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

Design Assignment 2

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 – Base Project Setup (CMSIS DSP + ADC w/ DMA + FFT)

- **Original Code (Before Changes):**

```
184     /* Display the setup on the console. */
185     UARTprintf("\033[2J\033[H");
186     UARTprintf("\rCMSIS DSP Demo...\n\n");
187     UARTprintf("\033[2GDC Average \033[31G\n");
188     UARTprintf("\033[2GRMS \033[31G\n");
189     UARTprintf("\033[2GFFT Amplitude \033[31G\n");
190     UARTprintf("\033[2GFFT Frequency \033[31G\n");
191
```

- **Modified Code (After Changes):**

```
66
67    UARTprintf("%s  peak=%d.%03d @ %d.%01d Hz  rms=%d.%03d  dc=%d.%03d\r\n",
68               tag,
69               peak_milli/1000,  abs(peak_milli%1000),
70               hz_tenths/10,     abs(hz_tenths%10),
71               rms_milli/1000,   abs(rms_milli%1000),
72               dc_milli/1000,    abs(dc_milli%1000));
73 }
```

Description: The original code initially only displayed DC, RMS, and FFT values using ANSI cursor controls. To support real-time streaming into Python and improve readability, I reformatted output to a single line per acquisition, including:

- Peak amplitude
- Frequency bin converted to Hz
- RMS of the audio waveform
- DC offset value

This ensures clean UART decoding for Python while still supporting terminal readability.

Task 2 – Hardware Setup (Microphone Input via ADC)

- **Original Code (GPIO & ADC Input):**

```
98
99      /* Configure PE3 as ADC input channel */
00      MAP_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);
01
```

- **Modified Code (Complete ADC + DMA setup):**

```
/* --------- GPIOE / PE3 as ADC input (AIN0) --------- */
MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE)) { }
MAP_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3); /* PE3 = AIN0 */

/* --------- ADC0 + Sequencer 2 (single sample) ------- */
MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_ADC0)) { }

MAP_ADCHardwareOversampleConfigure(ADC0_BASE, 8);
MAP_ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_CH0 | ADC_CTL_IE | ADC_CTL_END);
MAP_ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_TIMER, 2);

MAP_ADCIntClearEx(ADC0_BASE, ADC_INT_DMA_SS2 | ADC_INT_SS2);
MAP_ADCIntEnableEx(ADC0_BASE, ADC_INT_DMA_SS2 | ADC_INT_SS2);

MAP_ADCSequenceDMAEnable(ADC0_BASE, 2);
MAP_ADCSequenceEnable(ADC0_BASE, 2);
```

Description: I connected the MAX4466 microphone OUT → PE3(AIN0) and configured:

- Timer-triggered ADC sampling
- DMA Ping-Pong buffering (continuous acquisition at 100kHz)
- Interrupt notification when buffer is ready

This allows capturing over **over 1000 samples per FFT window** without CPU latency impacting performance.

Task 3 – Software Processing (FFT, RMS, Frequency Detection):

- **Original FFT Processing Code:**

```
/* Compute the 1024 point FFT on the sampled data and then find the
 * FFT point for maximum energy and the energy value */
arm_cfft_q15(&arm_cfft_sR_q15_len1024, (q15_t *)dstBufferA, IFFTFLAG,
             BITREVERSE);
arm_cmplx_mag_q15((q15_t *)dstBufferA, (q15_t *)fftOutput,
                  NUM_SAMPLES);
arm_max_q15((q15_t *)fftOutput, NUM_SAMPLES, (q15_t *)&setFFTmaxValue,
            &setFFTmaxFreqIndex);
```

- **Modified Full Processing Snippet:**

```
for (i = 0; i < NUM_SAMPLES; ++i) {
    float32_t s = ((float32_t)adcBufA[i] - 2048.0f) * (1.0f / 2048.0f);
    x_time[i] = s * hann[i];
}
arm_rms_f32(x_time, NUM_SAMPLES, &rmsVal);
arm_mean_f32(x_time, NUM_SAMPLES, &dcMean);
arm_rfft_fast_f32(&rfft, x_time, X_freq, 0);
arm_cmplx_mag_f32(X_freq, mag, NUM_SAMPLES/2);
arm_max_f32(mag, NUM_SAMPLES/2, &peakVal, &peakBin);
peakHz = ((float32_t)peakBin) * ((float32_t)SAMP_FREQ) / ((float32_t)NUM_SAMPLES);
```

Description: This implements the core DSP requirement:

- Calculate FFT using ARM-optimized CMSIS DSP
- Convert FFT complex output → magnitude
- Detect frequency with highest energy
- Convert bin index into Hz for display

This enables real-time sound frequency detection.

Task 4 – Python Interface (Real-Time FFT Visualization):

- **New Python UART Streaming Code Added:**

```
# --- settings ---
PORT = "COM5"
BAUD = 115200
SPEC_HISTORY = 120   # frames to keep in spectrogram (~15 s if ~8 fps)
TIMEOUT = 1.0

# open serial
ser = serial.Serial(PORT, BAUD, timeout=TIMEOUT)
time.sleep(0.8)           # give the board a moment after opening the port
ser.write(b'F')           # toggle stream ON in firmware
ser.flush()
print(f"Connected to {PORT} @ {BAUD}")

# live plots
plt.ion()
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(9,7))

line, = ax1.plot([], [], lw=1)
ax1.set_xlabel("Frequency (Hz)")
ax1.set_ylabel("Normalized magnitude")
ax1.set_ylim(0, 1.05)
```

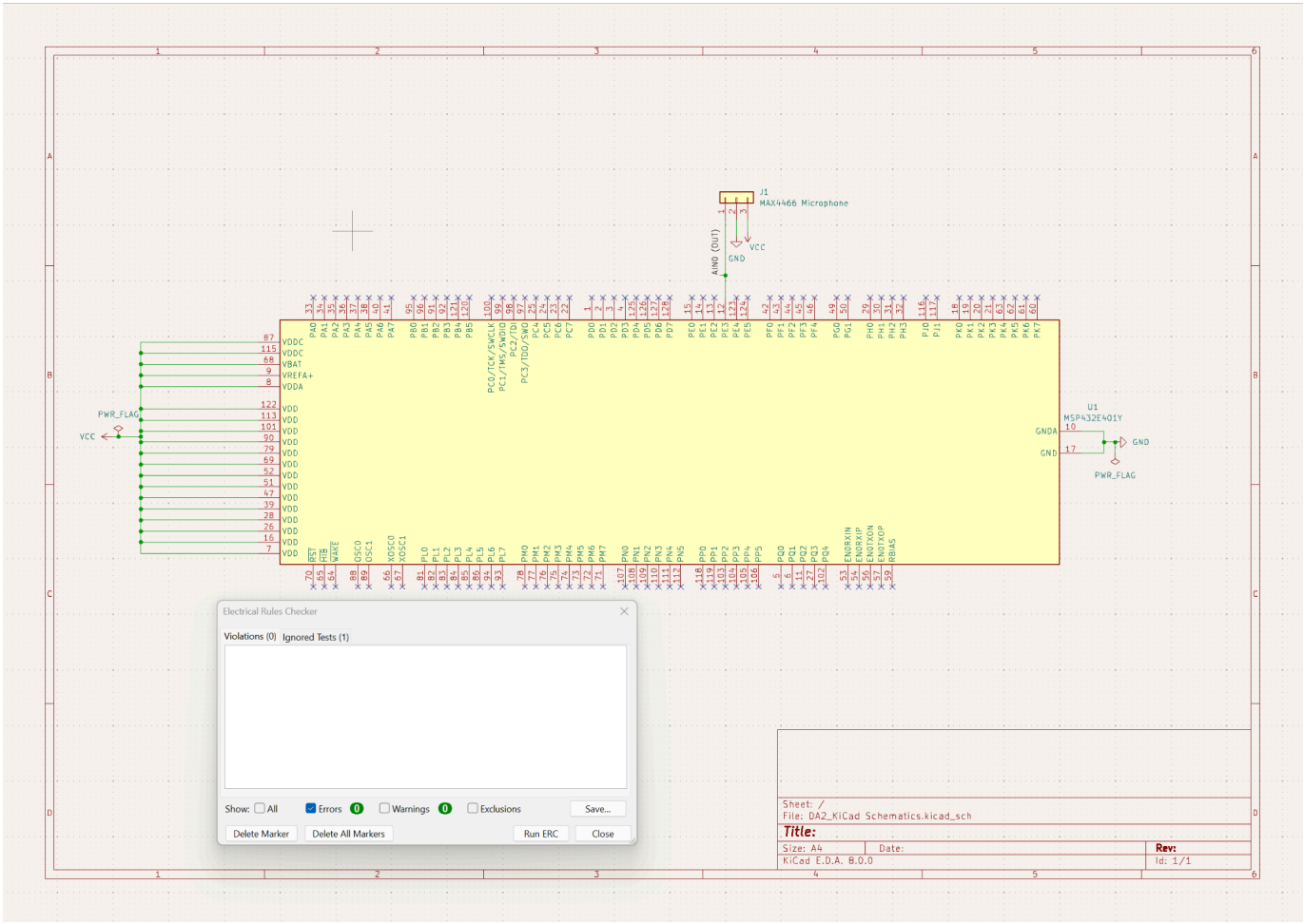
Description: To meet Task 4 requirements, I developed a Python real-time spectrogram/FFT plot tool:

- UART data stream parsing
- Automatic graph refresh
- Shows audio preak frequency in real-time
- Validates FFT output with test tones

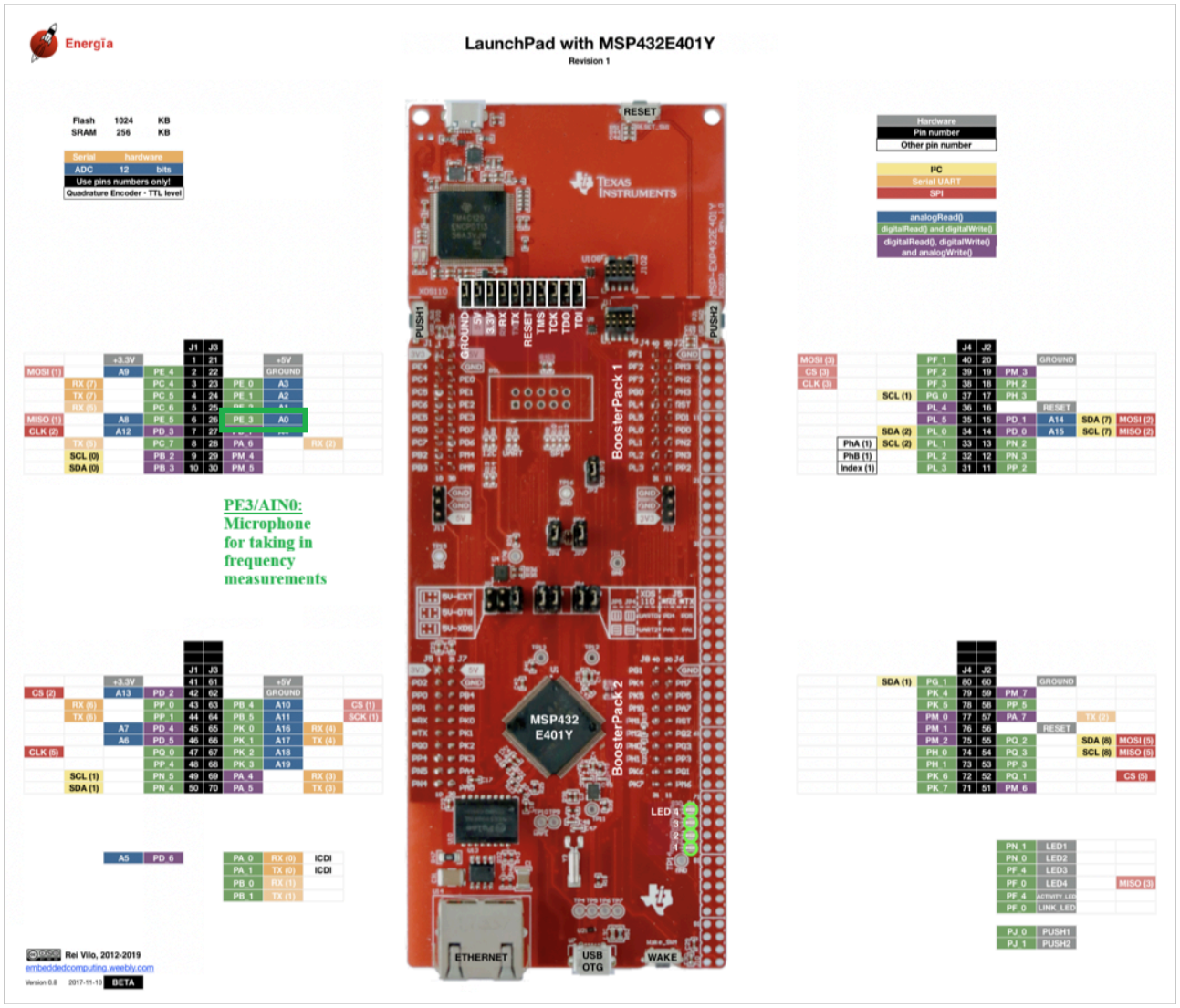
This completes the end-to-end system design: microphone → ADC → FFT → PC visualization.

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):

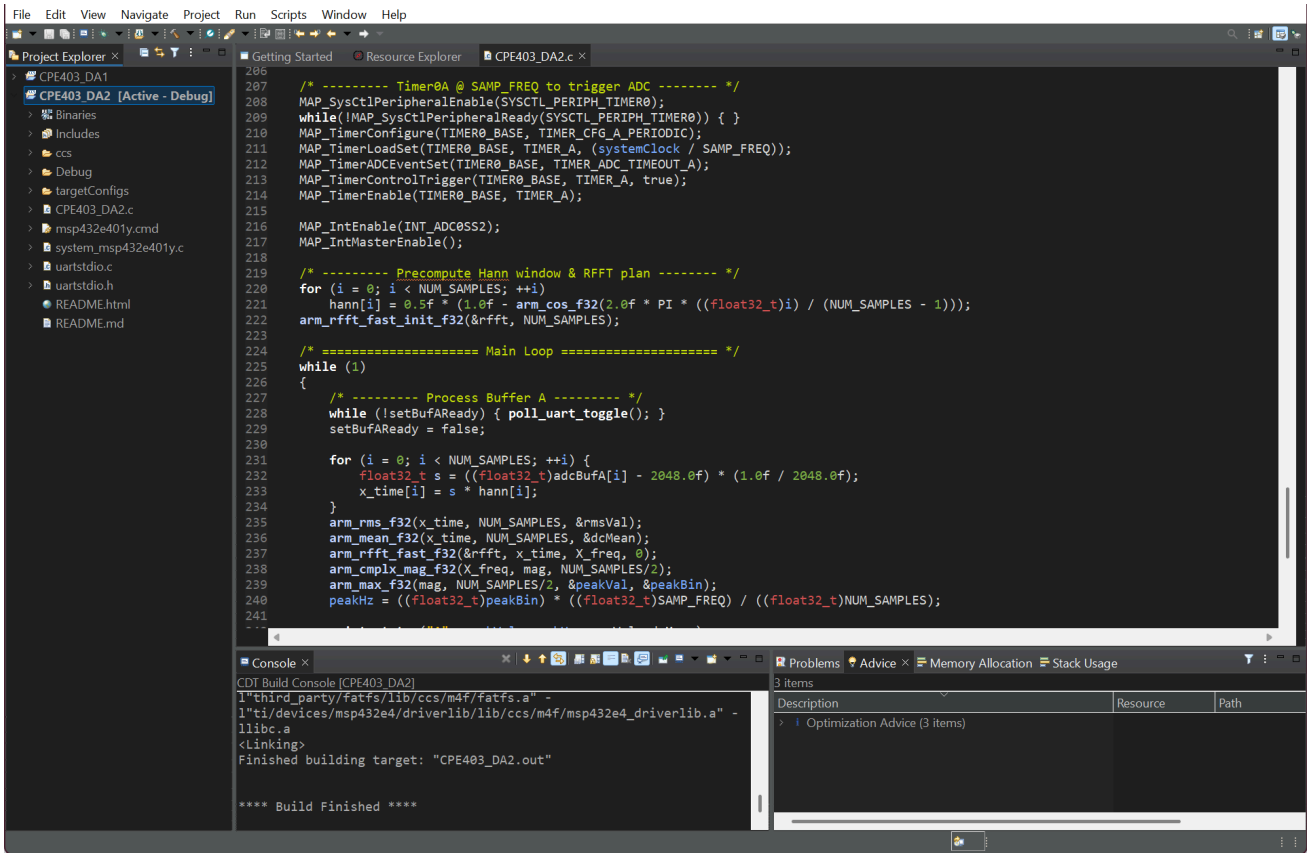


Block Diagram:

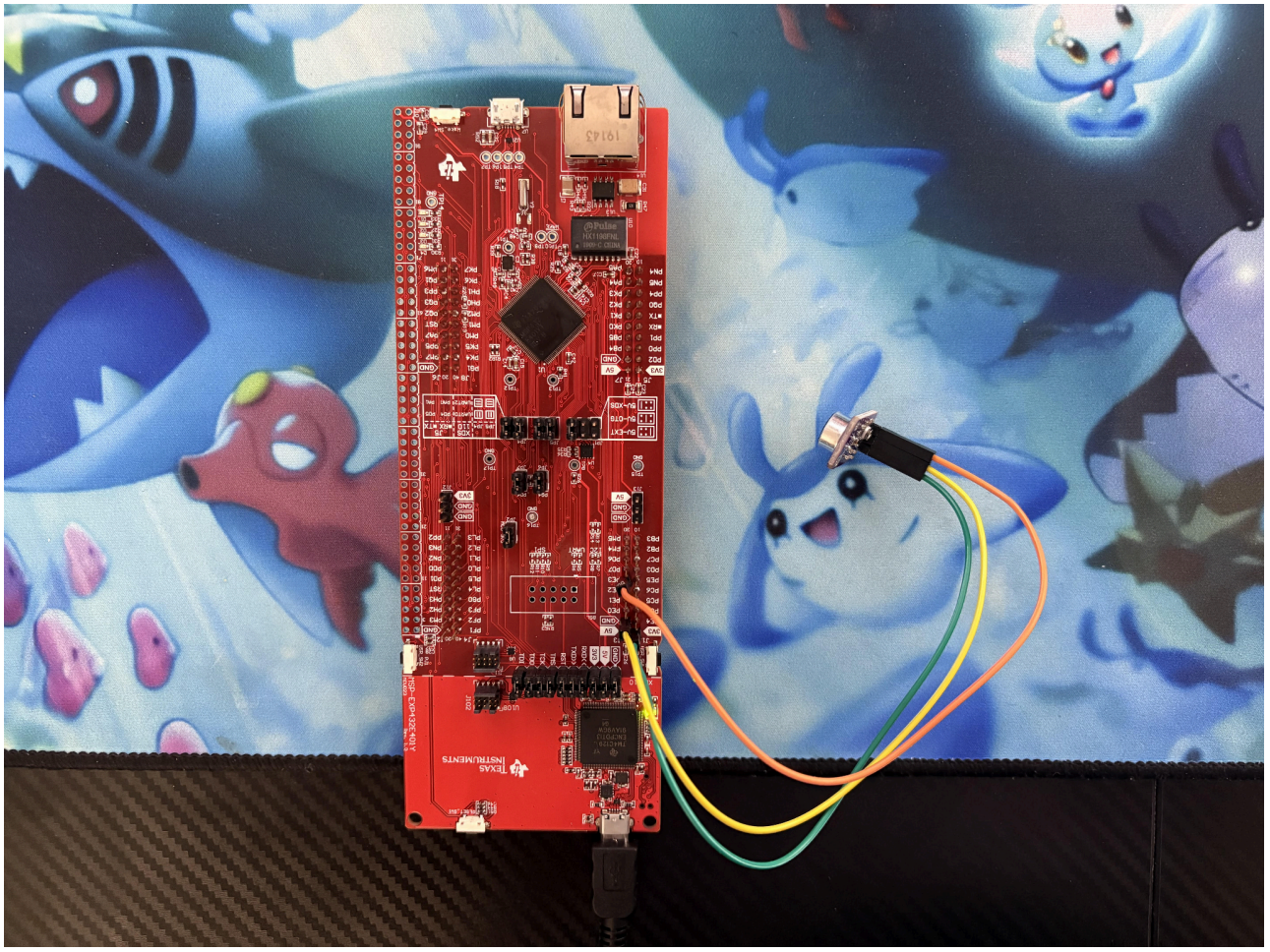


3. Screenshots of the IDE, physical setup, debugging process – Provide screenshot of successful compilation, screenshots of registers, variables, graphs, etc.

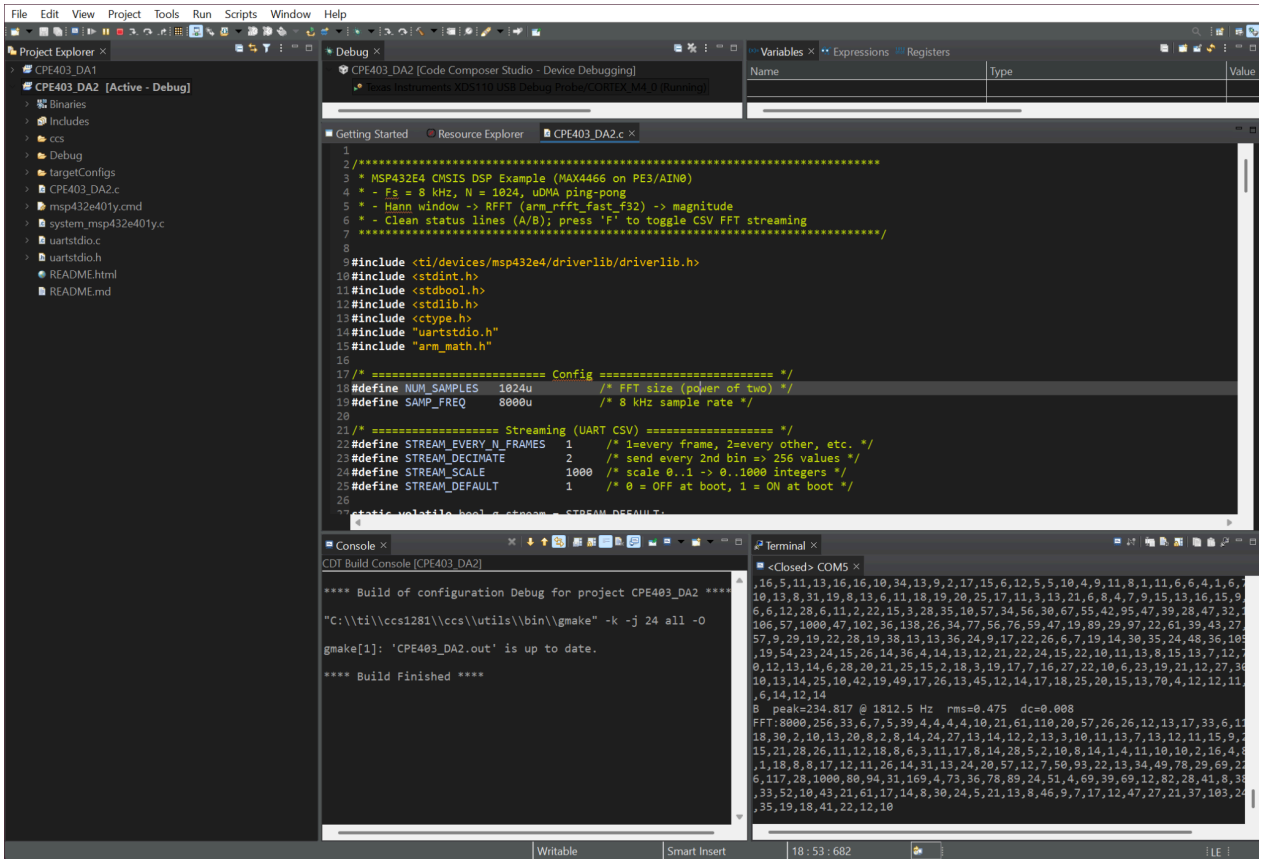
IDE Workspace:



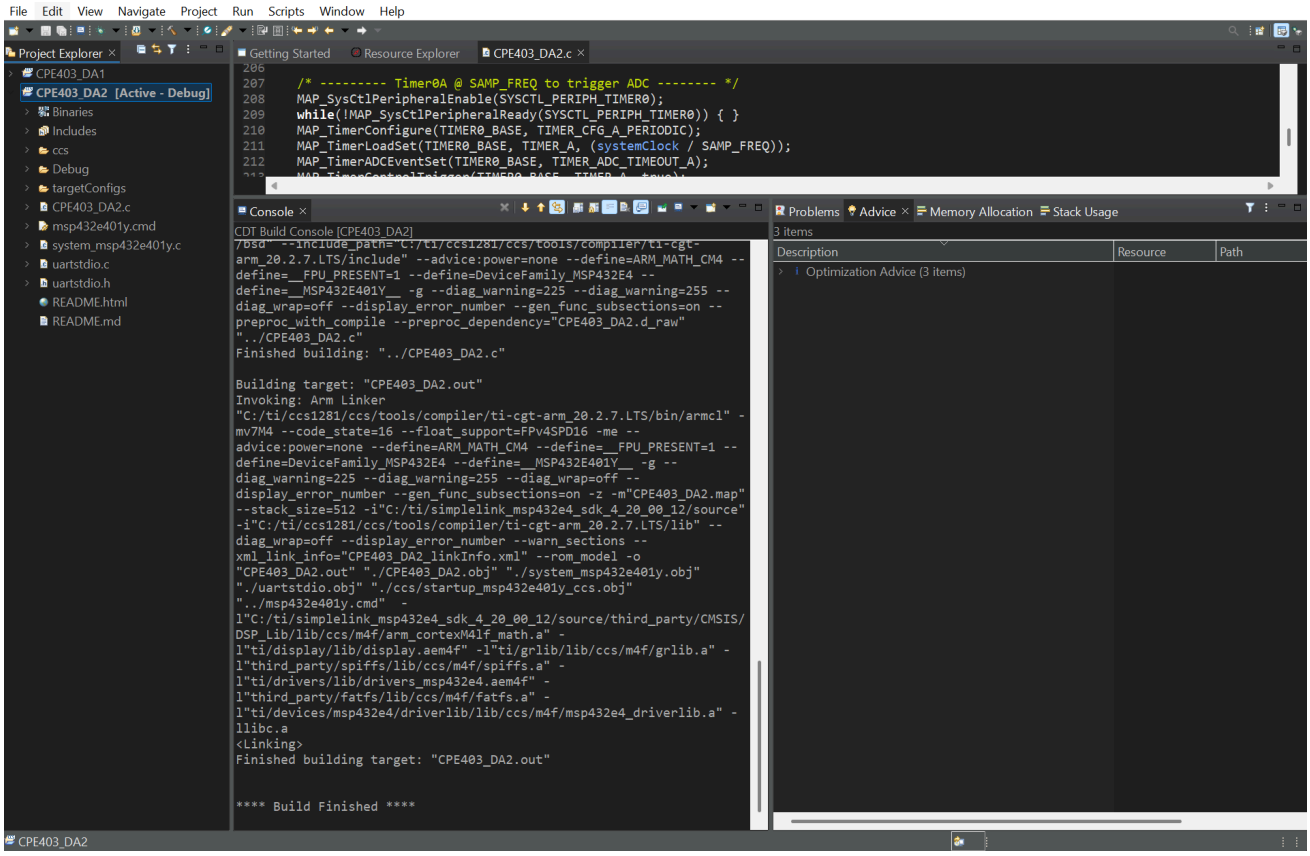
Physical Setup:



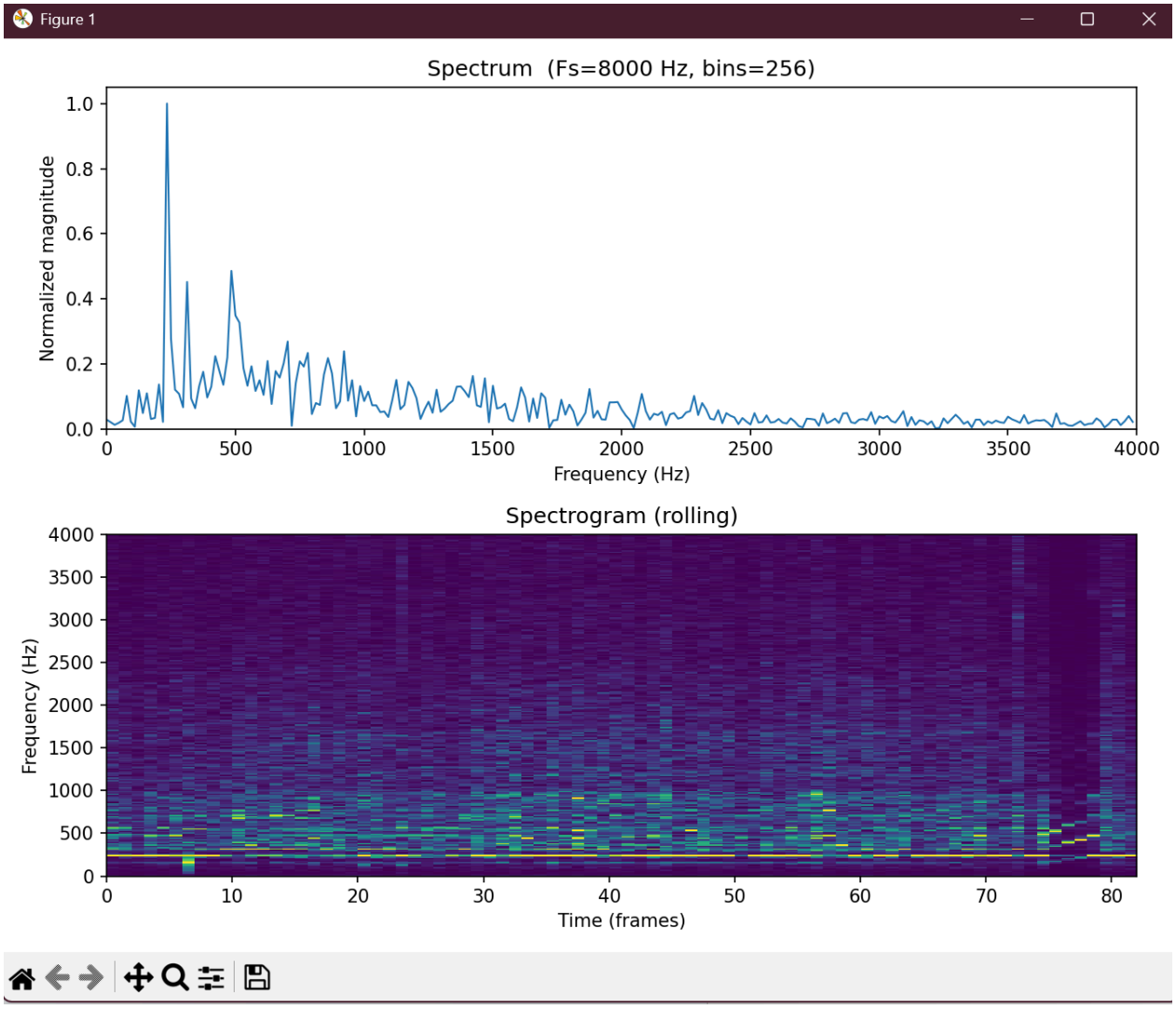
Debugging Process:



Successful Compilation:



Python Graph:



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