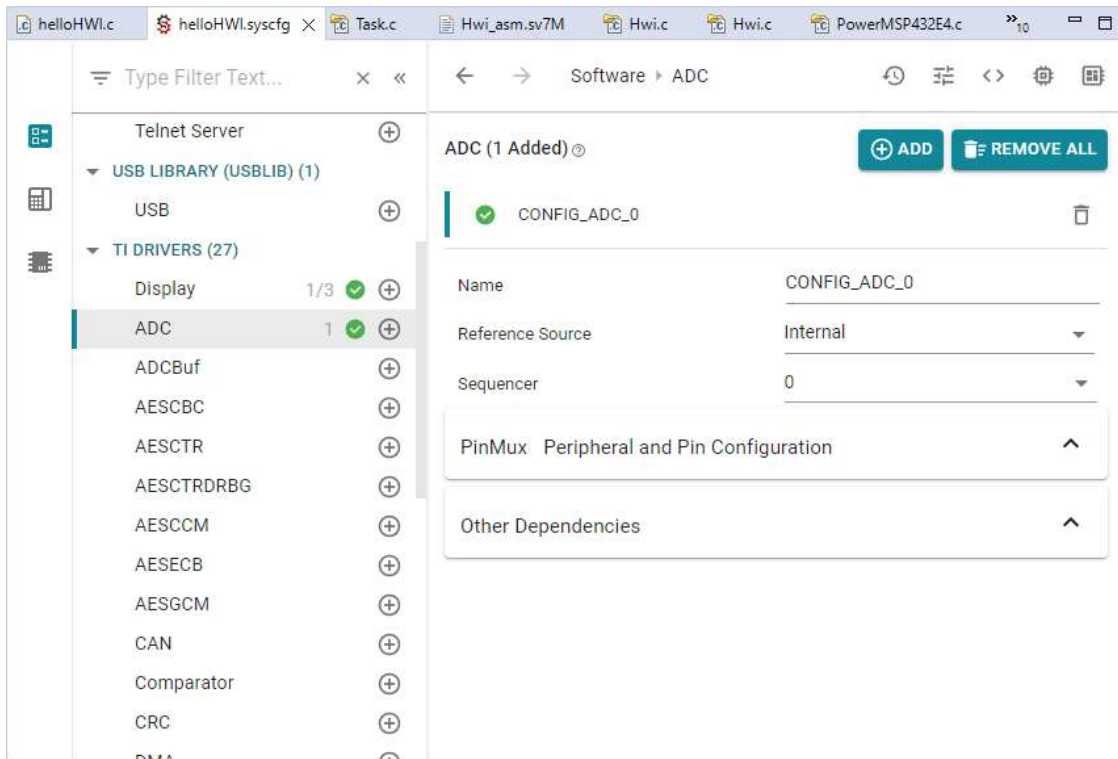
Follow the submission guideline to be awarded points for this Assignment.

Goal of this assignment is to create four tasks, 1) ADC task, 2) UART display task, 3) Switch Read task, 4) PWM task, and 4) Heartbeat function (PF4). The heartbeat function is performed throughout the execution of the program (not idle task). Each task will be executed in order specified above every 15 ms. Connect a potentiometer/joystick (one axis) to the ADC pin. Use ADC0-CH0. Also initialize a PWM signal to a LED (PF0). Initial value of the PWM duty cycle is set to 0. Create a timer/clock for every 1 ms, at 5th instance of clock the task ADC is performed, at 10th instance of clock the task UART displays the current value ADC in the terminal. The third task performs PWM update at the 15ms. The Switch Read task is triggered by HWI on the switch SW1/SW2. This updates the current /stored value of ADC as the duty cycle based of the PWM. Note that the duty cycle of the PWM does not change unless the switch is pressed, even when the ADC value changes. However, the UART should display the dynamic value of the ADC. Create a semaphore to perform resource sharing/IPC between ADC and UART (so that the UART displays the ADC values after a valid conversion). Implement a visual heartbeat function on PF4 with a suitable period. The period can be based of a timer/clock. Configure the hardware using sysconfig. Show the execution results using ROV and execution graphs. Verify the change of PWM duty cycle using the logic analyzer.

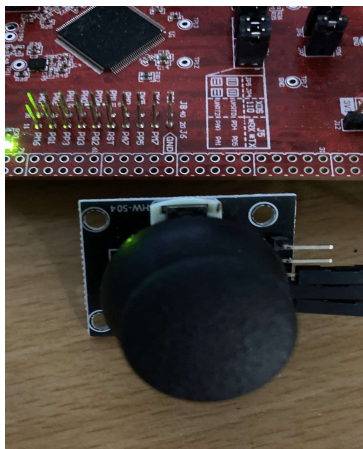
Task 1

This was the most difficult task to implement because it was the first task and all the examples used pthreads– something not required for this assignment. After gutting the pthreads examples for the adc configuration needed to run off a manual trigger the semaphore would not post from within the sysClock-driven Clock task and continued to throw an error during execution. By changing to a Timer task I was able to post the semaphore from within the timer and could move on to the subsequent tasks.

I used the *adcsinglechannel* TI-RTOS example to base my configuration off of and rearranged the variables, instantiation, and conversions as needed. The ADC is opened in *main()* and the *adcFxn* task created there as well. The task waits for the semaphore from the clock to post and ensures there are resources available, begins the conversion, then posts the adc completion semaphore before waiting again.



ADC sysconfig.



Joystick used in ADC

```

Void adcFxn(UArg arg0, UArg arg1)
{
    for (;;) {
        if (Semaphore_getCount(sem0Handle) == 0) {
            System_printf("Sem blocked in task1\n");
        }
        /* Get access to adc resource */
        Semaphore_pend(sem0Handle,
            BIOS_WAIT_FOREVER);

        /* Blocking mode conversion */
        res = ADC_convert(adc, &adcValue0);
        if (res == ADC_STATUS_SUCCESS) {
        }
        else {
            System_printf("CONFIG_ADC_0 convert
failed\n");
        }

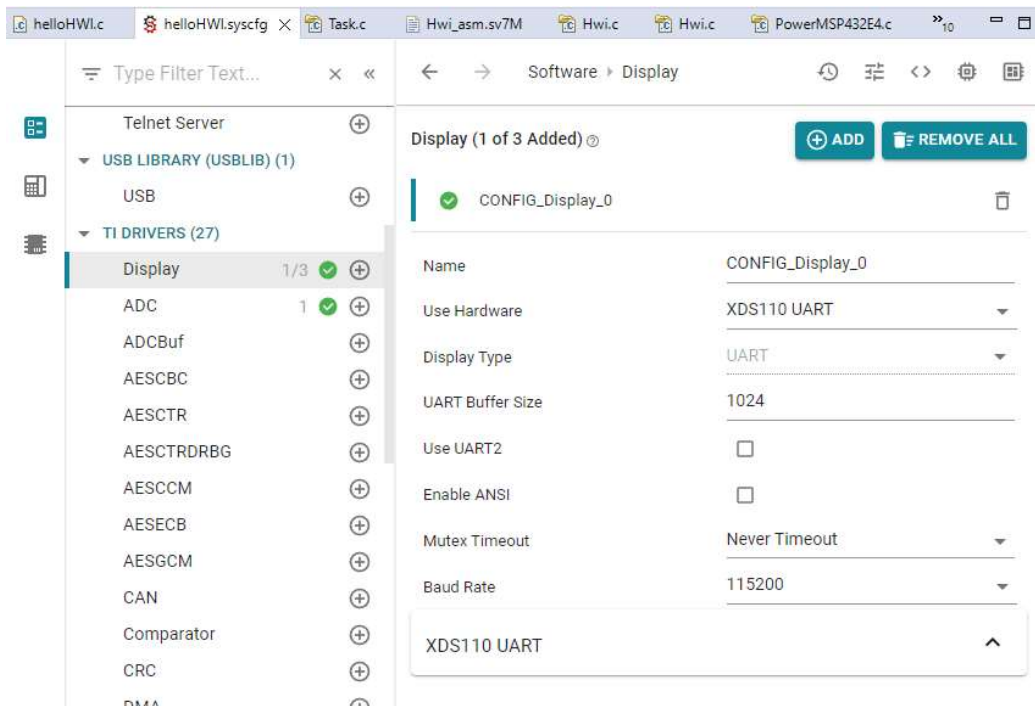
        Semaphore_post(sem0Handle);
    }
}

```

Task 2

This task also caused me problems because the board was crashing every time I added a display driver. I ended up using the simplest implementation from *buttonled* on the github with *display_print0()* calls to begin with. After getting the display to work I found that printing anything from the HWI function was crashing the board and I refactored everything to *display_printf()* once the crashes were fixed.

The UART is implemented automatically through the display sysconfig when properly setup.



Display sysconfig using XDS110 UART.



ESP-01 UART module.

```
Void uartFxn(UArg arg0, UArg arg1)
{
    for (;;) {
        if (Semaphore_getCount(sem1Handle) == 0) {
            System_printf("Sem blocked in task2\n");
        }
        Semaphore_pend(sem1Handle, BIOS_WAIT_FOREVER);

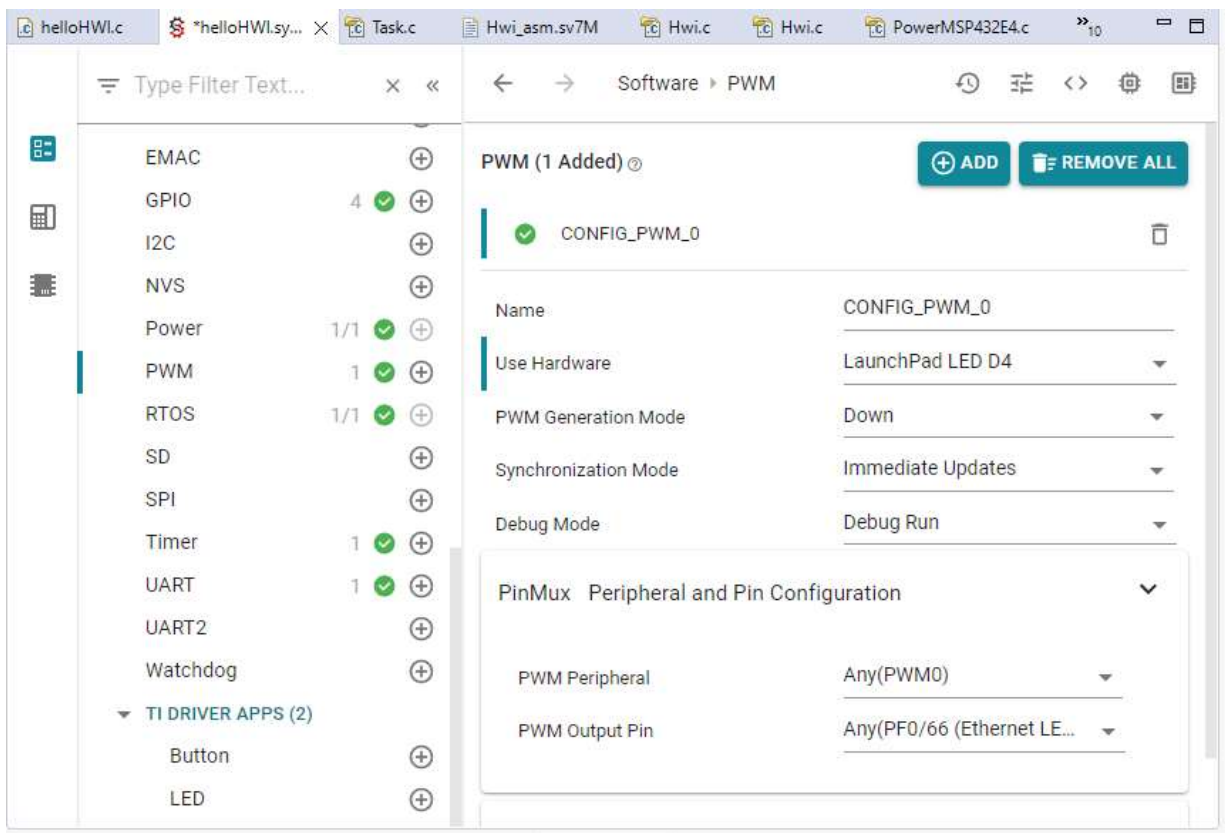
        if (Semaphore_getCount(sem0Handle) == 0) {
            System_printf("Sem blocked in task2\n");
        }
        Semaphore_pend(sem0Handle, BIOS_WAIT_FOREVER);

        //Print adc conversion
        Display_printf(display, 0, 0, "CONFIG_ADC_0 raw result: %d\n",
            adcValue0);
    }
}
```

Task 4

This was the PWM task and following the same methods used to gut the pthreads examples for the ADC I implemented the PWM using the *pwmled1* TI-RTOS example. I used a period of 4000us to mirror the 250Hz from earlier assignments and a duty cycle calculated from the joystick input.

This task also required the use of the LogicAnalyzer to show the changing duty cycle. To do this is altered the sysconfig for the PWM module and output through PF1 instead of PF0 (LED4) to capture the signal. When running in the video I used PF0 to visually show the changes on the LED when the HWI triggers.



PWM sysconfig set for LED4 output.



PWM duty cycle changing with the joystick and HWI.

```
uint16_t pwmPeriod = 4000;
...
int main() {
...
    /* Configure the PWM with period of 4000 us
    (250Hz)*/
    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 */
        while (1);
    }
    PWM_start(pwm1);
...
}
```

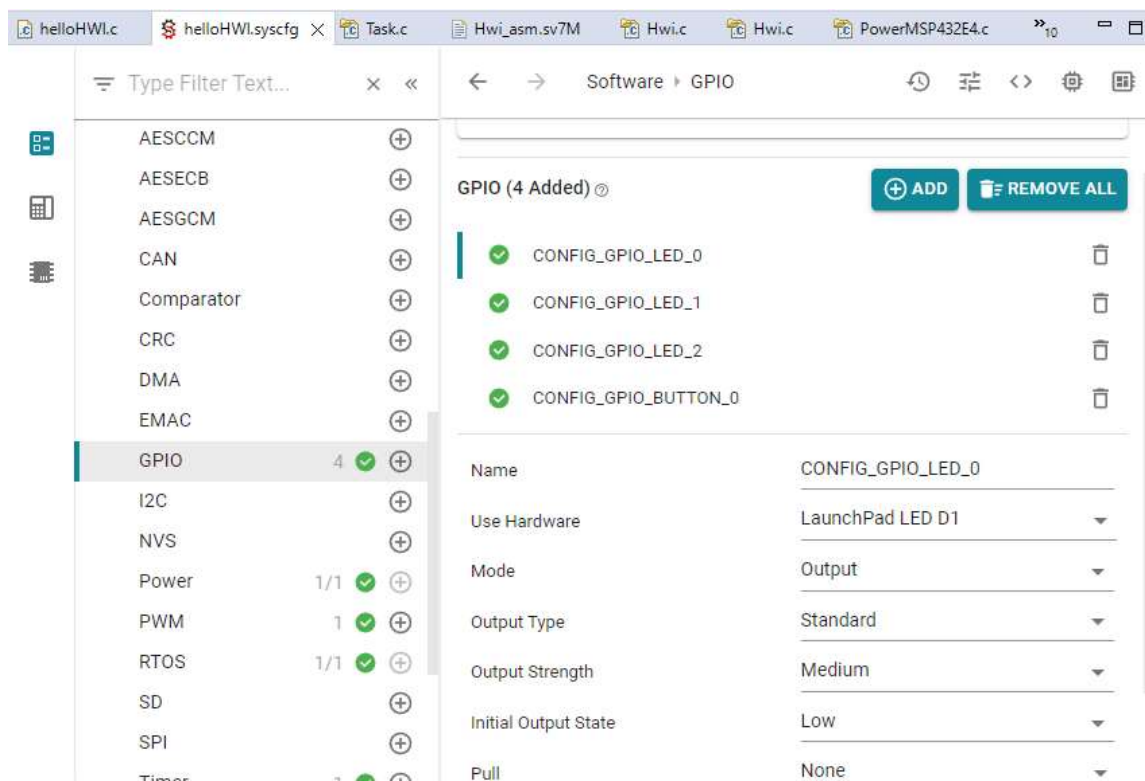
```
Void pwmFxn(UArg arg0, UArg arg1)
{
    for(;;)
    {
        if (Semaphore_getCount(sem0Handle) == 0) {
            System_printf("Sem blocked in task3\n");
        }
        Semaphore_pend(sem0Handle,
            BIOS_WAIT_FOREVER);

        //PWM duty based off of joystick
        duty = pwmPeriod * adcValue0/41 / 100;
    }
}
```

Task 3

The Switch/Read task was sourced from the github *mutex2* example. I left the LED functionality in for debugging purposes and added *PWM_setduty()* to the service routine *hwiFxn()* when the IRQ was functional.

```
Void hwiFxn(UArg arg){  
    CLEAR_INTERRUPT();  
  
    //Apply pwm duty cycle  
    PWM_setDuty(pwm1, duty);  
}
```

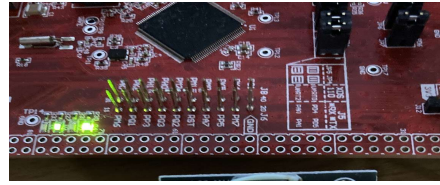


GPIO sysconfig with all debug LEDs and HWI BUTTON.

Task 5

This was the heartbeat function. I implemented it using the systemClock and LED1. A continuous clock was created with a period of 500ms which toggles the led.

```
Void heartbeatFxn(UArg arg0)
{
    GPIO_toggle(CONFIG_GPIO_LED_1);
}
...
int main() {
    ...
    Clock_Params_init(&clkParams);
    clkParams.period = 500000/Clock_tickPeriod;
    clkParams.startFlag = TRUE;
    /* Construct a periodic Clock Instance */
    Clock_construct(&clk0Struct, (Clock_FuncPtr)heartbeatFxn,
        500000/Clock_tickPeriod, &clkParams);
    ...
}
```

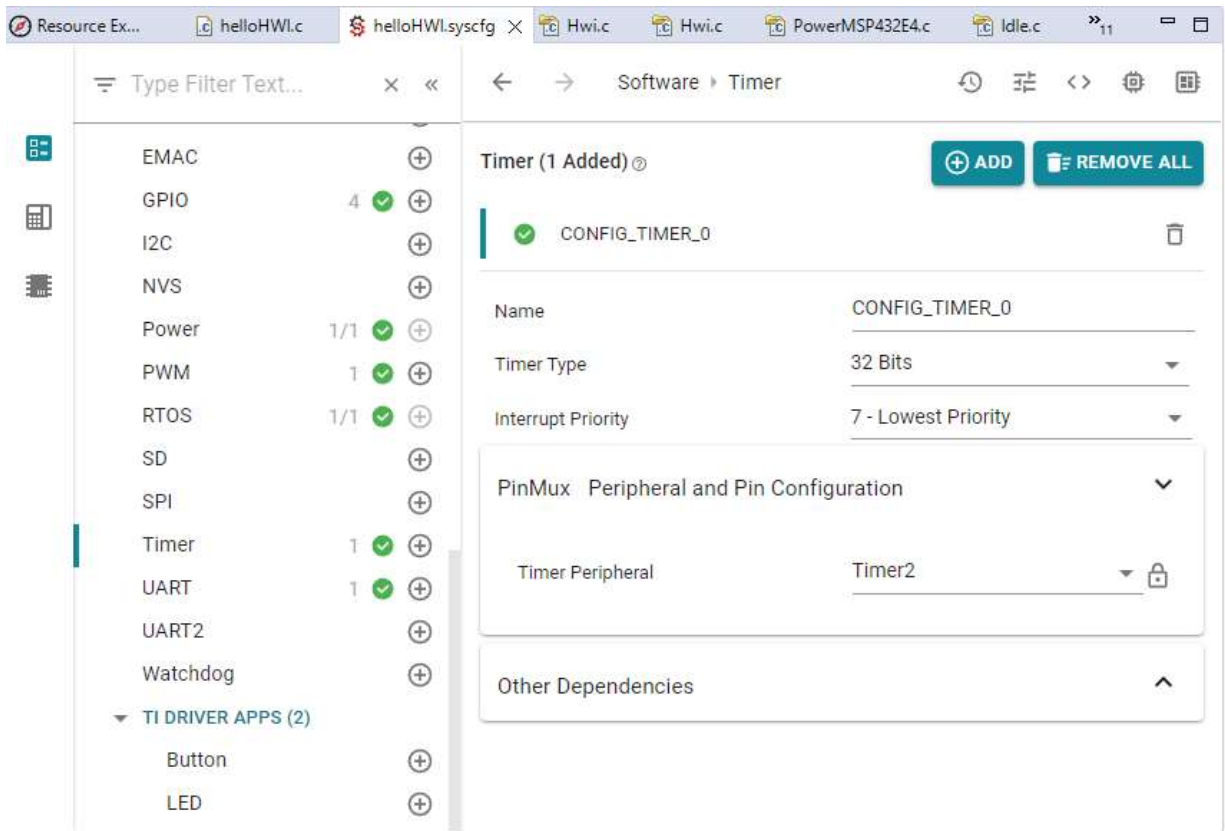


LED1 and LED4 shown.

main.cfg file showing Clock configuration:

```
/* ===== Clock configuration
===== */
var Clock = xdc.useModule('ti.sysbios.knl.Clock');
/*
 * Default value is family dependent. For example,
 * Linux systems often only
 * support a minimum period of 10000 us and
 * multiples of 10000 us.
 * TI platforms have a default of 1000 us.
 */
Clock.tickPeriod = 1000;
```


Additionally it was required to run the three main tasks off of a clock every 5ms. I used a timer task with a period of 5ms and an instance counter to track 5/10/15ms. The timer posted the semaphores for the adc, uart, and pwm tasks.

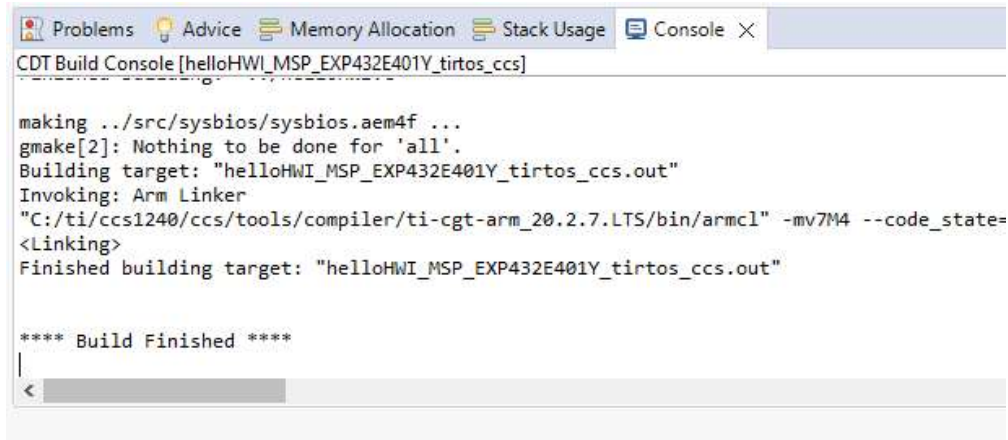


Timer sysconfig using Timer2 because clock uses Timer1.

```
int main() {
...
Timer_Params_init(&params);
params.period = 5000;
params.periodUnits = Timer_PERIOD_US;
params.timerMode = Timer_CONTINUOUS_CALLBACK;
params.timerCallback = timerCallback;
timer0 = Timer_open(CONFIG_TIMER_0, &params);
if (timer0 == NULL) {
    /* Failed to initialize timer */
    while (1) {}
}
if (Timer_start(timer0) == Timer_STATUS_ERROR) {
    /* Failed to start timer */
    while (1) {}
}
...
}
```

```
void timerCallback(Timer_Handle myHandle,
int_fast16_t status)
{
    //Increment instance counter
    clk_instance_cnt++;

    //Trigger adc conversion at 5ms
    if (clk_instance_cnt == 1) {
        Semaphore_post(sem0Handle);
    }
    //Trigger uart display at 10ms
    else if (clk_instance_cnt == 2) {
        Semaphore_post(sem1Handle);
    }
    //Calculate new pwi duty
    else if (clk_instance_cnt == 3) {
        Semaphore_post(sem0Handle);
        clk_instance_cnt = 0;
    }
    else {} // Never happens
}
```

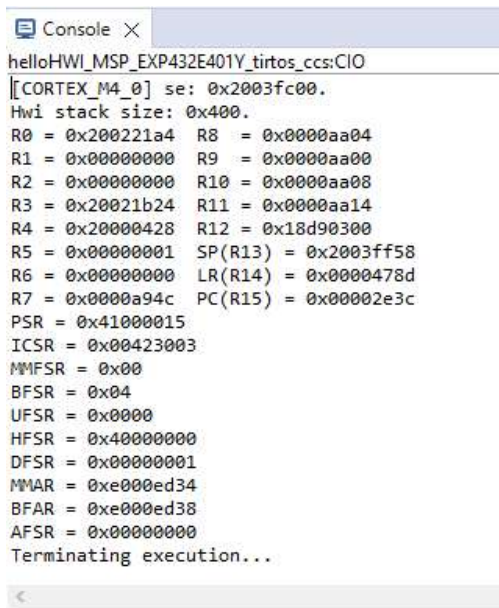


```
Problems Advice Memory Allocation Stack Usage Console X
CDT Build Console [helloHWI_MSP_EXP432E401Y_tirtos_ccs]

making ../src/sysbios/sysbios.aem4f ...
gmake[2]: Nothing to be done for 'all'.
Building target: "helloHWI_MSP_EXP432E401Y_tirtos_ccs.out"
Invoking: Arm Linker
"C:/ti/ccs1240/ccs/tools/compiler/ti-cgt-arm_20.2.7.LTS/bin/armcl" -mv7M4 --code_state=
<Linking>
Finished building target: "helloHWI_MSP_EXP432E401Y_tirtos_ccs.out"

**** Build Finished ****
```

Compiled project.



```
Console X
helloHWI_MSP_EXP432E401Y_tirtos_ccs:CIO
[CORTEX_M4_0] se: 0x2003fc00.
Hwi stack size: 0x400.
R0 = 0x200221a4 R8 = 0x0000aa04
R1 = 0x00000000 R9 = 0x0000aa00
R2 = 0x00000000 R10 = 0x0000aa08
R3 = 0x20021b24 R11 = 0x0000aa14
R4 = 0x20000428 R12 = 0x18d90300
R5 = 0x00000001 SP(R13) = 0x2003ff58
R6 = 0x00000000 LR(R14) = 0x0000478d
R7 = 0x0000a94c PC(R15) = 0x00002e3c
PSR = 0x41000015
ICSR = 0x00423003
MMFSR = 0x00
BFSR = 0x04
UFSR = 0x0000
HFSR = 0x40000000
DFSR = 0x00000001
MMAR = 0xe00ed34
BFAR = 0xe00ed38
AFSR = 0x00000000
Terminating execution...
```

Crash from adding display shows it happening in the HWI.

Youtube Playlist:

https://www.youtube.com/playlist?list=PLCSfNPhZUD_NZxU7Tsm_bMTwxiQqxqpBr