ESET 369 Lab 9 Report

Student name	Kyle Rex and Blake Brzezinski
Group number	41
Lab Section	501
Lab session date	April 21, 2025
Lab instructor	Xin Zhuang

INTRODUCTION

This lab focuses on configuring and programming a Raspberry Pi Pico microcontroller to perform two distinct embedded system tasks using MicroPython. The objective of System A is to utilize analog-to-digital conversion to interpret voltage levels from a potentiometer and internal temperature sensor, subsequently controlling an RGB LED based on the measured voltage. System B extends functionality by using digital button inputs and a potentiometer to control a motor's speed through pulse-width modulation (PWM). Each system required correct hardware interfacing with the BH board and precise implementation of MicroPython code. The lab emphasizes practical applications of embedded systems concepts such as ADC conversions, GPIO configurations, and PWM motor control, reinforcing skills essential for real-world electronics design and programming.

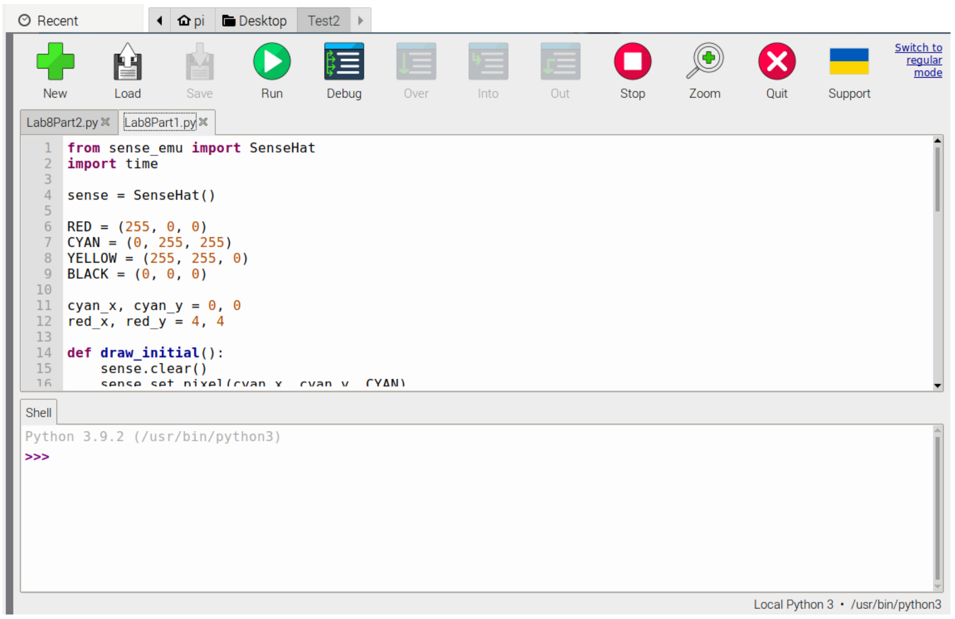
SYSTEM A

The purpose of System A is to correctly configure a physical Raspberry Pi (RPI) Pico to control an RGB LED based on an ADC value (voltage value) being controlled by a potentiometer on the BH board using a Thonny IDE micro-python code sequence while printing out the voltage value being read from the potentiometer and the internal temperature of the RPI Pico in the shell sub-window. ADC(0) is connected to the potentiometer and must be converted to a voltage value (between its min of 0V to its max of 3.3V) while being printed out on the shell sub-window updating at a reasonable speed. ADC(4) is connected to the internal temperature sensor of the RPI Pico and must be converted to a degree value in Fahrenheit while being printed out on the shell sub-window updating at a reasonable speed. The RGB LED will be a different color based on the converted ADC(0) voltage value from the potentiometer. For a voltage value less than 1V it will remain off (Black), between 1V and 2V it will be red, between 2V and 3V it will be green, and greater than 3V it will be blue.

The first step in addressing this problem is making the physical board connections between the BH board and the Raspberry Pi (RPI) Pico. This can be done by connecting six specific pins together using male-to-female/male-to-male jumper wires and checking that all the DIP switches on the BH board are in the correct configuration. P38 of the RPI Pico must be connected to the top left pin, of the left set of pins, of the BH board where the microcontroller usually sits. P36 of the RPI Pico must be connected to the top right pin, of the right set of pins, of the BH board where the microcontroller usually sits. The GP26 pin P31 of the RPI Pico is connected to the potentiometer and is used to read a voltage value between 3.3V and 0V. The GP15 pin P20, GP16 pin P21, and

GP17 pin P22 of the RPI Pico are connected to, and control, the red, green, and blue hues of the RGB LED respectively.

Using the code sequences found in: *Learning Embedded Systems with MSP430 FRAM microcontrollers* by B. Hur, as references the code sequence used for this system was created. One part of the system was coded and tested at a time to ensure that there were no issues with the code or the hardware. The MicroPython code implemented for this system is designed to interface the Raspberry Pi Pico with an RGB LED and two analog sensors via its onboard ADC channels. Specifically, ADC(0) reads voltage from a potentiometer connected to GP26, while ADC(4) captures data from the Pico’s internal temperature sensor. The code begins by importing necessary modules and initializing the GPIO pins 15, 16, and 17 as outputs to control the red, green, and blue components of the RGB LED. Inside the infinite loop, the raw ADC values are read from channels 0 and 4. The potentiometer value (ADC0) is converted to a voltage by scaling the 16-bit raw value (0–65535) to the Pico’s reference voltage (3.3V). This voltage is used to determine the LED color: the LED remains off (all components HIGH) if voltage is below 1V, displays red (only red LOW) between 1V–2V, green between 2V–3V, and blue above 3V. Simultaneously, the raw temperature sensor data (ADC4) is converted to Celsius using the manufacturer’s conversion formula and then converted to Fahrenheit. All voltage and temperature readings are printed to the Thonny IDE shell window for real-time monitoring. A 0.5-second delay between iterations ensures the loop updates at a readable pace without overwhelming the display. An example of the Thonny IDE coding environment used to run this code sequence can be seen in Figure 1. The measured ADC raw values for ADC(0) and ADC(4) can be seen in Table 1 and Table 2.



```
from sense_emu import SenseHat
import time

sense = SenseHat()

RED = (255, 0, 0)
CYAN = (0, 255, 255)
YELLOW = (255, 255, 0)
BLACK = (0, 0, 0)

cyan_x, cyan_y = 0, 0
red_x, red_y = 4, 4

def draw_initial():
    sense.clear()
    sense.set_pixel(cyan_x, cyan_y, CYAN)
```

```
Python 3.9.2 (/usr/bin/python3)
>>>
```

Figure 1: Thonny IDE Python Coding Environment

Table 1: System A ADC(0) Measurement / Potentiometer

	ADC_RAW (u16)
Case 1 (1 V)	20000

Case 2 (2 V)	40000

Table 2: System A ADC(4) Measurement

	ADC_RAW (u16)	Converted Temperature (Degrees F)
Case 3	41258	74.7813

SYSTEM B

The purpose of System B is to correctly configure a physical Raspberry Pi (RPI) Pico to control the speed of a motor based on button and potentiometer inputs on the BH board using a Thonny IDE micro-python code sequence. When the up button is pressed the motor spins at a fast speed. When the left button is pressed the motor spins at a slow speed. When the down button is pressed the motor stops spinning. If the right button is pressed the motor should spin at a speed relative to the position of the potentiometer. If the potentiometer is turned where more voltage is allowed to pass through, then the motor should spin faster. If the potentiometer is turned where less voltage is allowed through, then the motor should spin slower.

The first step in addressing this problem is making the physical board connections between the BH board and the Raspberry Pi (RPI) Pico. This can be done by connecting eight specific pins together using male-to-female/male-to-male jumper wires and checking that all the DIP switches on the BH board are in the correct configuration. P38 of the RPI Pico must be connected to the top left pin, of the left set of pins, of the BH board where the microcontroller usually sits. P36 of the RPI Pico must be connected to the top right pin, of the right set of pins, of the BH board where the microcontroller usually sits. The GP26 pin P31 of the RPI Pico is connected to the potentiometer and is used to read a voltage value between 3.3V and 0V. The GP10 pin P14, GP11 pin P15, GP12 pin P16, and GP13 pin P17 of the RPI Pico are connected to, and control, the up, down, left, and right buttons respectively. The 9 Volt power supply positive and negative pins connect to the motor supplying it with the required voltage to operate and the GP14 pin P19 of the RPI Pico controls the motor speed.

Using the code sequences found in: *Learning Embedded Systems with MSP430 FRAM microcontrollers* by B. Hur, as references the code sequence used for this system was created. One part of the system was coded and tested at a time to ensure that there were no issues with the code or the hardware. The MicroPython code for this system enables the Raspberry Pi Pico to control the motor’s speed using four digital button inputs and an analog input from a potentiometer. The motor is driven through PWM (Pulse Width Modulation) using GPIO pin 14, which is initialized with a frequency of 50 Hz. Four GPIO pins, 10, 11, 12, and 13, are configured as digital inputs connected to the up, down, left, and right buttons, respectively. The analog signal from the potentiometer is read through ADC(0) on GP26. Within the main loop, the code checks the state of each button. If the down button is pressed (logic low), the motor stops by setting the PWM duty cycle to 0. If the left button is pressed, the motor runs at a slow speed with a low PWM duty cycle. When the up button is pressed, a very high duty cycle is applied to drive the motor at full or near-full speed. If the right button is pressed, the code enters a secondary loop where it continuously reads the potentiometer’s value, maps it to a duty cycle range, and applies it to the motor. This allows dynamic, real-time speed control based on the potentiometer's position. A short delay of 0.1 seconds at the end of the main loop helps debounce the button inputs and manage CPU load.

CONCLUSION

The successful implementation of Systems A and B demonstrated effective use of the Raspberry Pi Pico for analog input processing and digital output control in embedded applications. System A showcased real-time monitoring and color-coded LED feedback based on voltage and temperature inputs, while System B implemented dynamic motor control using button states and a potentiometer input. These tasks strengthened understanding of interfacing hardware components with microcontrollers and applying MicroPython to solve real-time control problems. Overall, the lab reinforced key embedded systems techniques including ADC usage, PWM signal generation, and system debugging, providing valuable experience in integrating hardware and software within microcontroller environments.

REFERENCES

B. Hur, *Learning Embedded Systems with MSP430 FRAM microcontrollers*, 2nd ed., 2023.

K. Rex, *ESET 369 Lab Report 8*, 2025.

K. Rex, *ESET 369 Lab Report 7*, 2025.

K. Rex, *ESET 369 Lab Report 6*, 2025.

K. Rex, *ESET 369 Lab Report 5*, 2025.

K. Rex, *ESET 369 Lab Report 4*, 2025.

K. Rex, *ESET 369 Lab Report 3*, 2025.

K. Rex, *ESET 369 Lab Report 2*, 2025.

K. Rex, *ESET 369 Lab Report 1*, 2025.