2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #37487

Raspberry Pi Pico as an IoT Device

Mr. David R. Loker, Pennsylvania State University, Behrend College

David R. Loker received the M.S.E.E. degree from Syracuse University in 1986. In 1984, he joined General Electric (GE) Company, AESD, as a design engineer. In 1988, he joined the faculty at Penn State Erie, The Behrend College. In 2007, he became the Chair of the Electrical and Computer Engineering Technology Program. His research interests include wireless sensor networks, data acquisition systems, and communications systems.

Raspberry Pi Pico as an IoT Device

Abstract

The Raspberry Pi Pico is an inexpensive embedded processor board that can be used for introductory programming and embedded systems courses in engineering and engineering technology programs. For Internet of Things (IoT) applications, the Pi Pico contains a system-on-a-chip (SoC) device (the RP2040 microcontroller) that is capable of performing the responsibilities of a computer on a single chip. Some key features of the Pi Pico include digital peripherals (e.g., 2 SPI, 2 I2C, 2 UART, and 16 PWM), 23 GPIO pins for digital I/O, 3 ADC inputs, and an on-board LED and temp sensor. Although programming the Pi Pico can be performed in C, MicroPython, which is a subset of the Python standard library, is optimized to run on a variety of embedded microcontrollers including the Pi Pico. Thonny is a free download software development environment for writing Python code and downloading it to the Pi Pico.

Applications for the Pi Pico are broad enough to encompass both electrical and computing disciplines. There are several goals for this paper. For embedded courses, control of the I/O pins on the Pi Pico will be shown. This includes digital I/O, analog input, PWM output, UART, SPI, and I2C. Devices interfaced to the Pi Pico include an analog temperature sensor, a serial LCD display, a digital-to-analog converter, and an accelerometer. A WiFi device will also be interfaced to the Pi Pico to show the capabilities for embedded IoT applications. Examples of student assessments will be shown. For ABET, it will be shown how these projects can be used to assess student outcomes.

Introduction

In engineering and engineering technology programs, there are a variety of programming and embedded systems courses. C/C++ programming is often used as a primary component to these courses. Examples of devices used within embedded hardware and software courses include the Programmable System-on-Chip (PSoC 5LP) and the BeagleBone Black (BBB) [1-3]. Alternatively, MicroPython is a subset of the Python standard library, and it is optimized to run on a variety of microcontrollers for embedded applications [4]. One such device is the Raspberry Pi Pico [5]. The Pi Pico is an inexpensive embedded processor board that can be used in a variety of courses.

Embedded Courses

In electrical and computing disciplines, curriculum is set up so that programming and embedded systems are taught through a variety of courses. These courses can include programming (structured procedural design and object-oriented design), Digital Design, and microprocessors and microcontrollers (introduction, intermediate, and advanced). Additionally, systems-oriented courses (e.g., Communication Systems, Control Systems, Senior-level Project-based courses, etc.) typically include embedded systems as an integral component.

C/C++ programming is often used with embedded systems courses as the core component. An introductory microprocessors and microcontrollers course may utilize C programming on an "Arduino" type computer board. Students learn 'C' programming, and with the availability of

API/library calls, they can ignore the low-level register programming. For intermediate embedded courses, students can design software that interfaces directly to the hardware. Course topics can include interrupts, timers, ADC and DAC interfacing, and digital filtering.

Due to its open-source popularity, Python can be used in curriculums to supplement C/C++ course content. Examples include utilizing MicroPython on the Digi XBee3 module and on the Raspberry Pi Pico [4-5]. Python and MicroPython are very similar, the main difference being that MicroPython does not include the data analysis and graphing libraries of standard Python. However, students will learn to use hardware libraries associated with embedded programming.

There are several goals for this paper. For embedded courses, control of the I/O pins on the Pi Pico will be shown. This includes digital I/O, analog input, PWM output, UART, SPI, and I2C. Devices interfaced to the Pi Pico include an analog temperature sensor, a serial LCD display, a digital-to-analog converter, and an accelerometer. A WiFi device will also be interfaced to the Pi Pico to show the capabilities for embedded IoT applications. Thonny, the Python IDE, will be used for software development [6].

Thonny IDE

An IDE for developing Python code is called Thonny [6]. Thonny allows a Windows PC to seamlessly integrate with a Raspberry Pi-Pico. Thonny's console interface can display a program's text output, and Thonny has a mechanism to single step and debug Python programs running on the Pico. During development, programs can be saved onto the PC or the Pi Pico. It has a script area for writing code and a Python Shell for running programs. A simple example is shown in Figure 1. This program prints the value of a variable called "resistor" to the console.

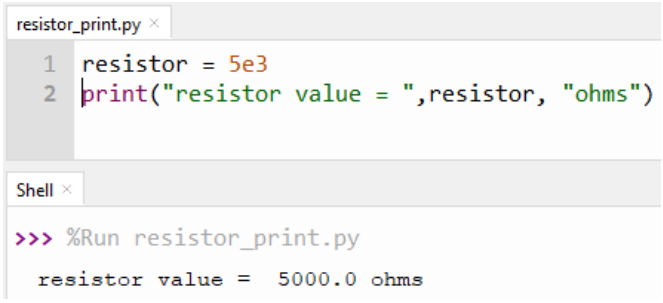

Figure 1. Python print command

MicroPython

MicroPython is a subset of the Python standard library, and it is optimized to run on microcontrollers (e.g., Pi Pico, etc.) for embedded applications [7-8]. The interface to its hardware I/O is accomplished via functional specific libraries. Some examples include Bluetooth, machine, and network communication. The machine module implements specific classes of functions related to the hardware on embedded boards. Examples of various classes include:

- Pin for I/O control pins
- ADC for analog to digital conversion

- PWM for pulse width modulation
- UART for serial communication
- SPI for serial peripheral interface protocol
- I2C for a two-wire serial protocol

Raspberry Pi Pico

The Raspberry Pi Pico board has the following features [9-10].

- 264 KB SRAM in six independent banks
- No internal Flash or EEPROM memory (after reset, the bootloader loads firmware from either the external flash memory or USB bus into internal SRAM)
- QSPI bus controller, which supports up to 16 MB of external Flash memory
- On-chip programmable LDO to generate core voltage
- 2 on-chip PLLs to generate USB and core clocks
- 30 GPIO pins, of which 4 can optionally be used as analog inputs
- Processor can be clocked up to 133 MHz

Figure 2 contains the pinouts for the Raspberry Pi Pico board.

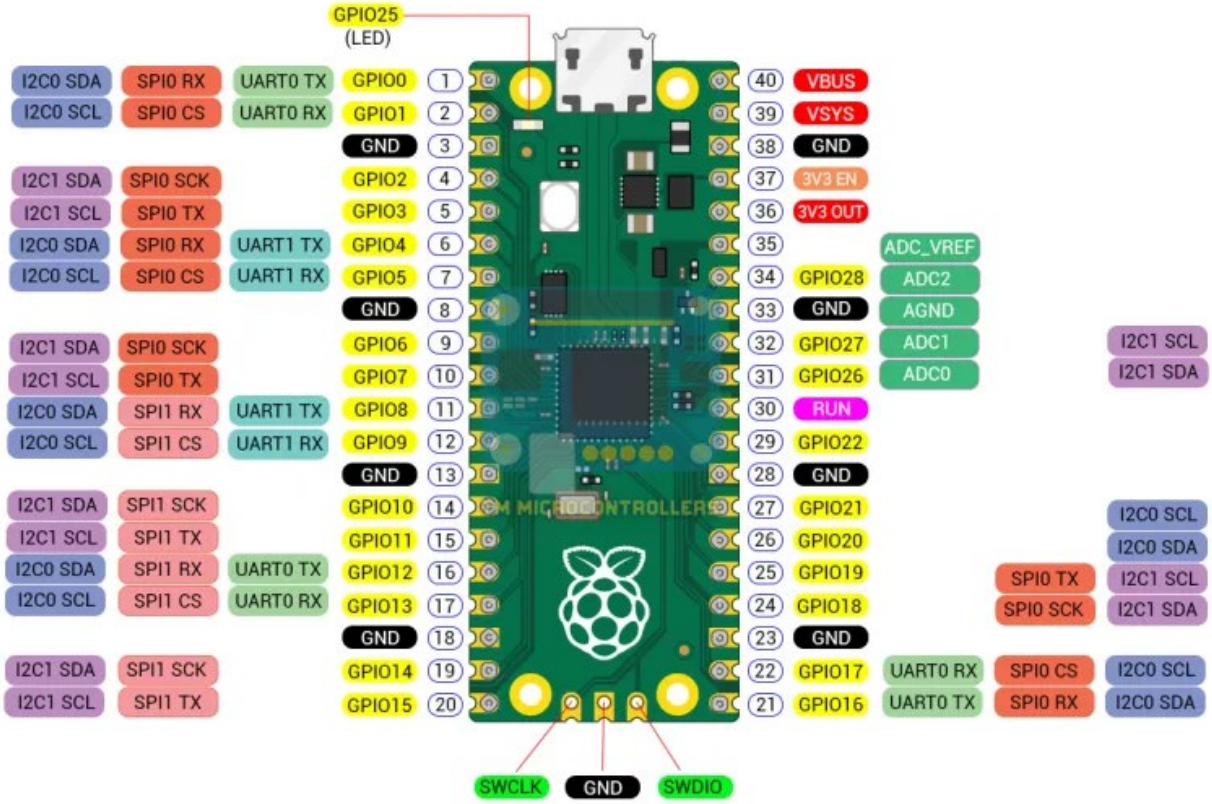

Figure 2. Pi Pico Pinout [9]

Digital I/O on the Pi Pico

Each of the Pi Pico I/O pins are identified by a GPIO number. On the bottom of the board, each pin has silk screening that details this number (along with power and ground). One of the GPIO pins is special (GP-25), since it connects to the on-board LED which is available to be used by the program. All I/O pins can be used as digital I/O pins, and a subset of these pins can be configured for PWM (pulse-width modulation), A-to-D inputs, UART, and SPI communication pins. These pinouts are shown in Figure 2.

Digital output pins can drive up to 3.3 volts at 8 milliamps. Input pins also have programmable pull-up and pull-down resistors. To configure a pin, the machine library is imported which contains the functions needed to configure and interface to the I/O pins. Examples are shown in Figure 3.

digital_i_o.py

```
1  import machine
2
3  pin1 = machine.Pin(1, machine.Pin.OUT)
4                      # configures GP01 as a digital output pin
5                      # and the variable pin1 is then used to set
6                      # ...the pin to a logic 1 or logic 0
7
8  pin1.value(1)       # sets the pin to a logic 1
9  pin1.value(0)       # sets the pin to a logic 0
10
11 pin2 = machine.Pin(2, machine.Pin.IN)
12                     # configures GP02 as a digital input pin
13                     # and the variable pin2 is used to read
14                     # ...the logic state of the input
15
16 state = pin2.value()# returns 0 or 1
17
18 pin2 = machine.Pin(2, machine.Pin.IN, machine.Pin.PULL_UP)
19                     # configures GP02 as a digital
20                     # input with internal pull-up
21
22 pin2 = machine.Pin(2, machine.Pin.IN, machine.Pin.PULL_DOWN)
23                     # configures GP02 as a digital
24                     # input with internal pull-down
```

Figure 3. Digital I/O Code Examples.

GPIO25 connects to the on-board LED. An example of blinking this LED on and off every 0.5 seconds is shown in Figure 4.

blinking_LED.py

```
1  import machine
2  import time
3
4  pin25 = machine.Pin(25, machine.Pin.OUT)
5
6  while True :
7      pin25.value(1)  # turns on the led
8      time.sleep(.5)
9      pin25.value(0)  # turns off the led
10     time.sleep(.5)
11
```

Figure 4. Blinking LED Code.

Analog Input

The Pi-Pico has 3 external analog-to-digital converter (ADC) inputs (pins 31, 32 and 34) and can monitor any voltage from 0 to 3.3 volts. Each connects to a 12-bit ADC. The 12-bit value is converted to 16 bits (i.e., the ADC reads 0-4095 but represents it as 0-65535). An example is an LM35 temperature sensor connected to an ADC input [11]. The temperature coefficient for the LM35 is 10mV/ºC. The Pi Pico reads the analog input from the LM35 on ADC2 (pin 34). The program is shown in Figure 5.

temp_C_read.py

```
1  import machine
2
3  pin34 = machine.ADC(28)
4  data = machine.ADC.read_u16(pin34)
5  volts = 3.3 * data / 65535.0
6  print("Input is ", volts, " volts \n")
7  temp = volts / 0.010
8  print("Temp in degrees C is ", temp)
9
```

Shell

```
>>>

MicroPython v1.19.1 on 2022-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
  Input is  0.2570611  volts

  Temp in degrees C is  25.70611
```

Figure 5. LM35 Temp Sensor Code and Output.

PWM Output

All of the digital output pins can also be configured for pulse-width modulation (PWM). This

means that you can configure the pin to output a square wave at a specific frequency and duty cycle. See Figure 2 for a full listing of available I/O pins.

An application would be varying the brightness of an LED by changing the duty cycle of a square wave. The PWM frequency should be set high enough (e.g., 5 KHz) to eliminate "flicker" at low duty cycles. An example of controlling the brightness of the on-board LED (GPIO25) is shown below in Figure 6. In this example, the duty cycle is ramped up linearly from 2% to 65.5% in steps of 1% (delays 2mS between steps), pauses for 2 seconds, then ramps down linearly from 65.5% to 2%.

pwm_LED_DC.py

```
import machine
import time

#pin25 = machine.Pin(25, machine.Pin.OUT)

# use the machine.PWM function to select a specific pin
# to produce a PWM output waveform
pin25 = machine.PWM(machine.Pin(25, machine.Pin.OUT))

# select 5000 Hz output
pin25.freq(5000)

# increase brightness
for dc in range (2,655) :
    pin25.duty_u16(dc * 100)
    time.sleep_us(2000)

# pause for 2 seconds
time.sleep(2)

# decrease brightness
for dc in range (655,2,-1) :
    pin25.duty_u16(dc * 100)
    time.sleep_us(2000)
```

Figure 6. Code for On-board LED Brightness Control using PWM.

UART/RS-232

The Pi Pico has two UART's, which can be mapped to a variety of GPIO pins. See Figure 2 for a full listing of available I/O pins.

An application would be interfacing to a serial LCD display [12]. From the datasheets, the manufacturer lists approximately 25 commands that the device can perform. Each command begins with the same prefix byte (0xFE), followed by 1 or 2 additional bytes. By storing the command sequences in byte arrays, a single instruction can be used to send commands to the display. Some LCD write examples are shown below in Figure 7.

```
    Cursor_Home = bytearray( [0xFE, 0x46] )      # move cursor home
    Clear_Display = bytearray( [0xFE, 0x51] )    # clears the display
    Row2_Col1 = bytearray( [0xFE, 0x45, 0x40] )  # moves the cursor to line 2, col 1

    lcd.write(Clear_Display)
    lcd.write(Cursor_Home)
    lcd.write(Row2_Col1)
    lcd.write('Hello')       # ASCII text can be directly written
```

Figure 7. Code for the serial LCD.

An example would be to display the temperature from the on-board temp sensor to both the console and LCD display. The on-board temperature sensor (base-emitter junction of a biased bipolar diode) is connected to the internal analog-to-digital converter (ADC4). The ADC has a full-scale voltage of 3.3V with 12 bits of resolution. From the datasheets, Vbe = 0.706V at 27 degrees C, with a slope of -1.721mV/degree. Hardware connections would be as shown in Figure 8. An image of the LCD display output is shown in Figure 9. Code for displaying temperature to both the console and LCD is shown in Figure 10.

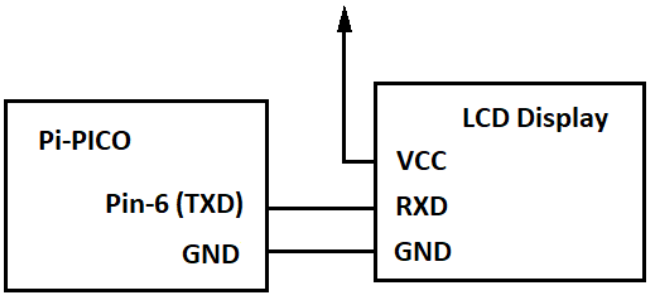

Figure 8. Hardware connections for the serial LCD.

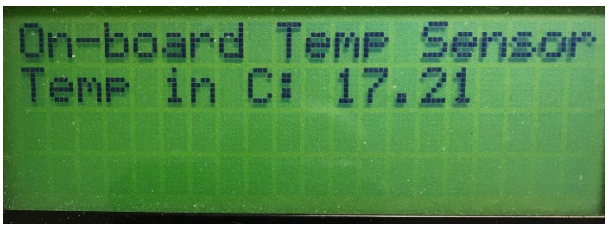

Figure 9. On-board Temp Sensor Serial LCD Display.

LCD_write_temp.py

```
from machine import UART, Pin, ADC
from time import sleep
#
# List of LCD commands as an array of bytes.
# returns the cursor to line 1 column 1
home = bytearray([0xFE, 0x46])
# clears the entire display and places the cursor at line 1 column 1
clear = bytearray([0xFE, 0x51])
# moves the cursor to line 2, col 1
line2 = bytearray([0xFE, 0x45, 0x40])
# sets the display contrast where the contrast setting can be between 1 and 50
contrast = bytearray([0xFE, 0x52, 50])
#
# uart1 assigned to variable "lcd"
lcd = UART(1, baudrate=9600, tx=Pin(4), rx=Pin(5))
#
# Initialize LCD screen.
# sets LCD to max contrast
lcd.write(contrast)
# clears entire LCD screen
lcd.write(clear)
lcd.write(home)
lcd.write('On-board Temp Sensor')
#
sensor_temp = ADC(4) # On-board temp sensor is connected to ADC4
conv_factor = 3.3 / 65535 #3.3V full-scale, read as 16 bits
while True:
    read = sensor_temp.read_u16() * conv_factor # Convert to voltage
    # Convert to temp in deg C
    temp = 27 - (read - 0.706)/0.001721
    print(temp)
```

Shell

```
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

  17.68152
  17.21338
  17.68152
  17.21338
  17.21338
  17.21338
```

Figure 10. On-board Temp Sensor Code and Output.

SPI

The Serial Peripheral Interface (SPI) refers to a 4-wire serial interface which includes the signals: MOSI or SPI-TX, MISO or SPI-RX, SCLK or SPI-SCK and CS. See Figure 2 for a full listing of available I/O pins. SPI initialization (from the online documentation) is found from the online SPI Python library [13].

An application would be interfacing an MCP4901 DAC to the Pi Pico. Connections would be as follows. Datasheets for the MCP4901 DAC are available online [14].

- Pi Pico pin 14 (SPI1 SCK – GPIO10) to 4901 pin 3 (SCK)
- Pi Pico pin 15 (SPI1 TX – GPIO11) to 4901 pin 4 (SDI)
- Pi Pico pin 17 (SPI1 CS – GPIO13) to 4901 pin 2 ($\overline{CS}$)

An example of a program is shown below in Figure 11. With VCC=3.3V, the output shown on a DMM is in Figure 12.

spi_dac_v1.py

```
from machine import Pin, SPI
#
data = bytearray([0x78, 0x00]) # DAC output is 1/2 of VREF=VCC
# Use SPI 1, bit rate=1Mbps
# sck=GP10, mosi(tx)=GP11, miso(rx)=GP12, csn=GP13
spi = SPI(1, baudrate=1000000, polarity=0, phase=0, bits=8, sck=Pin(10), mosi=Pin(11), miso=Pin(12))
cs = Pin(13, mode=Pin.OUT)
cs.value(0) # Enable chip select (active low)
spi.write(data)
cs.value(1)
```

Figure 11. SPI DAC Code.

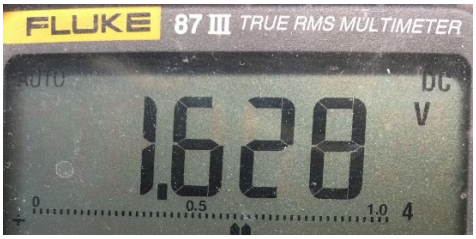

Figure 12. SPI DAC DMM Display.

I2C

I2C is similar to SPI. However, the same data is sent and received to the peripheral using only two wires (SDA and SCL). The SDA wire has both the transmit and receive data, and it moves data to/from the peripheral using a half-duplex mode. There is no chip select (CS) as the components ID is placed within the message. Like the other protocols, specific pins are allocated for this type of transmission. See Figure 2 for a full listing of available I/O pins. I2C initialization (from the online documentation) is found from the online I2C Python library [15].

An application would be interfacing an ADXL345 accelerometer to the Pi Pico. Connections would be as follows. Datasheets for the ADXL345 using a breakout board are available online [16].

- Pi Pico pin 21 (I2C0 SCL – GPIO16) to 345 breakout board (SDA)
- Pi Pico pin 22 (I2C0 SCL – GPIO17) to 345 breakout board (SCL)
- 345 breakout board (CS) to VCC
- 345 breakout board (SDO) to GND

An example of a portion of the program code and the output results are shown in Figures 13 and

14. Additional details regarding the program can be found in this reference [4].

i2c_accel_v1.py

```
while True:
    # This block of code gets the byte data of X axis
    RawXdata = get_data(ADXL345_ADDR, X0_REG, X1_REG)
    GXdata = convert_G(RawXdata)
    X = .8 * X + .2 * GXdata

    # This block of code gets the byte data of Y axis
    RawYdata = get_data(ADXL345_ADDR, Y0_REG, Y1_REG)
    GYdata = convert_G(RawYdata)
    Y = .8 * Y + .2 * GYdata

    # This block of code gets the byte data of Z axis
    RawZdata = get_data(ADXL345_ADDR, Z0_REG, Z1_REG)
    GZdata = convert_G(RawZdata)
    Z = .8 * Z + .2 * GZdata

    print("X Data: %2.2f " % X)
    print("Y Data: %2.2f " % Y)
    print("Z Data: %2.2f " % Z)
    print("-+-+-+-+-+-+-+-+-+-")
    time.sleep(0.5)
```

Figure 13. ADXL345 Results Code.

Shell

```
X Data: 0.02
Y Data: -0.07
Z Data: 0.77
-+-+-+-+-+-+-+-+-+-
X Data: 0.02
Y Data: -0.07
Z Data: 0.79
-+-+-+-+-+-+-+-+-+-
X Data: 0.02
Y Data: -0.08
Z Data: 0.81
-+-+-+-+-+-+-+-+-+-
X Data: 0.02
Y Data: -0.08
Z Data: 0.83
-+-+-+-+-+-+-+-+-+-
X Data: 0.03
Y Data: -0.09
Z Data: 0.84
-+-+-+-+-+-+-+-+-+-
```

Figure 14. ADXL345 Console Output Results.

WiFi Module for Web Server

An inexpensive WiFi module (ESP8266) with an integrated TCP/IP stack is available for embedded applications [17]. The ESP8266 WiFi module can be connected to an ESP-01 breakout board for interfacing to the Pi Pico. Connections would be as follows.

- Pi Pico pin 2 (UART0 RX – GPIO1) to ESP-01 breakout board (TXD)
- Pi Pico pin 1 (UART0 TX – GPIO0) to ESP-01 breakout board (RXD)
- Pi Pico pin 36 (3.3V) to ESP-01 breakout board (VCC)
- Pi Pico pin 36 (3.3V) to ESP-01 breakout board (CHPD)
- Pi Pico pin 3 (GDN) to ESP-01 breakout board (GND)

A tutorial showing the above connections along with code to configure the ESP8266 as a TCP web server is available online [18].

An example of a program is shown below in Figures 15(a-d). This program reads the voltage from an LM35 temperature sensor, converts the voltage to temperature in degrees C, and serially transmits (using UART0) the data to the ESP8266 WiFi Module which is configured as a TCP server. Once the SSID and Password for the access point are provided, the local IP address for the WiFi module is assigned. Then, using this local IP address, a web browser is opened to access the web server. The web server console output and an image of the web browser output are shown in Figures 16 and 17.

web_server.py

```
1  import uos
2  import machine
3  from machine import ADC
4  import utime, time
5
6  recv_buf="" # receive buffer global variable
7  #data collection variables
8  analog_pin = machine.ADC(28)
9
10 print()
11 print("Machine: \t" + uos.uname()[4])
12 print("MicroPython: \t" + uos.uname()[3])
13
14 uart0 = machine.UART(0, baudrate=115200)
15 print(uart0)
16
```

Figure 15(a). Web Server Initialization Code.

web_server.py

```
17 def Rx_ESP_Data():
18     recv=bytes()
19     while uart0.any()>0:
20         recv+=uart0.read(1)
21     res=recv.decode('utf-8')
22     return res
23 
24 def Connect_WiFi(cmd, uart=uart0, timeout=3000):
25     print("CMD: " + cmd)
26     uart.write(cmd)
27     utime.sleep(7.0)
28     Wait_ESP_Rsp(uart, timeout)
29     print()
30 
31 def Send_AT_Cmd(cmd, uart=uart0, timeout=3000):
32     print("CMD: " + cmd)
33     uart.write(cmd)
34     Wait_ESP_Rsp(uart, timeout)
35     print()
36 
37 def Wait_ESP_Rsp(uart=uart0, timeout=3000):
38     prvMills = utime.ticks_ms()
39     resp = b""
40     while (utime.ticks_ms()-prvMills)<timeout:
41         if uart.any():
42             resp = b"".join([resp, uart.read(1)])
43     print("resp:")
44     try:
45         print(resp.decode())
46     except UnicodeError:
47         print(resp)
48 
```

Figure 15(b). Web Server Functions Code.

web_server.py

```
49 Send_AT_Cmd('AT\r\n')          #Test AT startup
50 Send_AT_Cmd('AT+GMR\r\n')      #Check version information
51 #Send_AT_Cmd('AT+CIPSERVER=0\r\n')      #Check version information
52 Send_AT_Cmd('AT+RST\r\n')      #Check version information
53 Send_AT_Cmd('AT+RESTORE\r\n')  #Restore Factory Default Settings
54 Send_AT_Cmd('AT+CWMODE?\r\n')  #Query the WiFi mode
55 Send_AT_Cmd('AT+CWMODE=1\r\n') #Set the WiFi mode = Station mode
56 Send_AT_Cmd('AT+CWMODE?\r\n')  #Query the WiFi mode again
57 #Send_AT_Cmd('AT+CWLAP\r\n', timeout=10000) #List available APs
58 Connect_WiFi('AT+CWJAP="[redacted]"[redacted]"\r\n', timeout=5000)
59 Send_AT_Cmd('AT+CIFSR\r\n')    #Obtain the Local IP Address
60 utime.sleep(3.0)
61 Send_AT_Cmd('AT+CIPMUX=1\r\n')    #Obtain the Local IP Address
62 utime.sleep(1.0)
63 Send_AT_Cmd('AT+CIPSERVER=1,80\r\n')    #Obtain the Local IP Address
64 utime.sleep(1.0)
65 print ('Starting connection to ESP8266...')
```

Figure 15(c). Web Server WiFi Module Configuration Code.

web_server.py

```
while True:
    #reset receive buffer and check for connection
    res =""
    res=Rx_ESP_Data()

    reading = analog_pin.read_u16()
    volt = (reading*3.3)/65535

    print("Input is ", volt, " volts \n")
    temp = volt / 0.010
    print("Temp in degrees C is ", temp, "  \n")
    temp_str = str(temp)

    time.sleep(2)

    if '+IPD' in res: # if the buffer contains IPD(a connection), then respond with
        id_index = res.find('+IPD')
        print("resp:")
        print(res)
        connection_id =  res[id_index+5]
        print("connectionId:" + connection_id)
        print ('! Incoming connection - sending webpage')
        uart0.write('AT+CIPSEND='+connection_id+',200'+'\r\n')  #Send a HTTP respon
        utime.sleep(1.0)
        uart0.write('HTTP/1.1 200 OK'+'\r\n')
        uart0.write('Content-Type: text/html'+'\r\n')
        uart0.write('Connection: close'+'\r\n')
        uart0.write(''+'\r\n')
        uart0.write('<!DOCTYPE HTML>'+'\r\n')
        uart0.write('<html>'+'\r\n')
        uart0.write('<body><center><h1>Temp Sensor Project</h1></center>'+'\r\n')
        uart0.write('<center><h2>LM35 Temp in deg C: ')
        uart0.write(temp_str)
        uart0.write('</h2></center>'+'\r\n')
        uart0.write('</body></html>'+'\r\n')
        utime.sleep(4.0)
        Send_AT_Cmd('AT+CIPCLOSE='+ connection_id+'\r\n') # once file sent, close c
        utime.sleep(2.0)
        recv_buf="" #reset buffer
        print ('Waiting For connection...')
```

Figure 15(d). Web Server Data Code.

```
Shell
Input is  0.2433646  volts

Temp in degrees C is  24.33646

resp:
0,CONNECT

+IPD,0,455:GET / HTTP/1.1
Host: 192.168.50.144
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.
36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36
Accept: text/html,app
connectionId:0
! Incoming connection - sending webpage
CMD: AT+CIPCLOSE=0
```

Figure 16. Web Server Console Output.

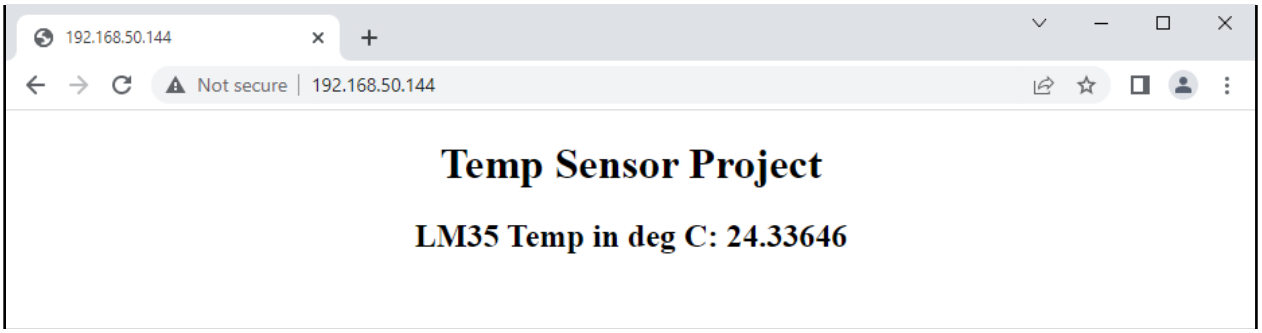

Figure 17. Web Browser Image.

Internet of Things (IoT) Projects

Technology used in IoT projects can include devices for:

- Wireless applications (ZigBee, Bluetooth, WiFi, etc.)
- Sensing applications (temperature, pressure, moisture)
- Control applications (motor)
- Displays (LCD, LED OLED)

There are many possible examples of student IoT projects for a 2-3 week lab suitable for embedded systems courses in engineering and engineering technology programs [5]. These projects required MicroPython with the Pi Pico. Project deliverables included the following.

- Executive summary of the results (Word file):
  - Written description (1-2 paragraphs) of the hardware design (also include OrCAD PSpice schematics)
  - Written description (1-2 paragraphs) of the software design (also include a flowchart created in Word, ppt, or other s/w)

    - Testing procedure (numbered step by step testing procedure for each engineering requirement)
    - Results (1-2 paragraphs)
    - Signed academic integrity statement
- 2-3 minute video (posted on YouTube) demonstrating successful completion of the lab project
- Upload to Canvas the following:
    - Word file that contains the executive summary
    - All software source code
    - Link to video

Some additional possible examples of student projects are as follows.

- Automated plant monitoring and control system
- Concussion Protocol Color Test System
- Remote weather station
- Home Security System
- Home Automation System
- Remote vehicle speed monitoring

Automated Plant Monitoring and Control System

The overall objective is to automate a plant-growing system that can be sustained all year round.

For this project, several engineering requirements were as follows:

- Moisture sensors will be used to monitor the moisture content of the soil
- Lights will be used to control the growth rate
- Pumps will be used to control food and water into the soil
- System components:
    - 2 analog moisture sensors
    - 4 channel 5V relay module
    - 2 Peristaltic Dosing Pumps (controls water inlet and feed)

The software application used the following classes.

- Pi Pico
- ADC for reading the analog moisture sensors
- Pins for relay control and pumps

The student's flowchart for this project is shown in Figure 18.

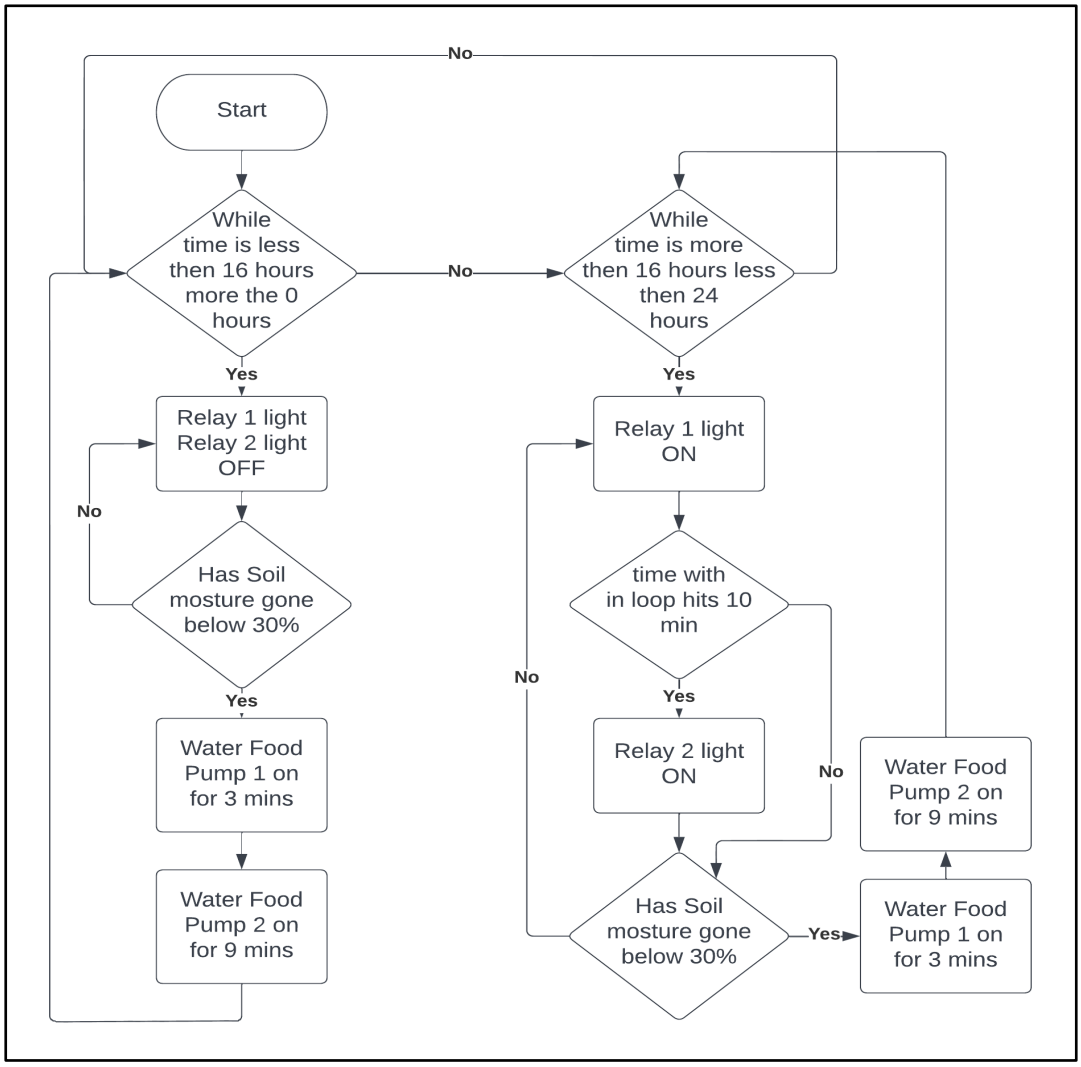

Figure 18. Student Flowchart for Plant Monitoring and Control System

The YouTube video showing the student's project is found here [19].

Remote Vehicle Speed Monitoring System

The overall objective is to remotely monitor and display vehicle speed. The engineering requirements are:

- Simulate the pulses generated by road sensors
- Calculate vehicle speed based on the time between the pulses
- Allow the vehicle speed to be monitored using a web browser

The web server portion of the project was based on the online script for the ESP8266 [18].

The hardware components for this project consisted of:

- Pi Pico
- WiFi Module with Breakout Board
- National Instruments USB DAQ

The software application used the following classes.

- ADC for reading the pulses generated by the USB DAQ
- UART for serially receiving and transmitting
- UOS for returning information about the system

The student's flowchart for this project is shown in Figure 19.

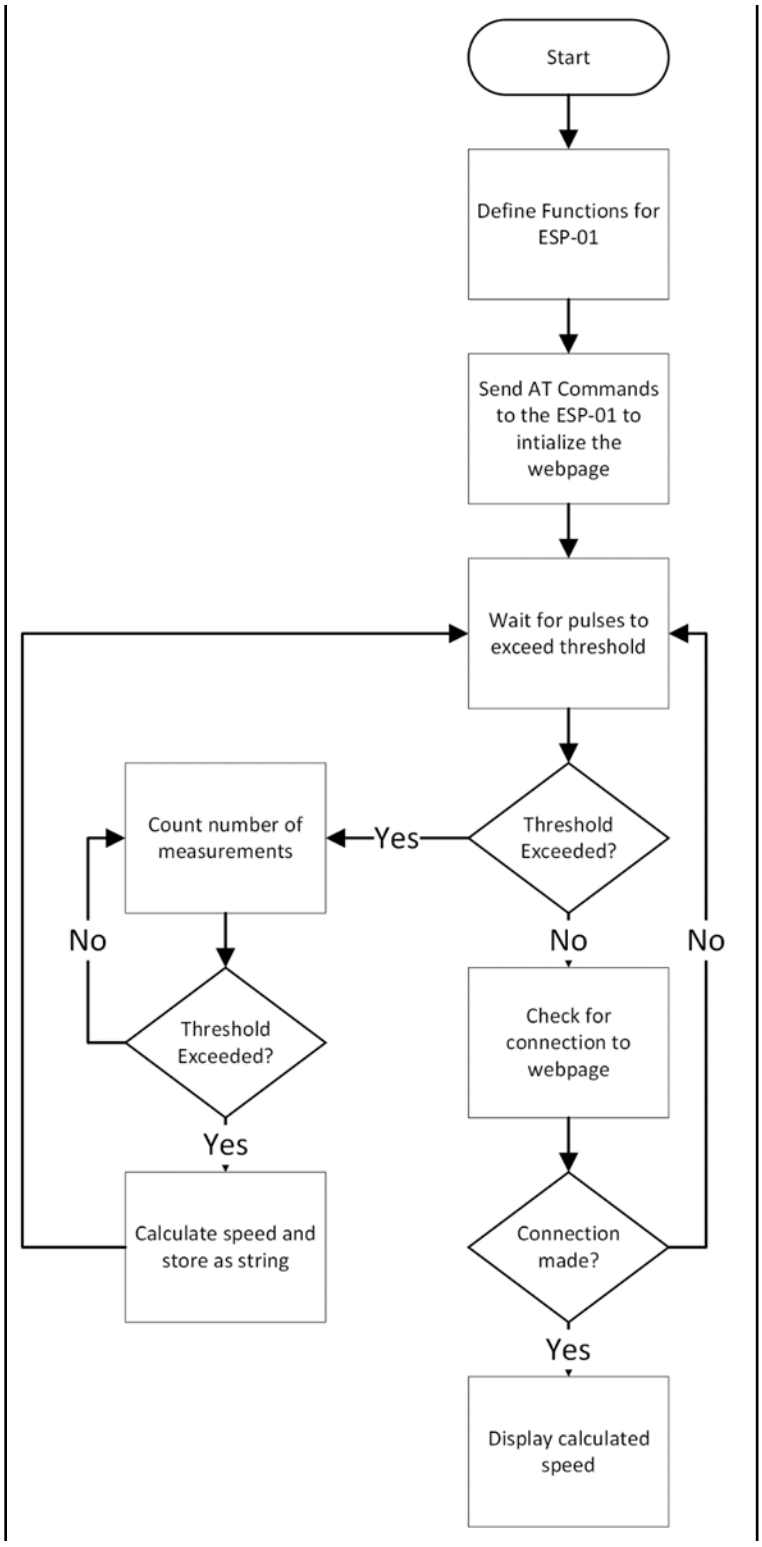

Figure 19. Student's Flowchart for Vehicle Speed Monitoring System

The YouTube video showing the student's project is found here [20].

Assessing Student Outcomes

From the 2022-2023 Criteria for Accrediting Engineering Technology Programs, student outcome 3 is listed below [21].

- (Associate Degree): An ability to apply written, oral, and graphical communication in well-defined technical and non-technical environments; and an ability to identify and use appropriate technical literature.
- (Baccalaureate Degree): An ability to apply written, oral, and graphical communication in broadly-defined technical and non-technical environments; and an ability to identify and use appropriate technical literature.

where:

- Well-defined activities or problems are practical, narrow in scope, use conventional processes and materials in traditional ways, and require knowledge of standard operating processes
- Broadly-defined activities or problems are practical, broad in scope, relatively complex, and involve a variety of resources; use new processes, materials, or techniques in innovative ways; and may require extension of standard operating procedures**.**

Analytic rubrics provide repeatable performance scales to inform the assessment process. These scales de-couple the assessment process from student grades, providing the opportunity for insight to be gained into overall trends in student performance, independent of the grading schemes used in the various classes.

The same rubrics can be used to assess the associate level students and the bachelor level students. To achieve this, the rubrics can be organized in a "staggered developmental" manner, with overlapping or cascading performance scales at the top. An example of a rubric for assessing student outcome 3 is shown below in Table 1. The rubric lists the Baccalaureate and Associate Student Outcome, performance indicators for the Outcome, and three performance levels for each performance indicator. These performance levels are:

- Developing
- Meets Expectations
- Exceeds Expectations

A fourth level (N/A) is used when it is not possible to use the rubric with the performance indicator for this assessment method.

A similar rubric can be used for assessing student outcome 3 based on the 2022-2023 Criteria for Accrediting Engineering Programs [22].

Based on the above student projects, both the written reports and the YouTube videos can be used as instruments for assessing student outcome 3.

Table 1. Rubric for Assessing Student Outcome 3

	Performance level			
	Developing (AS)	Meets Expectations (AS)	Exceeds Expectations (AS)	
Performance Indicator		Developing (BS)	Meets Expectations (BS)	Exceeds Expectations (BS)
a. Ability to apply written, oral, and graphical communication in technical environments	Technical communications have little content that distinguishes them from that for a general audience.	Technical communications include properly labelled graphs and figures	Technical communications properly employ graphs, figures & equations suitable for audience.	Technical communications exceptionally clear & concise: advance knowledge beyond classroom content
b. Ability to apply written, oral, and graphical communication in non-technical environments	Non-technical communications are poorly organized and presented, difficult to comprehend.	Non-technical communications are well-organized, grammatical, and avoid jargon. Graphs and figures easily understood by non-technical reader.	Non-technical communications are well-organized, grammatical, and avoid jargon. Graphs and figures easily understood by non-technical reader. Both communicate content from BS level.	Non-technical communications exceptionally clear, explain engineering topics from across the curriculum to non-technical or non-college audience.
c. Ability to identify and use appropriate technical literature	Technical communications limited to popular or introductory sources.	Technical communications uses datasheets, textbook content, basic familiarity with industry codes, specifications & standards	Technical communications identifies and uses datasheets, thorough application of industry codes, specifications, and standards.	Technical communications identifies & incorporates journal article results, patent research, or novel techniques.

Summary

The Raspberry Pi Pico is an inexpensive board suitable for a variety of courses in engineering and engineering technology programs. For embedded courses, applications for the Pi Pico can include

IoT. MicroPython is optimized to run on microcontrollers, such as the Pi Pico. An IDE for developing Python code is called Thonny, and it is available for free.

The projects shown in this paper were designed to introduce students to the I/O capabilities of the Pi Pico. Devices interfaced to the Pi Pico included an analog temperature sensor, a serial LCD display, a digital-to-analog converter, an accelerometer, and a WiFi device. Two student projects were shown to illustrate examples of IoT applications. Methods of using results from these student projects to assess student outcomes was also shown.

Future Work

The Raspberry Pi Pico is now available with WiFi capability [23]. The authors will be evaluating the usage of this device for IoT applications.

References

[1] S. Strom and D. Loker, "Programmable System-On-Chip (PSoC) Usage in an Engineering Technology Program," *Annual Meeting, American Society for Engineering Education,* 2016.
[2] D. Loker and S. Strom, "Programmable System-On-Chip (PSoC) Usage in Embedded Programming Courses," *Annual Meeting, American Society for Engineering Education,* 2020.
[3] S. Strom and D. Loker, "BeagleBone Black for Embedded Measurement and Control Applications," *Annual Meeting, American Society for Engineering Education,* 2018.
[4] D. Loker, "MicroPython in a Wireless Communications Systems Course," *Annual Meeting, American Society for Engineering Education,* 2021.
[5] D. Loker, "Embedded Systems using the Raspberry Pi Pico," *Annual Meeting, American Society for Engineering Education,* 2022.
[6] Thonny. [Online]. Available: https://thonny.org/
[7] MicroPython Documentation. [Online]. Available: https://docs.micropython.org/en/latest/
[8] MicroPython. [Online]. Available: http://www.mircropython.org
[9] Raspberrypi.com. [Online]. Available: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf
[10] G. Halfacree and B. Everard, *Get Started with MicroPython on Rapsberry Pi Pico*, Raspberry Pi Trading Ltd, 2021.
[11] Analog Temperature Sensor (Part number: LM35DZ). [Online]. Available: https://www.digikey.com/en/products/detail/texas-instruments/LM35DZ-NOPB/32489
[12] Serial LCD Display (Part number: NHD-0420D3Z-NSW-BBW-V3). [Online]. Available: https://www.digikey.com/en/products/detail/newhaven-display-intl/NHD-0420D3Z-NSW-BBW-V3/2626390?s=N4IgTCBcDaIHIAkAiBaADAFjGpBmAWinAMoDqKAQheQGq5FIgC6AvkA
[13] SPI Python Library. [Online]. Available: https://docs.micropython.org/en/latest/library/machine.SPI.html
[14] MCP4901 Datasheets. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/22248a.pdf
[15] I2C Python Library. [Online]. Available: https://docs.micropython.org/en/latest/library/machine.I2C.html
[16] ADXL345 Breakout Board. [Online]. Available: https://www.sparkfun.com/products/9836
[17] ESP8266 WiFi Module. [Online]. Available: https://www.sparkfun.com/products/17146
[18] MicroPython Script for ESP8266. [Online]. Available: https://microcontrollerslab.com/esp8266-wifi-module-raspberry-pi-pico-web-server/
[19] YouTube Video. [Online]. Available: https://www.youtube.com/watch?v=Wjs0kJOb5-w
[20] YouTube Video. [Online]. Available: https://www.youtube.com/watch?v=he4y5hctvNs
[21] Criteria for Accrediting Engineering Technology Programs, 2022 – 2023. [Online]. Available: https://www.abet.org/wp-content/uploads/2022/01/2022-23-ETAC-Criteria.pdf
[22] Criteria for Accrediting Engineering Programs, 2022 – 2023. [Online]. Available: https://www.abet.org/wp-content/uploads/2022/01/2022-23-EAC-Criteria.pdf
[23] Raspberry Pi Pico W. [Online]. Available: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html