CPE403 – Advanced Embedded Systems

Design Assignment 1

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

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.
1. Code for tasks: For each task, submit the relevant modified/section 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 initialization

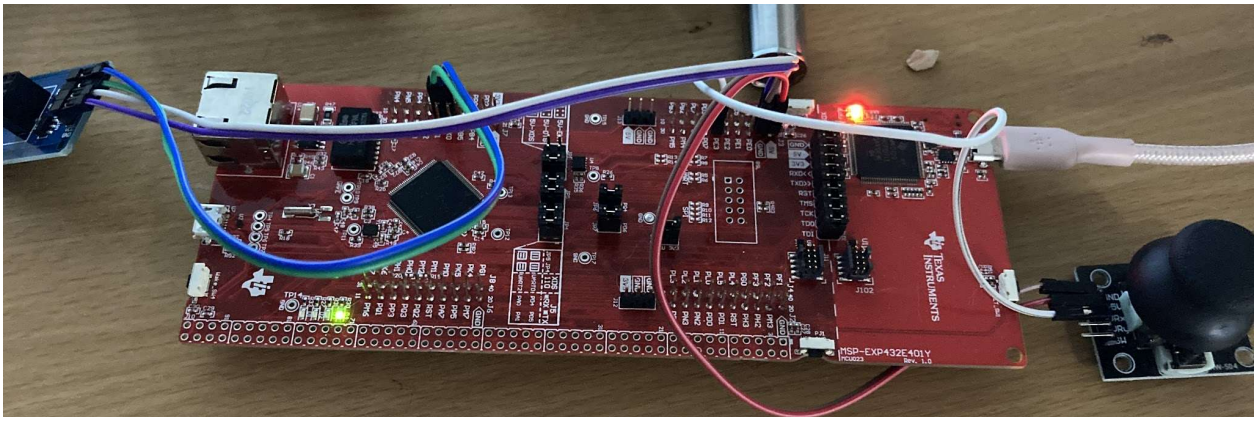
and execution section of each task separately. Use a separate page for each task.

2. Block diagram and/or Schematics showing the components, pins used, and interface. You can use KiCAD/Eagle/Altium to get the schematics. KiCAD Symbol libraries for TI uCs are @https://kicad.github.io/symbols/MCU_Texas or https://bityl.co/LJKN or https://github.com/mik4el/cc1352-swim-thermo or https://github.com/terjeio/CNC_Boosterpack
3. Screenshots of the IDE, physical setup, debugging process – Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.
4. Declaration

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

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

Samuel McCormick

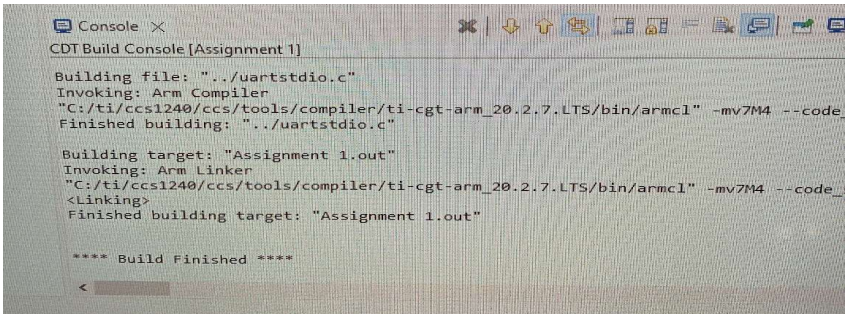


➔ Task 1) ADC

The project was started using example code from CCS resource explorer, namely the adc_singleended_multichannel_timertrigger example. From that code we added the DMA request and eventually the hardware averaging. The only altered code from the examples is shown on the right.

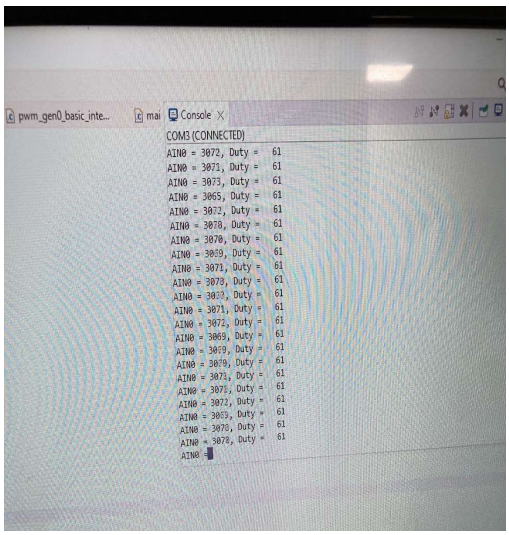
The altered example code for single channel and additional hardware averaging

```
/* Configure PE3 as ADC input channel */
   MAP_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);

   /* Enable the clock to ADC-0 and wait for it to be ready */
   MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
   while(!(MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_ADC0)))
   {
   }

   /* Configure HW Averaging of 32x */
   MAP_ADCHardwareOversampleConfigure(ADC0_BASE, 32);

   /* Configure Sequencer 2 to sample the analog channel : AIN0-AIN3.
The
    * end of conversion and interrupt generation is set for AIN3 */
   MAP_ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_CH0);
   MAP_ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_CH0);
   MAP_ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_CH0);
   MAP_ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_CH0 |
ADC_CTL_IE |  ADC_CTL_END);
```



➔ Task 2) UART

The UART was set up using the same example code as the ADC. We used a 120MHz system clock and output the DMA buffered, hardware averaged results from the ADC, namely srcBuffer[0]. The output line in main was the only altered code from the example and includes additional output for the calculated Duty Cycle value used in later tasks.

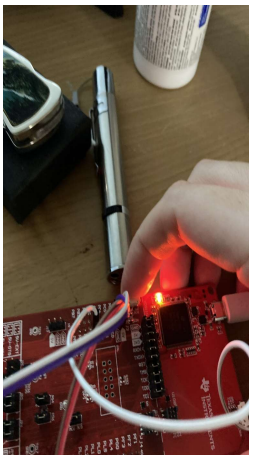


Altered UART code from the main while() loop that outputs the dynamic ADC value and rough Duty Cycle.

```
while(1)
      {
          /* Display the AIN0 (PE3) digital value on the
console. */
          UARTprintf("AIN0 = %4d, Duty = %4d\n",
srcBuffer[0], srcBuffer[0]/50);

      }
```

➔ Task 3) Switch Interrupt

The switch interrupt task was task 3 but last to be added to the project. Using the example code provided the interrupt service routine and feedback led were changed. LED2 was already in use by the Heartbeat so it was changed to LED1 to provide simple debugging feedback for switch presses. The interrupt routine was altered to recalculate the PWM duty cycle when entered.

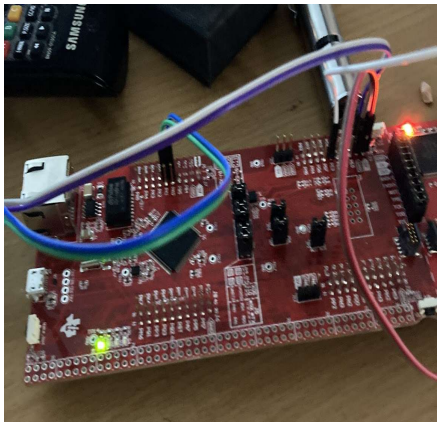


Altered switch code to calculate duty cycle and change to LED1(PN1)

```
void GPIOJ_IRQHandler(void)
{
  … examplecode …

      /* Set our duty cycle based off of ADC buffer
      * Formula for duty cycle with adc values between
      * 0-4000 (my joystick limit): PWMPeriod *
(srcBuffer/x) / 100 = 0-100%
      * srcBuffer[0]/50 would be 0-81 so we'll use that */
      MAP_PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0,
       MAP_PWMGenPeriodGet(PWM0_BASE,
PWM_GEN_0) * (srcBuffer[0]/50) / 100 );



      /* Toggle the LED for feedback*/
      MAP_GPIOPinWrite(GPIO_PORTN_BASE,
GPIO_PIN_1,

~(MAP_GPIOPinRead(GPIO_PORTN_BASE,
GPIO_PIN_1)));
   }
}

… inside main() …
/* Configure the GPIO PN0 as output */
   MAP_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE,
GPIO_PIN_1);
   MAP_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1,
0);
```

➔ Task 4) PWM Generation

The PWM generation task was set up after the ADC and UART tasks. Using the example code from pwm_gen0_basic_interrupt we removed PF1 from the setup and altered the duty cycle calculation for PF0.

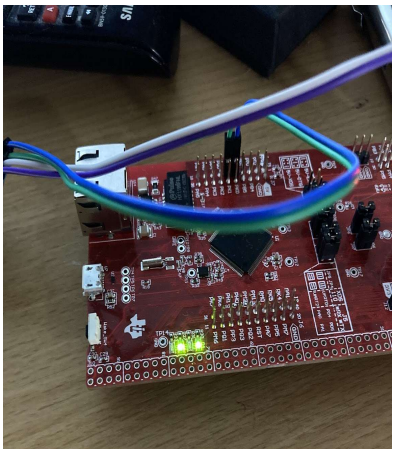


Altered PWM example code to only use PF0 with a duty cycle of 0 initially. DC recalculated in switch task above.

```
/* Set the PWM period to 250Hz.  To calculate the
appropriate parameter
    * use the following equation: N = (1 / f) * SysClk.
    * In this case you get: (1 / 250Hz) * 120MHz =
480000 cycles. */
   MAP_PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0,
480000);

   // We alter this in Switch interrupt handler
   // With an initial duty cycle of 0
   MAP_PWMPulseWidthSet(PWM0_BASE,
PWM_OUT_0, 0);
```

➔ Task 5) Heartbeat

The heartbeat task was taken from the watchdog example code. The switch button functionality was removed but the code is otherwise unchanged from the example.



No code was altered, only removed.

```
void
WATCHDOG_IRQHandler(void)
{
   // If we have been told to stop feeding the watchdog,
return immediately
   // without clearing the interrupt.  This will cause the
system to reset
   // next time the watchdog interrupt fires.
   if(!g_bFeedWatchdog)
   {
      return;
   }

   // Clear the watchdog interrupt
   MAP_WatchdogIntClear(WATCHDOG0_BASE);

   // Invert the GPIO PN0 value.
   MAP_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0,
            (MAP_GPIOPinRead(GPIO_PORTN_BASE,
                  GPIO_PIN_0) ^ GPIO_PIN_0));
}
```