CPE403/ECG603/ECG710 – Advanced Embedded Systems/Real-Time Embedded Systems

Design Assignment 1

DO NOT REMOVE THIS PAGE DURING SUBMISSION:

Name: Joshua Martinez

Email: marti87@unlv.nevada.edu

Github Repository link (root): https://github.com/JoshuaMa2003/submission_da_0011

Youtube Playlist link (root):
https://www.youtube.com/playlist?list=PLZ09MA_uFt61Vw2JCAtw-_8vAdmoCKXpZ

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 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 videos as a playlist under the name “ADVEMBSYS”. The playlist should

have the video sequence arranged as submission or due dates.

7. Only submit the PDF. Upload this document in the github repository and in canvas.

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.**

Task 1 – ADC Sampling with Timer Trigger DMA

- **Original Code (Template)**

```
/* 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_CH1);
MAP_ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_CH2);
MAP_ADCSequenceStepConfigure(ADC0_BASE, 2, 3, ADC_CTL_CH3 | ADC_CTL_IE |
                             ADC_CTL_END);

/* Enable sample sequence 2 with a timer signal trigger.  Sequencer 2
 * will do a single sample when the timer generates a trigger on timeout*/
MAP_ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_TIMER, 2);
```

- **Modified Code (Joystick X-Axis Only on PE3/AIN0)**

```
MAP_ADCSequenceConfigure(JOY_ADC_BASE, JOY_ADC_SEQ, ADC_TRIGGER_TIMER, 2);
MAP_ADCSequenceStepConfigure(JOY_ADC_BASE, JOY_ADC_SEQ, 0,
                             JOY_ADC_CH | ADC_CTL_IE | ADC_CTL_END);
MAP_ADCIntClearEx(JOY_ADC_BASE, ADC_INT_DMA_SS2);
MAP_ADCIntEnableEx(JOY_ADC_BASE, ADC_INT_DMA_SS2);
MAP_ADCSequenceDMAEnable(JOY_ADC_BASE, JOY_ADC_SEQ);
MAP_ADCSequenceEnable(JOY_ADC_BASE, JOY_ADC_SEQ);
MAP_IntEnable(INT_ADC0SS2);
```

Description: In the original TI example, the ADC was configured to sample four channels (AIN0-AIN3) simultaneously and output all results through UART. For this application, I modified the ADC configuration to sample only one analog channel – **PE3 (AIN0)** – which corresponds to the joystick's X-axis output. I also enabled **64x hardware oversampling** to improve signal stability and accuracy and used **Timer2A** to generate a periodic trigger at 1 kHz. The ADC results are transferred automatically using **DMA** rather than polling, which offloads the CPU and ensures consistent sampling intervals. This modification was necessary to measure joystick position in real time with minimal processor overhead.

Task 2 – PWM Output (PF0/D4 Active-LOW LED)

- **Original Code (No PWM present):**
  The source code contained no PWM or LED code; it focused only on ADC and DMA
- **Added Code (PWM Initialization and Duty Function):**

```
#define PWM_GPIO_BASE   GPIO_PORTF_BASE
#define PWM_GPIO_PIN    GPIO_PIN_0              // PF0 -> D4 (active-LOW)
#define PWM_GPIO_CFG    GPIO_PF0_M0PWM0
#define PWM_BASE_M      PWM0_BASE
#define PWM_GEN         PWM_GEN_0
#define PWM_OUT         PWM_OUT_0
#define PWM_OUT_BIT     PWM_OUT_0_BIT
#define PWM_FREQ_HZ     1000
```

```
static void initPWM_PF0(uint32_t sysclk)
{
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF)) {}
    MAP_GPIOPinConfigure(PWM_GPIO_CFG);
    MAP_GPIOPinTypePWM(PWM_GPIO_BASE, PWM_GPIO_PIN);

    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
    while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_PWM0)) {}

    MAP_PWMClockSet(PWM_BASE_M, PWM_SYSCLK_DIV_64);
    uint32_t pwmClk = sysclk / 64U;
    uint32_t period = (pwmClk / PWM_FREQ_HZ) - 1U;

    MAP_PWMGenConfigure(PWM_BASE_M, PWM_GEN, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
    MAP_PWMGenPeriodSet(PWM_BASE_M, PWM_GEN, period);
    MAP_PWMPulseWidthSet(PWM_BASE_M, PWM_OUT, 1U);   // start LED bright (leftmost)
    MAP_PWMGenEnable(PWM_BASE_M, PWM_GEN);
    MAP_PWMOutputState(PWM_BASE_M, PWM_OUT_BIT, true);
}
```

Description: The base example did not include any PWM functionality, so a new PWM module was added to control the LED on **PF0 (D4)**. I configured **PWM0_OUT0** with a frequency of **1 kHz** and used the joystick's ADC value to determine the PWM duty cycle. This task demonstrates analog-to-digital-to-PWM conversion, where the joystick's analog position directly influences LED brightness through hardware-timed pulse-width modulation.

Task 3 – UART Serial Output

- **Original Code (Template):**

```
46          /* Display the AIN0-AIN03 (PE3-PE0) digital value on the console. */
47          UARTprintf("AIN0-3 = %4d %4d %4d %4d\r", srcBuffer[0],
48                     srcBuffer[1], srcBuffer[2], srcBuffer[3]);
49      }
50  }
```

- **Modified Code (Single ADC Reading Every 500 ms):**

```
212     UARTprintf("\r\nMSP432E401Y: ADC + PWM (inverted) + UART + SWI + WDT\r\n");
213
214     uint16_t tick500 = 0;
215     while (1)
216     {
217         if (g_adcReady) {
218             g_adcReady = false;
219             if (++tick500 >= 500) {
220                 tick500 = 0;
221                 UARTprintf("ADC = %4u\r\n", g_adcLatest);
222             }
223         }
224     }
225 }
```

Description: In the source code, UART was used to print multiple ADC channel readings continuously. I simplified this to display a single ADC value (from PE3) every **500 ms.** This reduced serial output clutter and provided clear feedback of the joystick's analog position in real time. The UART operates on **UART0 (PA0/PA1)** at 115200 bps, and the console output serves as a straightforward way to verify that the ADC and DMA subsystems are functioning correctly. This modification was made for a clear digital readout without overwhelming the terminal.

Task 4 – Switch Interrupts (SW1/SW2 on PJ0/PJ1)

- **Original Code (Template):**
  None – source code had no GPIO input logic.
- **Added Code:**

```
186 static void initSwitches(void)
187 {
188     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
189     while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOJ)) {}
190     MAP_GPIOPadConfigSet(SW_GPIO_BASE, SW1_PIN | SW2_PIN,
191                          GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
192     MAP_GPIOIntDisable(SW_GPIO_BASE, SW1_PIN | SW2_PIN);
193     MAP_GPIOIntTypeSet(SW_GPIO_BASE, SW1_PIN | SW2_PIN, GPIO_FALLING_EDGE);
194     MAP_GPIOIntClear(SW_GPIO_BASE, SW1_PIN | SW2_PIN);
195     MAP_GPIOIntEnable(SW_GPIO_BASE, SW1_PIN | SW2_PIN);
196     MAP_IntEnable(INT_GPIOJ);
197 }
```

```
88 void GPIOJ_IRQHandler(void)
89 {
90     uint32_t status = MAP_GPIOIntStatus(SW_GPIO_BASE, true);
91     MAP_GPIOIntClear(SW_GPIO_BASE, status);
92     if (status & (SW1_PIN | SW2_PIN))
93         setPWMDuty_u16(g_adcLatest);
94 }
95
```

Description: Switch input handling was not present in the original source code, so I added **interrupt-driven control** for the two onboard switches (SW1 and SW1). Each button press triggers a GPIO interrupt that reads the current joystick ADC value and updates the PWM duty cycle. This ensures that LED brightness changes only when a switch is pressed, rather than continuously, which aligns with requirements for event-based control. Using interrupts instead of polling also improves responsiveness and conserves processing time between button events.

Task 5 – Watchdog Timer Heartbeat

- **Original Code (Template):**
  None – no watchdog used in source code.
- **Added Code:**

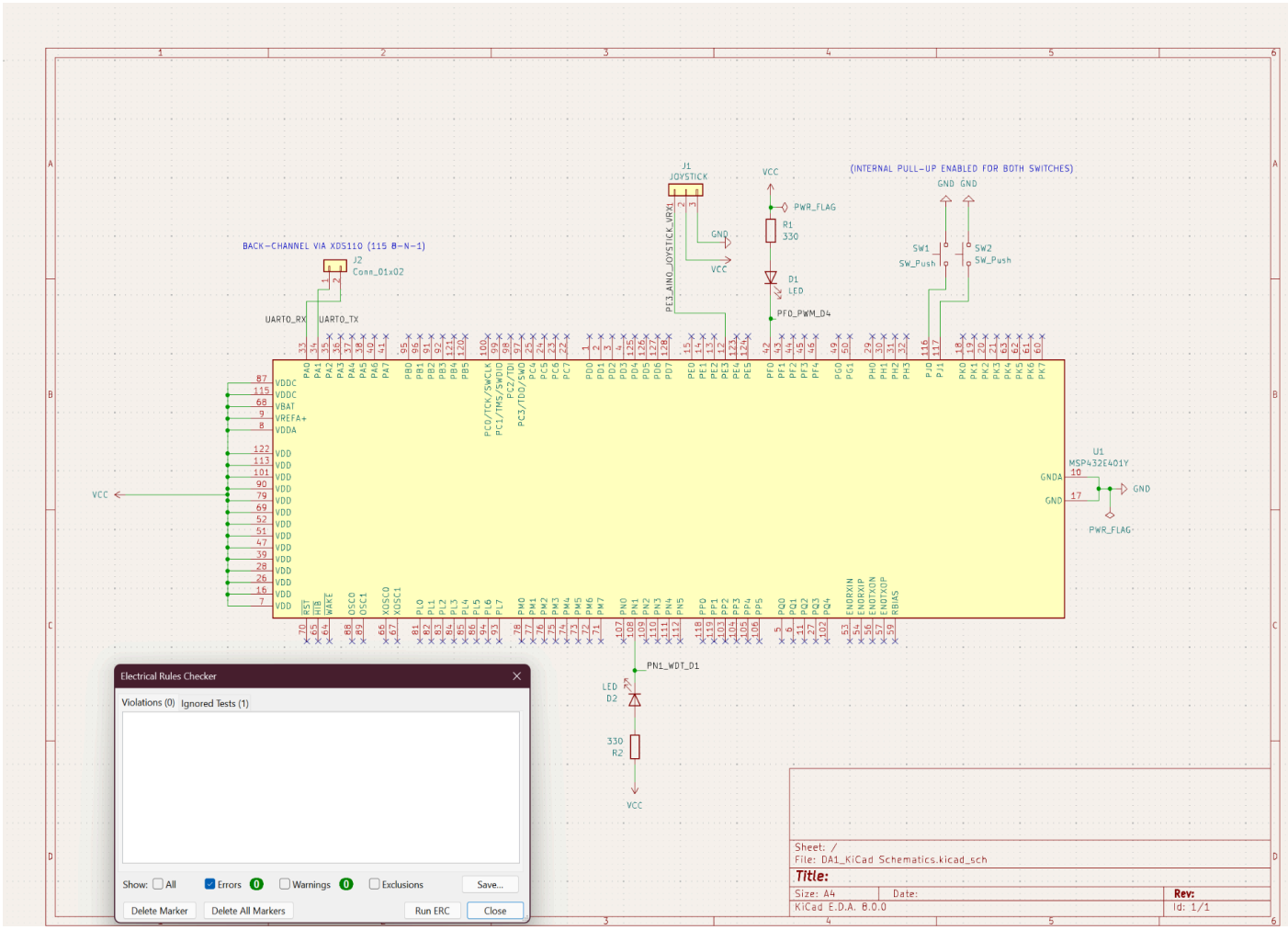
```
167 static void initHeartbeat_WDT(uint32_t sysclk)
168 {
169     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
170     while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_GPION)) {}
171     MAP_GPIOPinTypeGPIOOutput(HB_GPIO_BASE, HB_GPIO_PIN);
172     MAP_GPIOPinWrite(HB_GPIO_BASE, HB_GPIO_PIN, 0);
173
174     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);
175     while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_WDOG0)) {}
176     MAP_WatchdogUnlock(WATCHDOG0_BASE);
177     MAP_WatchdogResetDisable(WATCHDOG0_BASE);
178     MAP_WatchdogIntEnable(WATCHDOG0_BASE);
179     MAP_WatchdogStallEnable(WATCHDOG0_BASE);
180     MAP_WatchdogReloadSet(WATCHDOG0_BASE, sysclk / 2U);
181     WatchdogIntRegister(WATCHDOG0_BASE, WatchdogISR);
182     MAP_IntEnable(INT_WATCHDOG);
183     MAP_WatchdogEnable(WATCHDOG0_BASE);
184 }
185
```

```
69 /* --- ISRs --- */
70 static void WatchdogISR(void)
71 {
72     MAP_WatchdogIntClear(WATCHDOG0_BASE);
73     MAP_GPIOPinWrite(HB_GPIO_BASE, HB_GPIO_PIN,
74         (MAP_GPIOPinRead(HB_GPIO_BASE, HB_GPIO_PIN) ^ HB_GPIO_PIN));
75 }
76
```

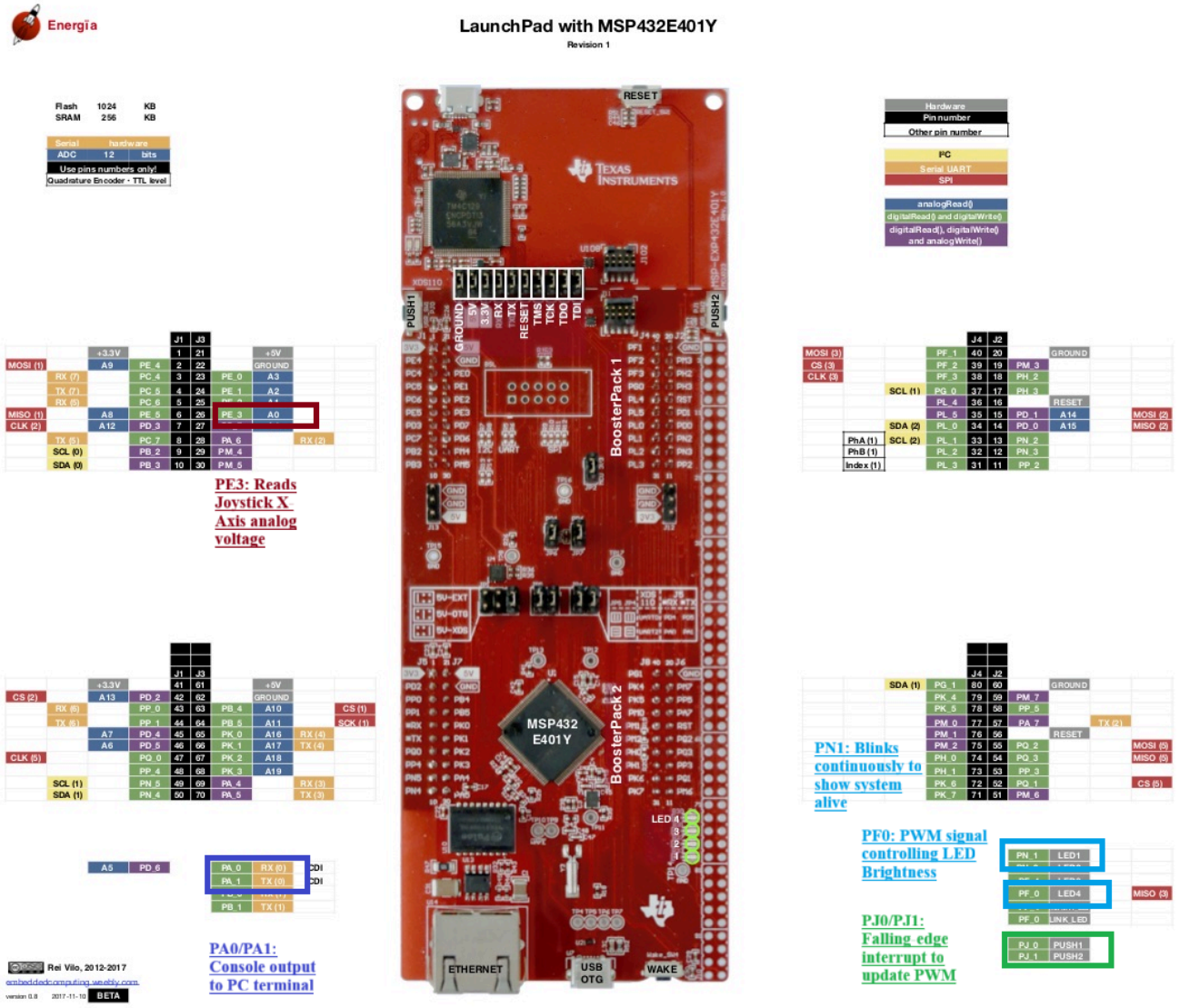
Description: The source code had no watchdog functionality. I implemented **Watchdog Timer 0** in **interrupt mode** to generate a 2Hz heartbeat signal that toggles the **D1 (PN1)** LED. This serves as a visual indicator that the system is running and that the interrupt infrastructure is functioning properly. The watchdog was configured to generate interrupts instead of performing resets, and I used ***WatchdogIntRegister()*** to ensure the handler was correctly registered for the MSP-EXP432E401Y. This addition provides a reliable, non-blocking method to monitor system activity during real-time operation.

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.html and https://www.snapeda.com/parts/CC1352P1F3RGZT/Texas%20Instruments/view-part/

KiCad Design Schematics (ERC Compliant):

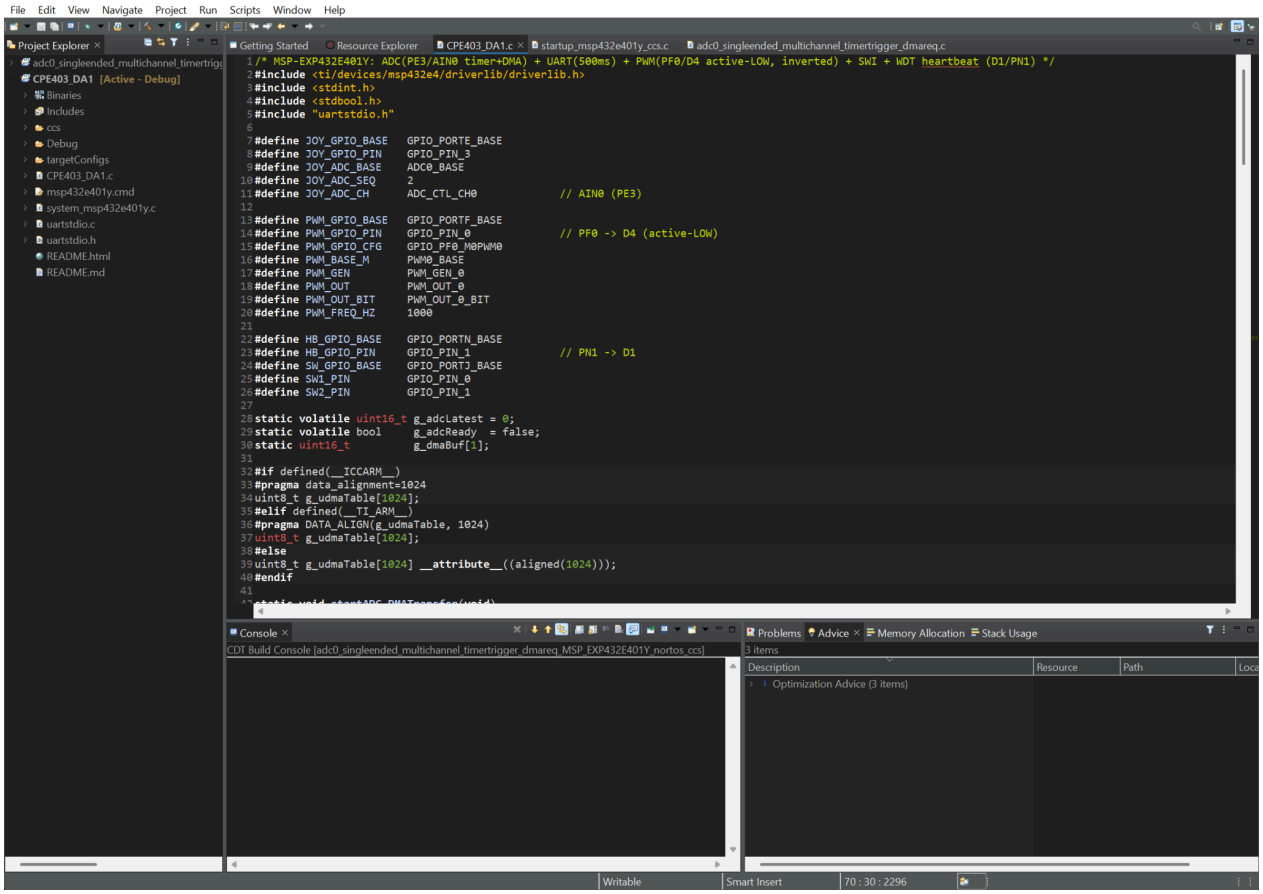


Block Diagram:

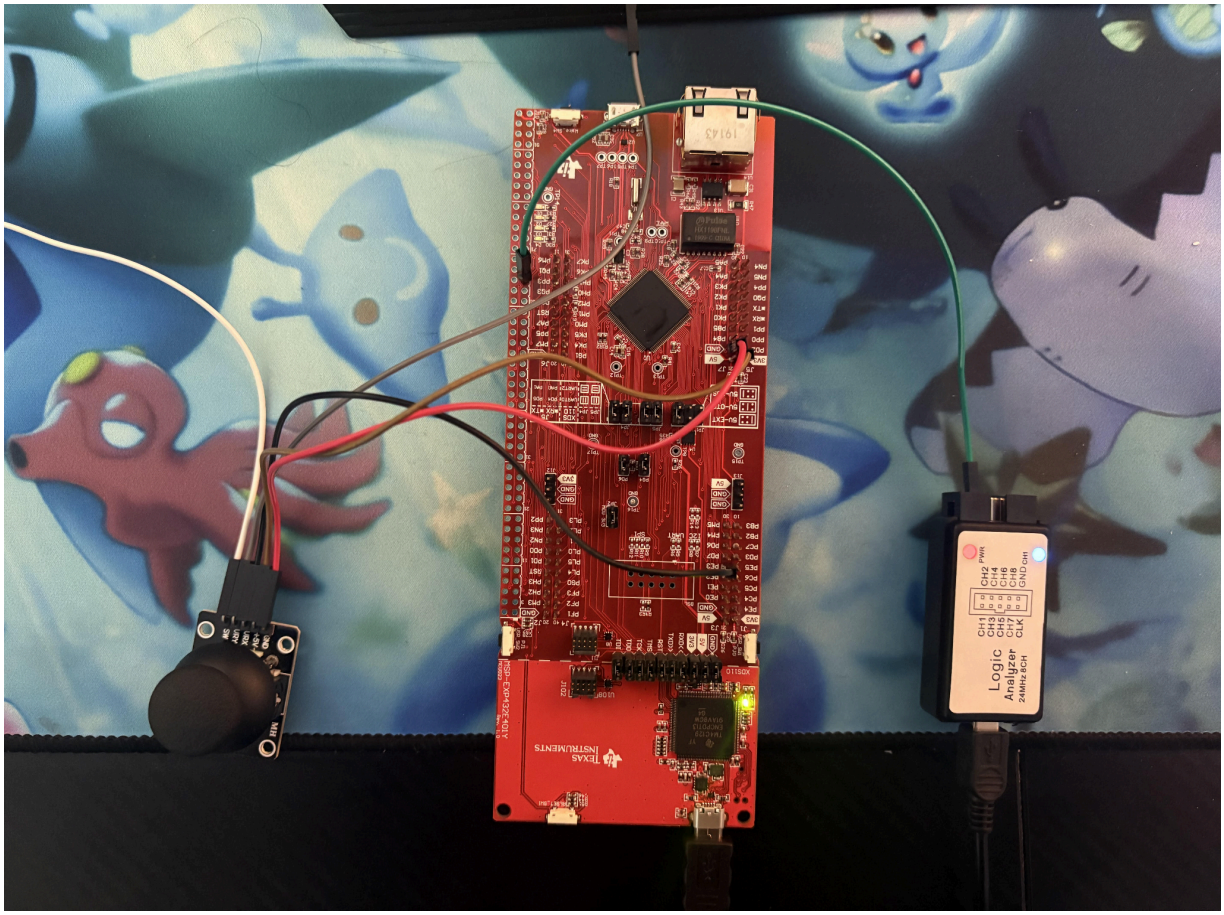


3. Screenshots of the IDE, physical setup, debugging process – Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.

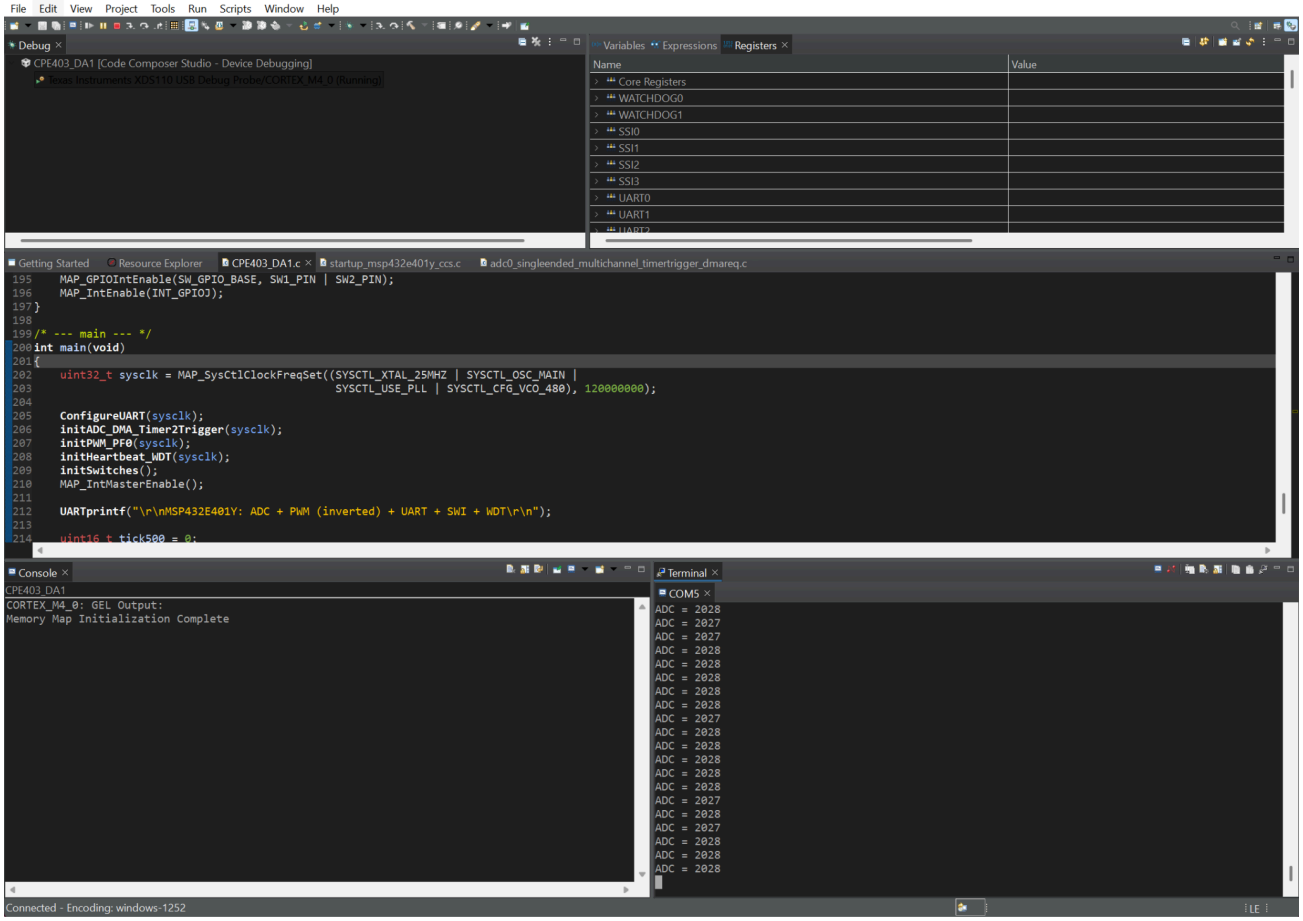
IDE Workspace:



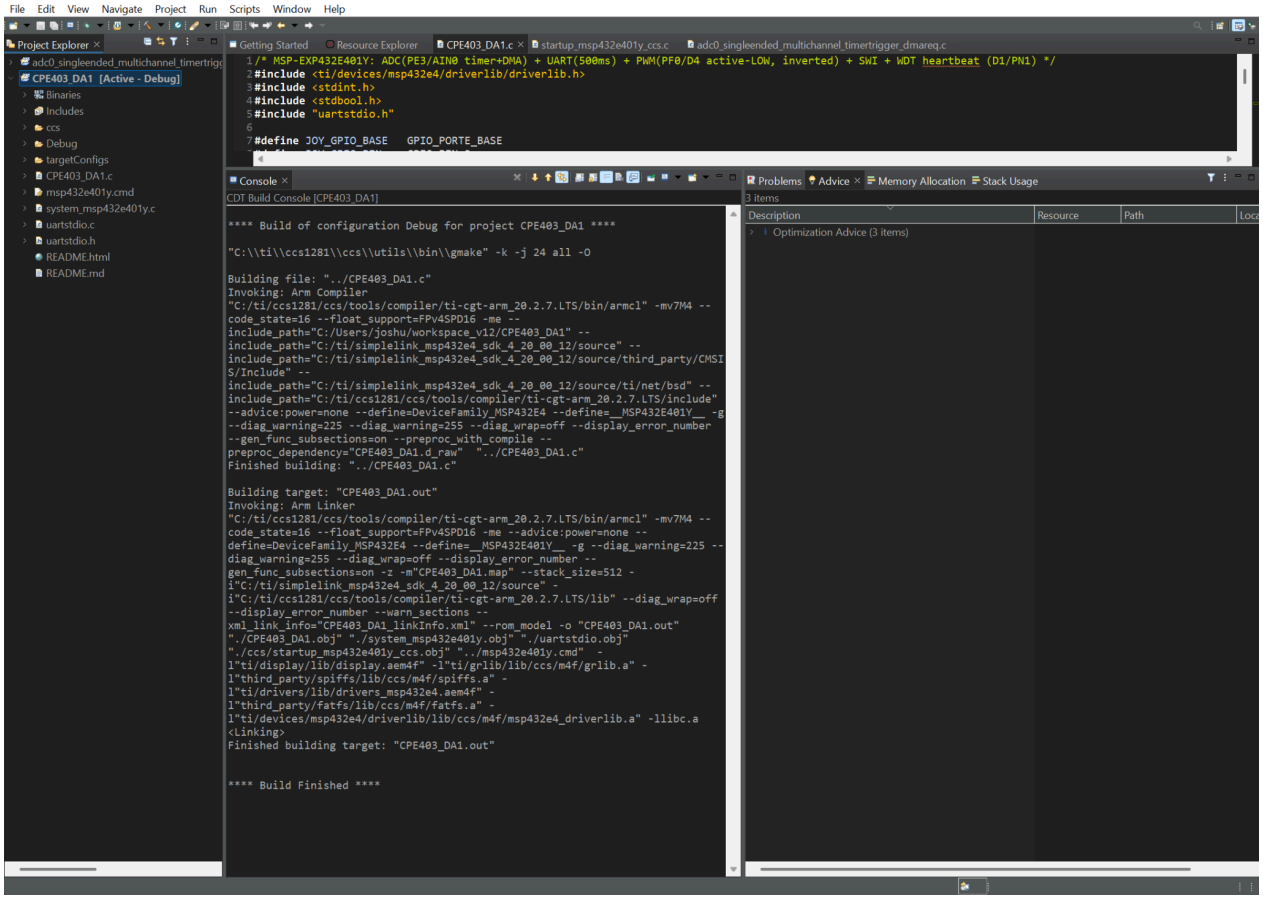
Physical Setup:



Debugging Process:



Successful Compilation:



4. Declaration

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

"This assignment submission is my own, original work".
Joshua Martinez