ECE 4240 – Lab 3 Report

ADCs, Sensing and Bad DAC

Course: ECE 4240 – Microprocessor Interfacing

Instructor: Dr. Dean McNeill

Student: Dayaveer Gill (7969885)

Lab Partner: Severyn Shved

Date Performed: November 3, 2025

Date Submitted: November 18, 2025

University of Manitoba

Department of Electrical and Computer Engineering

Laboratory Experiments

Introduction:

This lab consisted of three parts, first part required us to use ADC1 and the internal temperature sensor on the STM board to output the average temperature to the console. Part 2 required us to use external temperature sensor that is continuously polling the temperature and turns on a LED if the current temperature is less than the desired temperature and turns off once the desired temperature is met. It also uses hysteresis to minimize flickering near the desired temperature. For part 3 we were required to get the approximate the DAC for a PWM with a low pass filter and duty cycle ranging from 0 to 100%.

Part 1:

For the first part of this lab we begun by programming the board as instructed in part 1, and adding all the files to their necessary locations. Then we wrote the code to sample the internal temperature sensor using ADC1 eight times per second these values are run through a moving average and then once per second the calibrated value from the moving average is converted to °C using the formula provided in the lab manual with values of $ADC_{Precision}$, MV_{25C} and AvgSlope values being gotten from data sheet and is then sent displayed over console. The results were shown to a TA. Figures 1 show the code for part 1 and Figure 2 shows the results of the console.

Part 2:

In part 2 we started off by enabling ADC2 and setting it to the alternative pins like directed by the lab manual. Then we wrote code to over-sample the TMP36 temperature sensor eight times, eight times per second. The temperature is ran through an moving average and converted to °C using the equation provided in the lab manual. Output of the voltage divider created by the potentiometer is over-sampled 16 times four times per second and ran through an moving average and scaled to a range of 50°C centered at 25°C. We also implemented hysteresis control loop with a hyRange value of 2°C. The results were shown to a TA. Figures 3 show the code for part 2.

Part 3:

For part 3 we started off enabling the PMW using the code from previous lab. Then we made our circuit and wrote code which over samples the approximated DAC output 64 times and averages it and send the result to the console. We used a 0.1 microfarad capacitor and 1.5k

Ω resistor which gives us a cut off frequency of 1,061 Hz. We then used the oscilloscope to verify the approximated output DAC which is shown in figure 4 in the below. We results were then shown to the TA. The results from the oscilloscope are shown in figure 4 and our graph of 'ADC Average vs Duty Cycle' is shown in figure 5, the code for part 3 is shown in figure 6 which is attached in the appendix.

Conclusion:

Overall this lab showed us how to work with ADC, internal and external temperature sensors. For part 1, we used the internal temperature sensor and ADC1 to get the averaged temperature. For part 2, we used the external temperature sensor, ADC2 and a potentiometer to get the averaged temperature and set a desired cutoff temperature with the potentiometer and had the light turn on when the current averaged temperature was lower than the desired cutoff temperature and turn off when it was above, we also implemented hysteresis control loop with hysteresis range of two degrees. For part 3 we used a PWM signal with a low pass filter to get the approximated DAC and plotted the values of the ADC response and saw the output on the oscilloscope. The equation for the best fit line was calculated to be y = 14.327x - 12.925 and MSE to be 23.77.

Laboratory Questions:

Question 1:

We require I_{Max} to be 2 mA and V_{Max} to be 3V then we get:

$$R_{\text{Max}} = \frac{V_{\text{Max}}}{I_{\text{Max}}}$$

$$R_{\text{Max}} = \frac{3}{0.002} = 1500\Omega$$

Question 2A:

Assuming the timer clock is still configured to 160MHz from part 3 of the lab if it isn't we would configure it to 160MHz, and then enable timer 1, TIM1 and configure the prescaler to 159 and the ARR to 99, which give us the required trigger frequency of 10kHz. Then we can enable ADC1 and set the resolution to 8 bits. This would ensure the timer goes up to 10kHz and then reset as required.

Question 2B:

From the datasheet we know the STM32F407 has 196kB of SRAM and sampling would be 10,000 bytes per second[2].

$$196kB - 64kB = 132kB = 132,000\ bytes$$

$$totalTime(Seconds) = \frac{132,000}{10,000} = 13.2\ Seconds$$

Question 2C:

We get the dynamic range is DR = 6.021N + 1.763 dB [1].

$$DR_{8\text{ bit}} = 6.021(8) + 1.763 = 49.931dB$$

$$DR_{8\text{ bit}} = 6.021(10) + 1.763 = 61.973dB$$

$$DR_{8\text{ bit}} = 6.021(12) + 1.763 = 74.015dB$$

Question 3:

Another system that uses hysteresis is hot water tank where the heating element turns on when the water temperature is below a certain temperature and turns off when its above an certain temperature. If the hot water tank did not have hysteresis the heating element would flicker a lot when warming up and it reaches around the cut off temperature at which the heating element starts. This would cause the heater to burn out extremely fast.

Additional Questions:

Question 4:

One of the main reasons why the output of a Wheatstone bridge can't be directly read into a ADC is because the voltage difference from a Wheatstone bridge is often very small, in milli-volts which would not be able to use the full range of the ADC and result in poor accuracy[3].

Question 5.A:

The slope for the perfectly linear line is more steep since the the voltage RC filter does not output the whole PMW voltage which makes perfectly linear line more steep. The reason the

RC filter does not output the whole PWM voltage can be due to the noise and not exact values for the resistor and capacitor.

Question 5.B:

From my graph of ”ADC Average vs Duty Cycle” I can see that the ADC values are monotonic since for every duty cycle increase the ADC value is also increasing. ’The ADC count vs duty cycle’ line is linear since it fits perfectly to the best fit line and produces a R^2 value of 1.

Citations

[1] C. Slattery and M. McCarthy, *Oversampled ADC and PGA Combine to Provide 127-dB Dynamic Range*, Available: https://www.analog.com/en/resources/analog-dialogue/articles/oversampled-adc-pga-dynamic-range.html, accessed Nov. 18, 2025.

[2] STMicroelectronics, *STM32F407xx Datasheet*, Nov. 2024, Available: https://www.st.com/resource/en/datasheet/DM00037051.pdf, accessed Nov. 19, 2025.

[3] Altium, *An Introduction to Wheatstone Bridge Circuits and Differential Amplifiers*, Available: https://resources.altium.com/p/wheatstone-bridges, accessed Nov. 18, 2025.

Appendix:

```
uint32_t start = HAL_GetTick();
uint32_t start2 = HAL_GetTick();
float movingAverageArray[8] = {0};
float movingAvgADC = 0;
float movingAverageTemp = 0;

//Equation Variables
float AvgSlope = 2.5;
float ADCPrecision = 4095; //2^12 - 1
float vRef = 3300;
float mV25C = 760;

//Function
void movingAverage(float inputTemp){
    int i = 0;
    movingAvgADC = 0;
    while (i < 7){
        movingAverageArray[i]= movingAverageArray[i+1];
        movingAvgADC += movingAverageArray[i];
        i++;
    }
    movingAvgADC += inputTemp;
    movingAverageArray[7] =  inputTemp;
```

```
    movingAvgADC = movingAvgADC/8;
}
while (1){
    if ((HAL_GetTick()- start)>125){
        start = HAL_GetTick();
        uint16_t ADCValue = getCalibratedADCReading(&hadc1,
        ADC_CHANNEL_TEMPSENSOR, ADC_SAMPLETIME_480CYCLES);
        movingAverage(ADCValue);
    }
    if ((HAL_GetTick()- start2)>1000){
        start2 = HAL_GetTick();
        float tempCel = (( (( movingAvgADC/ADCPrecision ) * vRef) -
        mV25C )/ AvgSlope) + 25;
        printf("The Temperature is: %.2f \260C\n",tempCel);
    }
}
```

Figure 1: Part 1 Code.

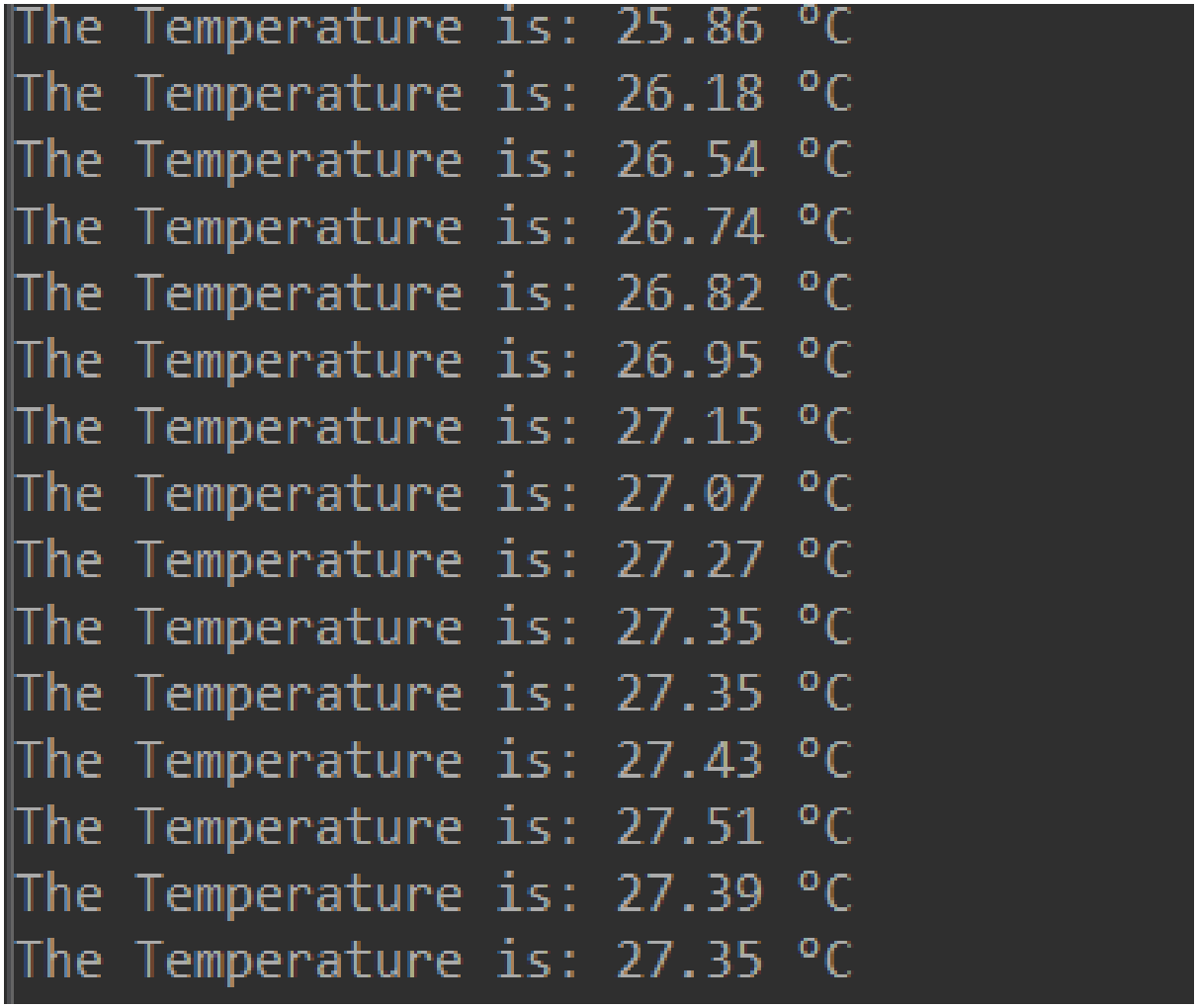

Figure 2: Part 1 Results.

```
uint32_t start = HAL_GetTick();
uint32_t start2 = HAL_GetTick();
float movingAverageArray[8] = {0};
float movingAverageTemp = 0;
float movingAverageArrayPot[16] = {0};
float movingAveragePot = 0;
float hyRange = 2;
float tempCel = 0;
float potCel = 0;
float DACFINAL = 0
void movingAverageTemperature(float inputTemp){
    int i = 0;
    movingAverageTemp = 0;
    while (i < 7){
        movingAverageArray[i]= movingAverageArray[i+1];
```

```
        movingAverageTemp += movingAverageArray[i];
        i++;
    }
    movingAverageTemp += inputTemp;
    movingAverageArray[7] =  inputTemp;
    movingAverageTemp = movingAverageTemp/8;
}
void movingAveragePotentiometer(float inputPot){
    int i = 0;
    movingAveragePot = 0;
    while (i < 15){
        movingAverageArrayPot[i]= movingAverageArrayPot[i+1];
        movingAveragePot += movingAverageArrayPot[i];
        i++;
    }
    movingAveragePot += inputPot;
    movingAverageArrayPot[15] =  inputPot;
    movingAveragePot = movingAveragePot/16;
}
while (1){
    if ((HAL_GetTick()- start)>125){
    start = HAL_GetTick();
    int i = 0;
    while(i < 8){
        uint16_t SensorOutput = getCalibratedADCReading(&hadc2,
        ADC_CHANNEL_1, ADC_SAMPLETIME_480CYCLES);
        movingAverageTemperature(SensorOutput);
        printf("The Input  is: %d*C\n",SensorOutput);
        i++;
    }
    tempCel = (((movingAverageTemp/4095)*3300) - 500) / (10);
    // printf("The Temperature is: %.2f*C\n",tempCel);
    //printf("The Input Averaged is: %.2f*C\n",movingAverageTemp);
    }
    if ((HAL_GetTick()- start2)>250){
        start2 = HAL_GetTick();
        int i = 0;
        while(i < 16){
```

```
            uint16_t PotOutput = getCalibratedADCReading(&hadc2,
            ADC_CHANNEL_2, ADC_SAMPLETIME_480CYCLES);
            movingAveragePotentiometer(PotOutput);
            //printf("The potentiometer output: %d*C\n",PotOutput);
            i++;
        }
        potCel = ((movingAveragePot / 4095) * 50);
        printf("The potentiometer set at: %.2f*C\n",potCel);
    }

    if(tempCel <= (potCel - hyRange/ 2)){
        HAL_GPIO_WritePin(GPIOD, LD3_Pin, GPIO_PIN_SET);
    }
    else if(tempCel >= (potCel + hyRange/ 2)){
        HAL_GPIO_WritePin(GPIOD, LD3_Pin, GPIO_PIN_RESET);
    }
}
```

Figure 3: Part 2 Code.

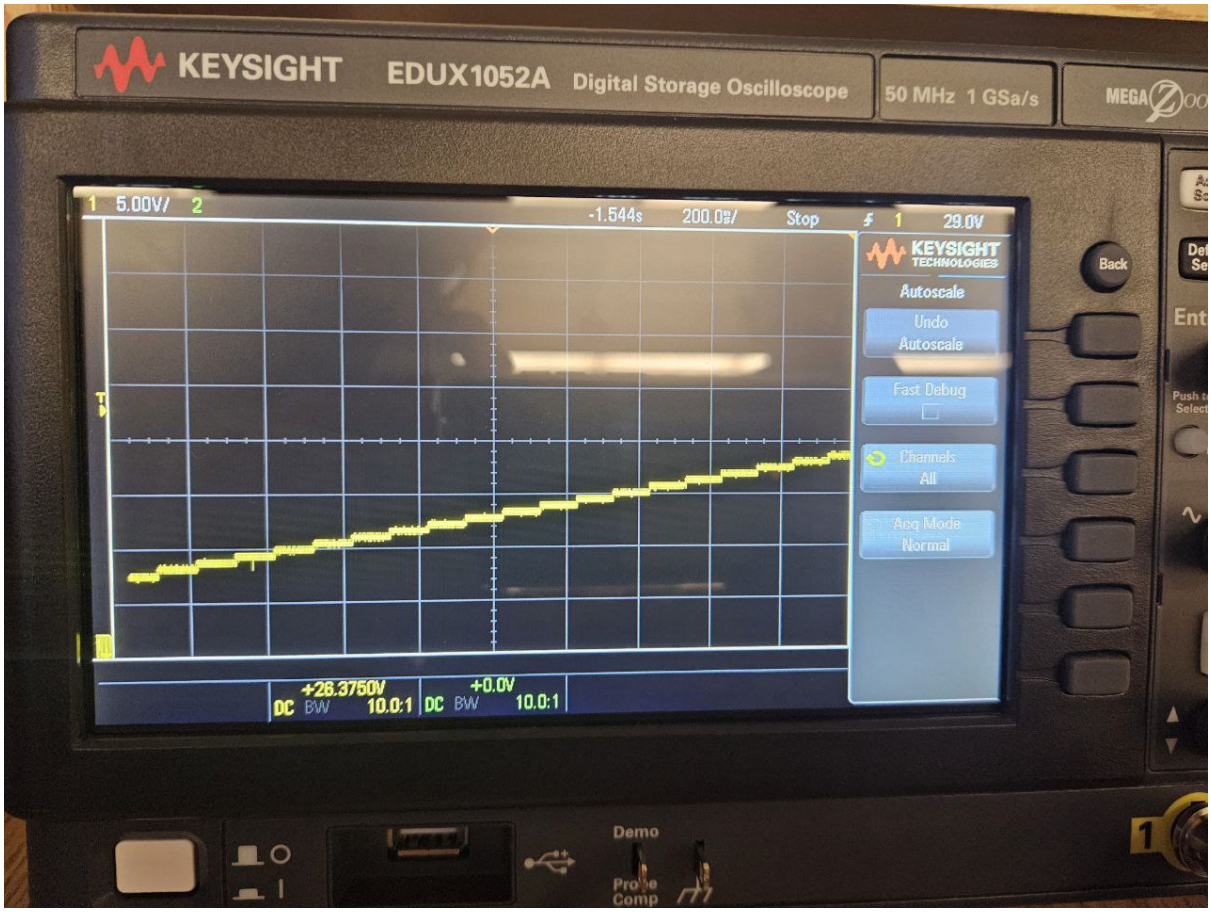

Figure 4: Part 3 oscilloscope sweep.

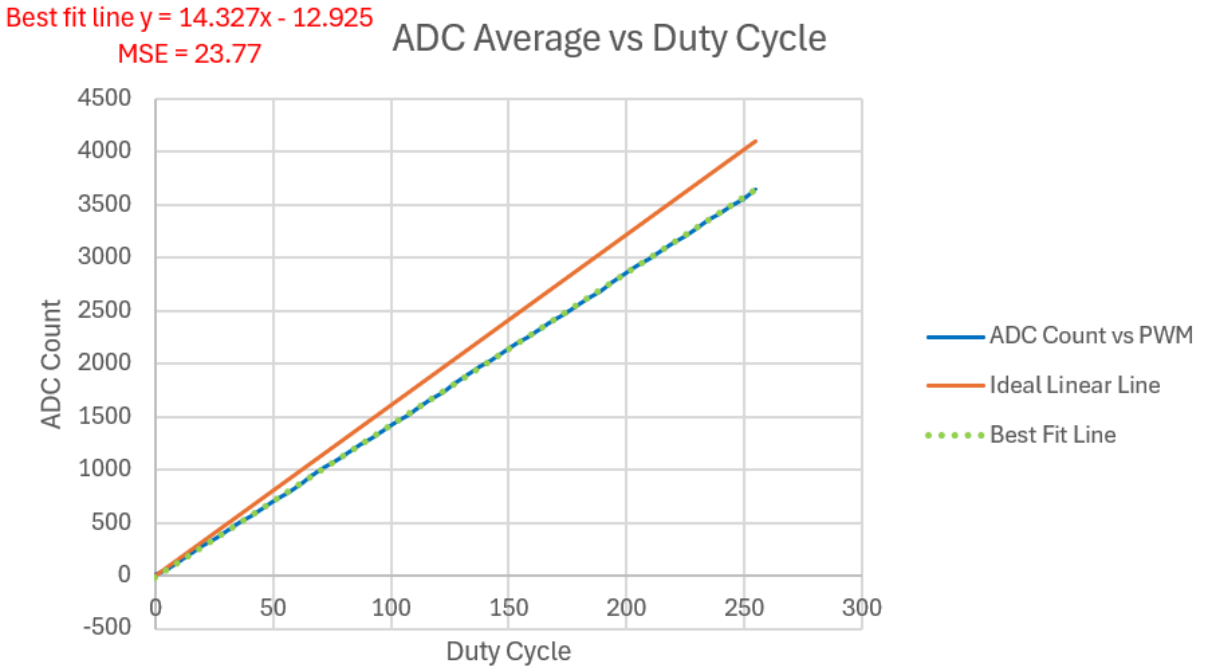

Figure 5: Part 3 Graph 'ADC Average vs Duty Cycle'.

```
void PWMAdjust(int32_t percent){
```

```
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, (uint32_t)
    (percent*254)/100);
}

void PWMStart(void){
    HAL_TIM_Base_Start(&htim1);
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
}
uint32_t start = HAL_GetTick();
uint32_t start2 = HAL_GetTick();
float movingAverageArrayDAC[64] = {0};
float movingAverageDACValue = 0;
int duty = 0;
void movingAverageDAC(int inputADC){
    int i = 0;
    movingAverageDACValue = 0;
    while (i < 63){
        movingAverageArrayDAC[i]= movingAverageArrayDAC[i+1];
        movingAverageDACValue += movingAverageArrayDAC[i];
        i++;
    }
    movingAverageDACValue += inputADC;
    movingAverageArrayDAC[63] =  inputADC;
    movingAverageDACValue = movingAverageDACValue/64;
}
while (1){
    PWMStart();
    if ((HAL_GetTick()- start)>100){
        start = HAL_GetTick();
        HAL_Delay(10);
        int i =0;
        while(i<64){
            int ADCValueForDAC = getCalibratedADCReading(&hadc2,
            ADC_CHANNEL_1, ADC_SAMPLETIME_480CYCLES);
            movingAverageDAC(ADCValueForDAC);
            i++;
        }
        printf("The Duty Value is: %d\n",duty);
```

```
        printf("The ADC Count is: %.2f\n",movingAverageDACValue);
        if(duty < 100){
            duty += 2;}
        PWMAdjust(duty);
    }
}
```

Figure 6: Part 3 Code.