ALTAIR

HOW-TO TUTORIAL

Altair Embed® 2025.2

Programming Raspberry Pi Peripherals with Embed

Intellectual Property Rights Notice:

Contents

Introduction

Altair Embed® is model-based design software for developing algorithms for complex embedded systems. With Embed, you design, analyze, and simulate using block diagrams and state charts. You can then automatically generate compact and optimized firmware to run on an extensive selection of micro-controllers.

There are three editions of Embed: Embed Pro, Embed SE, and Embed Personal. Only Embed Pro and Embed Personal can be used for embedded system development; however, for simplicity, this guide uses the term Embed when referring to Embed Pro and Embed Personal.

Embed and Raspberry Pi

The Raspberry Pi® is an inexpensive computer that runs Linux®. It also provides a set of general purpose input/output (GPIO) pins, allowing you to control electronic components for physical computing and explore the Internet of Things (IoT).

Embed has integrated Raspberry Pi as a microcontroller enabling you to design and develop applications on the device using its capabilities (such as, GPIO pins, IoT, and image/video processing).

To integrate Raspberry Pi in Embed requires a communication protocol that can be used to send and receive data and binaries. To do so, you will use a Daemon program. This Daemon is installed as part of Repository the first time you boot the Raspberry Pi. Once Repository is installed, Daemon runs on Raspberry every time Raspberry Pi is running. This Daemon polls on commands from Embed and communicates with Embed via the TCP/UDP protocol, as shown below:

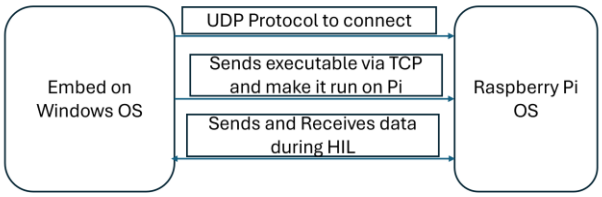


Figure 1: Embed and Raspberry Pi communication.

Examples

This section presents examples of programming Raspberry Pi peripherals with Embed.

Blinking the LED

The blink LED experiment uses a GPIO pin to blink an LED at a controlled frequency. In addition to the Raspberry Pi, the following hardware is required:

- Breadboard
- Jumper cables
- LED
- 600 to 1000 ohm resistor

Raspberry Pi target connection

In this experiment, you connect GPIO 0 to the positive pin on the LED and the GND to the negative pin on the LED.

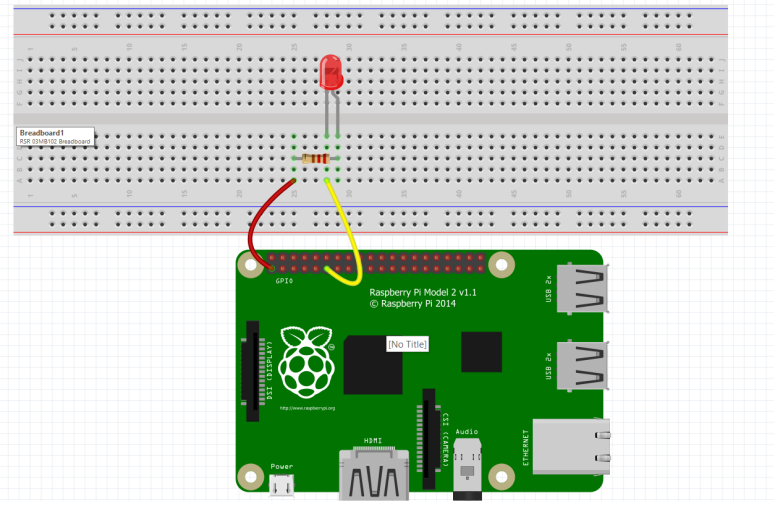


Figure 2: LED blink hardware schematic.

Embed diagram

Location: Examples > Embedded > Linux > Raspberry Pi > Blink > BlinkLEDOnRPi3BPlus.vsm

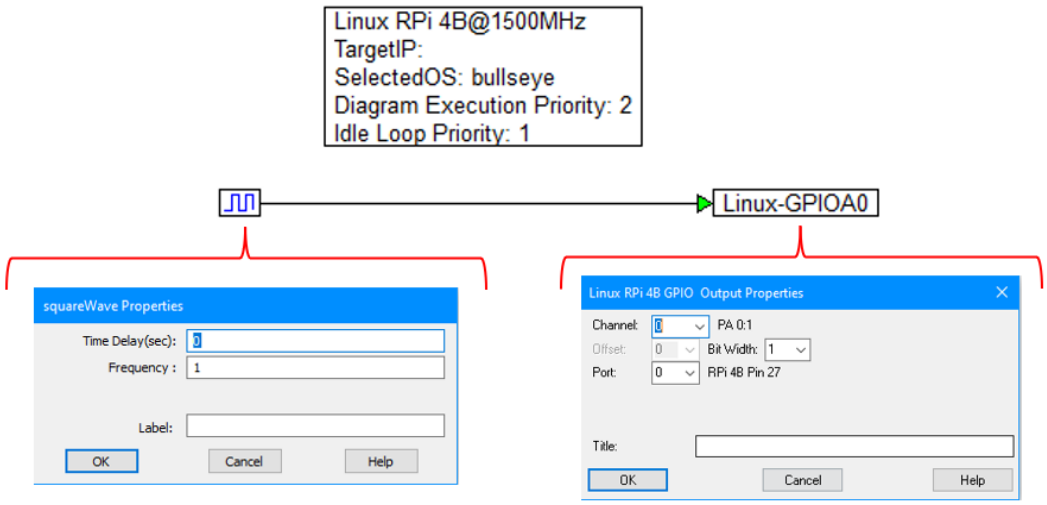


Figure 3: LED blink diagram.

Integration steps

1. Connect the **LED**, as shown in Figure 2.
2. In the diagram, note that a **squareWave** block is used as input to the **GPIO output** block.
3. Select the **GPIO pin** connected to LED in **GPIO output** block.
4. Click **Tools > Code Gen**.
5. In the Code Gen dialog, click **Code Gen** then the **Compile** buttons.
6. Once the Compile window displays success, click the **Download** button.
7. The LED is now blinking.

Blinking the LED using HIL feature

The blink LED experiment uses a GPIO pin to blink an LED at a controlled frequency. In addition to the Raspberry Pi, the following hardware is required:

- Breadboard
- Jumper cables
- LED
- 600 to 1000 ohm resistor

Raspberry Pi target connection

In this experiment, you connect GPIO 0 to the positive pin on the LED and the GND pin to the negative pin on LED.

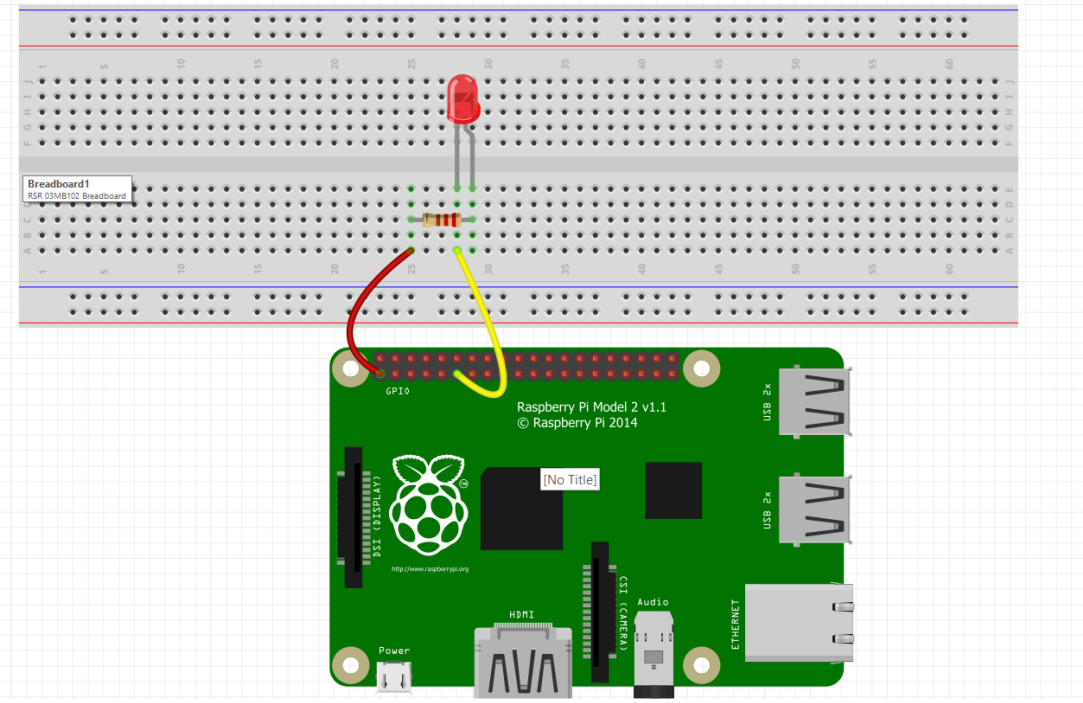


Figure 4: LED blink hardware schematic.

Embed diagram

InterruptUsingButtonFallingEdge.vsm

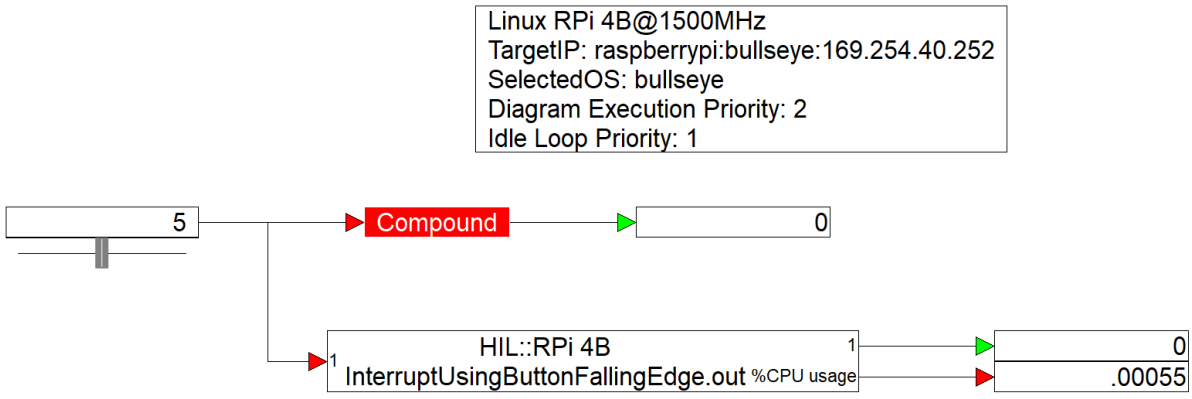


Figure 5: LED blink diagram.

Integration steps

1. Connect the **LED**, as shown in Figure 4.
2. The contents of **Compound** block are:

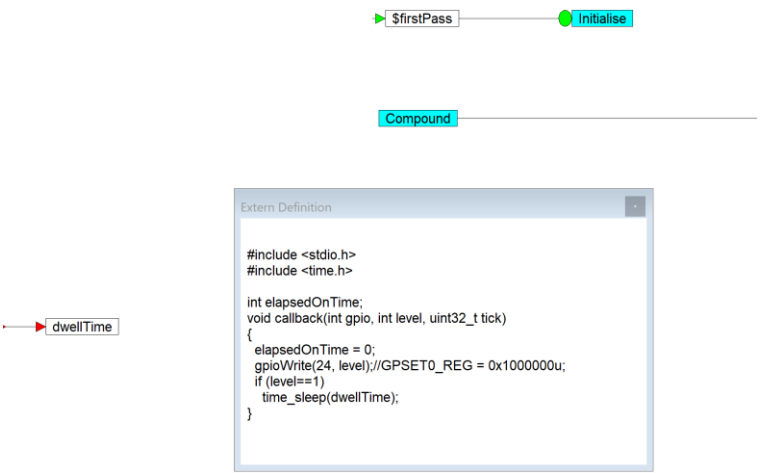


Figure 6: Compound block.

3. Inside the compound block, the Alert function is defined using the **Extern Definition** block. The **dwellTime** variable is also defined and it tells the GPIO to be ON for the specific time and then switch OFF.
4. Using the **slider** block shown in Figure 5, you can define how much time (in seconds) you want the LED to be ON. You can also see the LED blinking on rising edge of the defined interrupt.

Dimming LED

The dim LED experiment uses a GPIO pin to dim an LED by changing the duty cycle of the PWM signal. In addition to the Raspberry Pi, the following hardware is required:

- Breadboard
- Jumper cables
- LED
- 600 to 1000 ohm resistor

Raspberry Pi target connection

In this experiment, you connect GPIO 12 to a positive pin on the LED, and the GND to the negative pin on the LED.

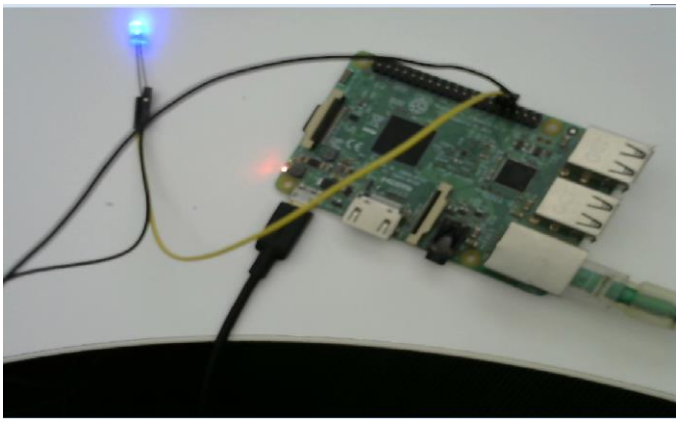


Figure 7: Dimming the LED using a PWM signal.

Embed diagram

Location: Examples > Embedded > Linux > Raspberry Pi > Application > PWM_Slider_RPi3BPlus.vsm

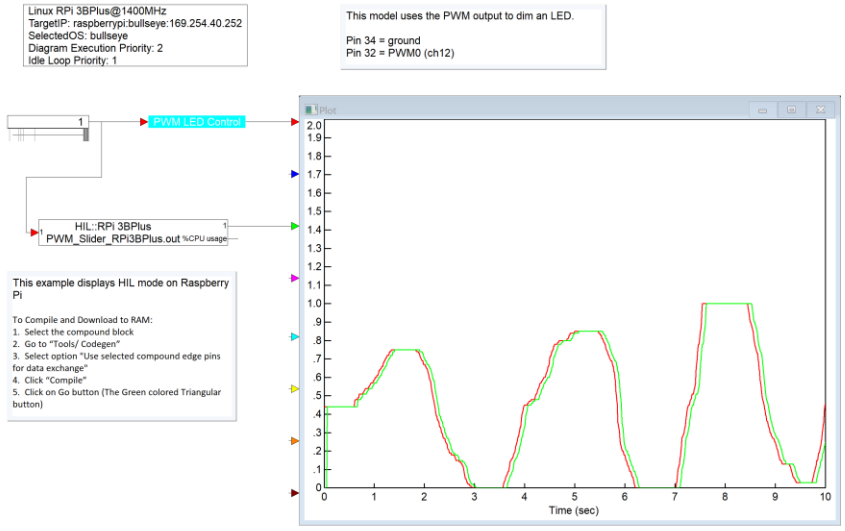


Figure 8: Embed diagram for LED dimmer using PWM.

Integration steps

1. Connect the **LED**, as shown in Figure 7.
2. Select the **Compound** block.
3. Click **Tools > Code Gen**.
4. In the Code Gen dialog, activate the **Use selected compound edge pins …**
5. Click the **Code Gen** then the **Compile** buttons.
6. Once the Compile windows display success, close all the dialogs and windows.
7. Click the **Go** toolbar button.
8. You can now control the brightness of LED using the **slider** block in the diagram.

Light intensity using the sensor SEN0390

In this experiment, you connect a SEN0390 light sensor to the I2C pins on the Raspberry Pi.

Figure 9: SEN0390 light sensor.

The screen below was captured when the light sensor SEN0390 board was connected to the Raspberry Pi on port 1 and the i2cdetect command was entered:

```
pi@raspberrypi:~/SEN0390 $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- 4a -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~/SEN0390 $
```

Figure 10: Slave address of SEN0390.

Raspberry Pi target connection

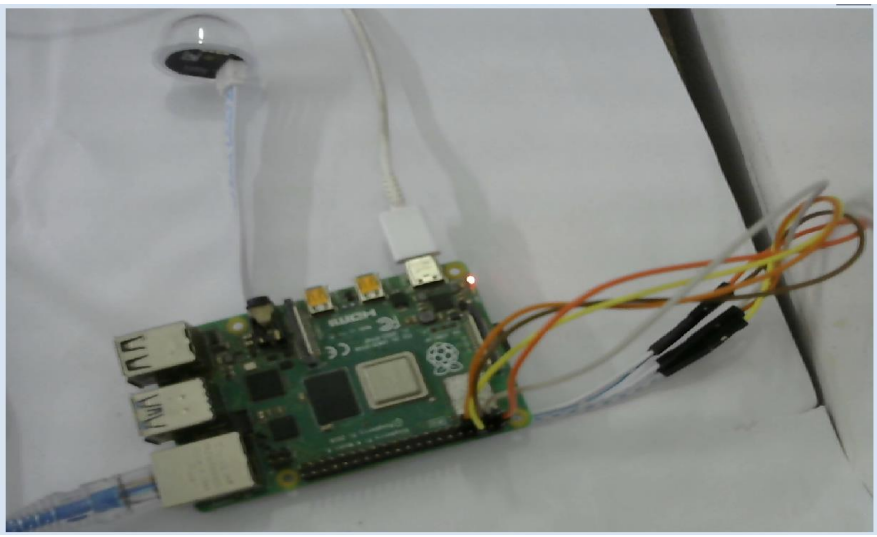


Figure 11: Blink LEDs on PICKit SPI board.

Embed diagram

SEN0390_Pi.vsm

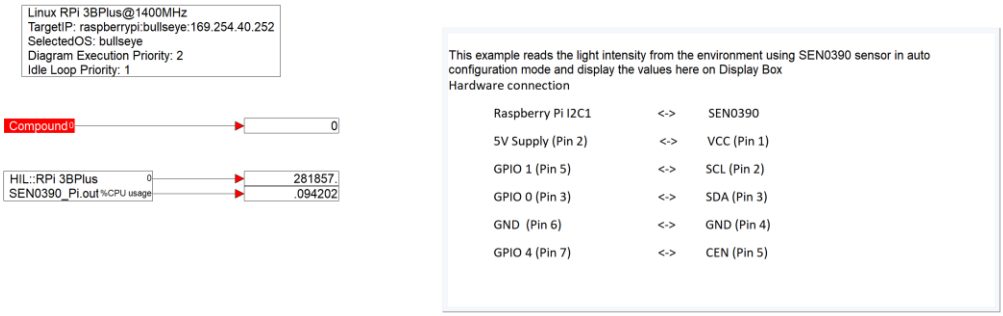


Figure 12: Embed diagram.

Integration steps

1. To understand the diagram, you need to understand how the SEN0390 sensor works, which is described in the SEN0390 datasheet.
2. As per the datasheet, the sensor works in two modes:
    - **Automatic Configuration**: You can send the data register address and start reading the light intensity value from the sensor.
    - **Manual Configuration:** You need to send right commands to the Config register and then send the data register address and start reading the light intensity value.
3. There are four registers sending data from the sensor. You need to get data from all four registers and do some computation to get the actual light intensity value in Lux.

This is captured inside the compound block of the diagram.

8-bit expander on PICKit serial SPI demo board

In this experiment, you connect the PICKit SPI serial pins to the SPI pins on the Raspberry Pi.

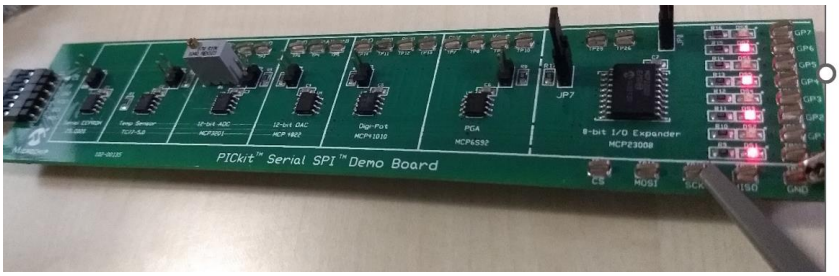


Figure 13: PICKit serial SPI demo board.

Raspberry Pi target connection

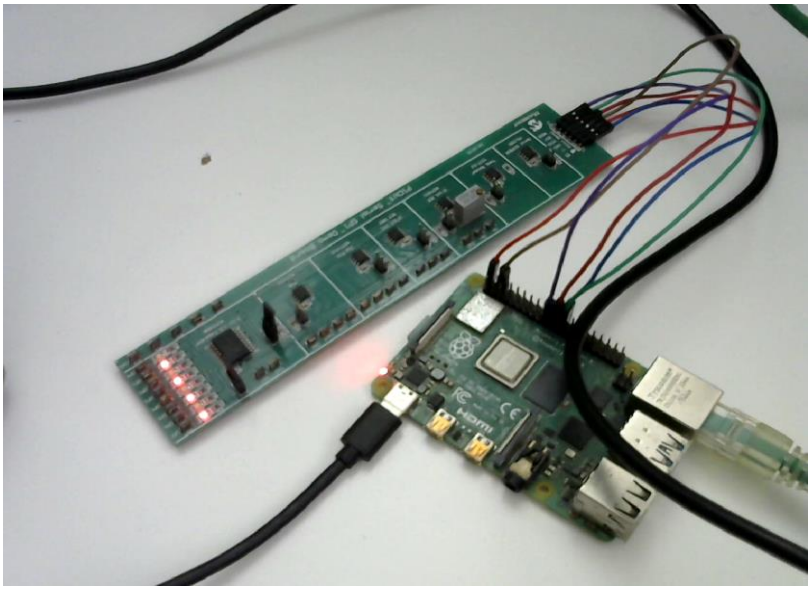


Figure 14: Blink LEDs on PICKit SPI board.

Embed diagram

Location: Examples > Embedded > Linux > Raspberry Pi > SPI > SPI0_MCP23S08_RPi3BPlus.vsm

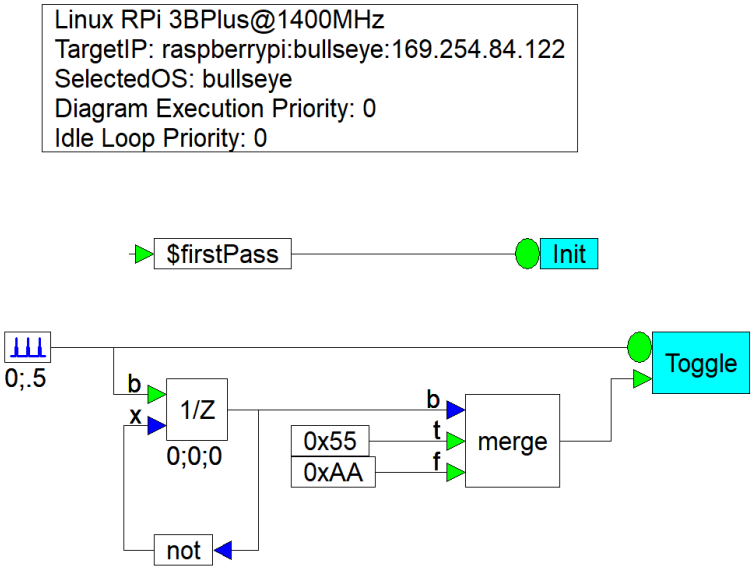


Figure 15: Embed diagram.

Integration steps

1. To understand the diagram, you need to understand how the MCP23008 sensor works, which is described in the MCP23008 datasheet.
2. As per the datasheet, follow these steps:
    a. The MCP23X08 contains 11 registers that can be addressed through the serial interface block.

Address	Access to:
00h	IODIR
01h	IPOL
02h	GPINTEN
03h	DEFVAL
04h	INTCON
05h	IOCON
06h	GPPU
07h	INTF
08h	INTCAP (Read-only)
09h	GPIO
0Ah	OLAT

Figure 16: MCP23008 registers.

    b. The Sequential Operation (SEQOP) bit (IOCON register) controls the operation of the address pointer. The address pointer can either be enabled (default) to allow the address pointer to increment automatically after each data transfer, or it can be disabled. When operating in Sequential mode (IOCON.SEQOP = 0), the address pointer automatically increments to the next address after each byte is clocked. When operating in Byte mode (IOCON.SEQOP = 1), the MCP23X08 does not increment its address counter after each byte during the data transfer. This gives the ability to continually read the same address by providing extra clocks (without additional control bytes). This is useful for polling the GPIO register for data changes.

    c. The MCP23S08 is a slave SPI device. The slave address contains five fixed bits and two user-defined hardware address bits (pins A1 and A0), with the read/write bit filling out the control byte.

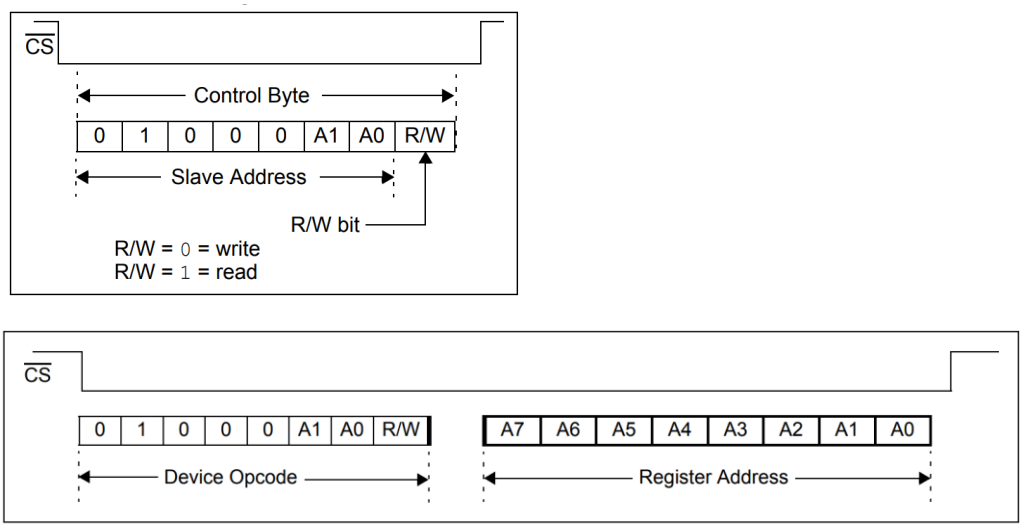


Figure 17: Working of SPI23008.

d. Steps 2a to 2c in the description is modeled as **Init** compound block (connected to **firstPass** variable) for the initialization (to set I/O register) in the example:

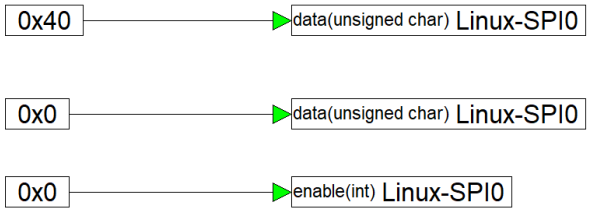


Figure 18: SPI write example in Embed.

e. The GPIO module contains the data port (GPIO), internal pull up resistors and the Output Latches (OLAT). Reading the GPIO register reads the value on the port. Reading the OLAT register only reads the OLAT, not the actual value on the port. Writing to the GPIO register causes a write to the OLAT. Writing to the OLAT register forces the associated output drivers to drive to the level in OLAT. Pins configured as inputs turn off the associated output driver and put it in high impedance.

f. Step e describes the normal execution of the I/O expander and is encapsulated (setting of OLAT register) in compound block below:

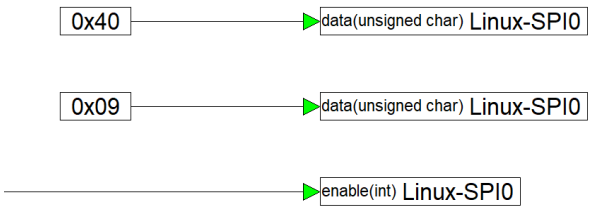


Figure 19: SPI write example.

g. The logic below is applied to toggle the LEDs on the I/O expander:

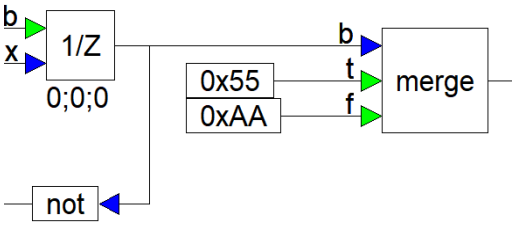


Figure 20: Logic to toggle MCP23008 LEDs.

h. At one time, 0x55 – 0x01010101; that is, odd set of LEDs blink.

i. At other time, 0xAA – 0x10101010; that is, even set of LEDs blink.

Temperature, pressure, and humidity sensor on BME280

This experiment uses state charts along with the I2C peripheral to read data from a BME280 sensor.

Raspberry Pi target connection

In this experiment, you connect the BME280 pins to I2C1 pins on Raspberry Pi:

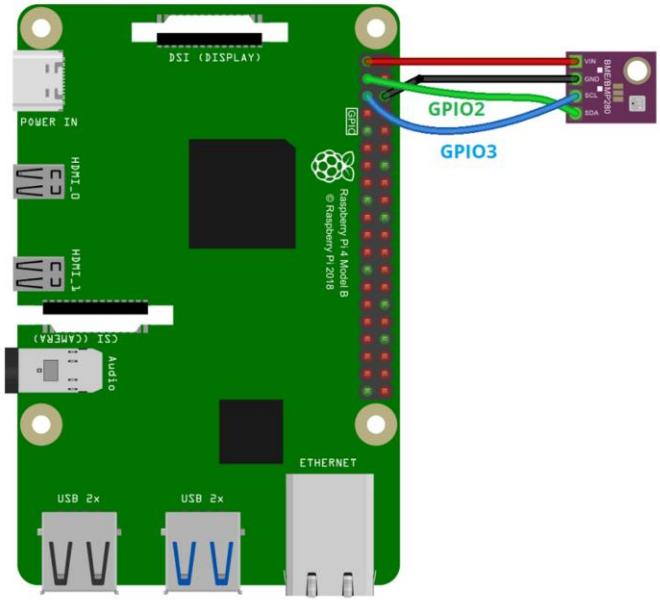


Figure 21: BME280 connection with Raspberry Pi.

Embed diagram

Location: Examples > Embedded > Linux > Raspberry Pi > I2C > BMEP280-RPi.vsm

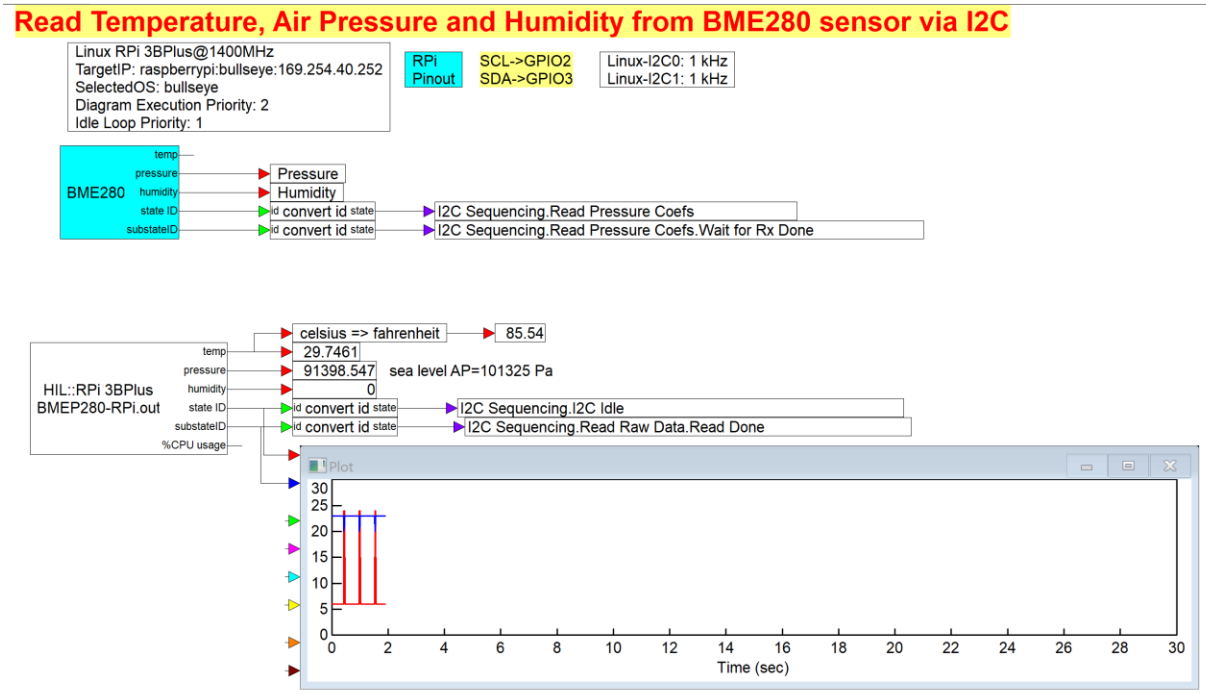
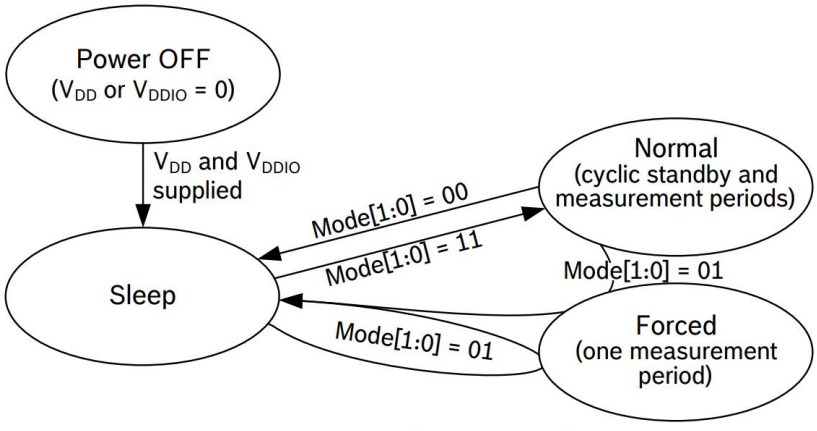


Figure 22: Embed diagram.

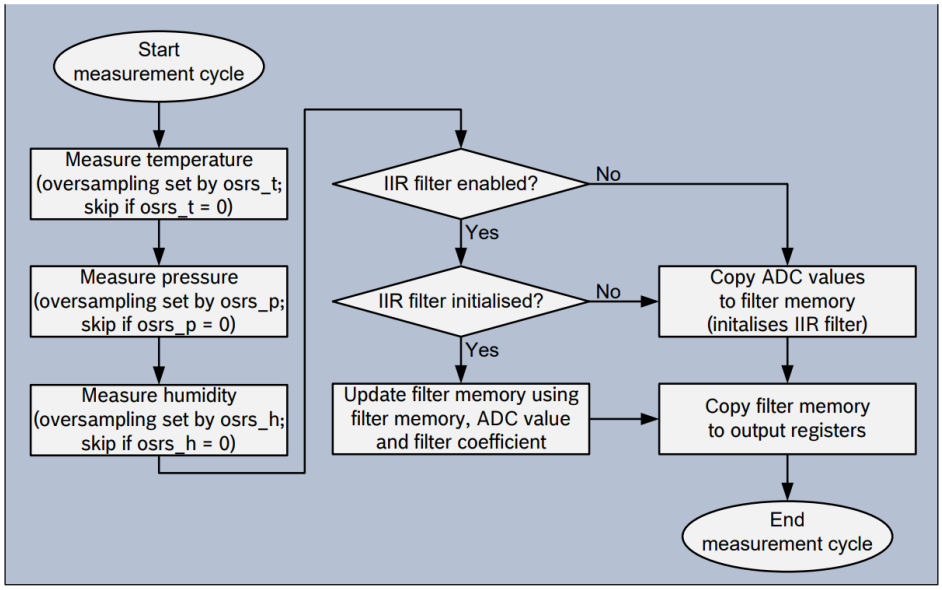
Integration steps

1. To understand the diagram, you need to understand how the BME280 sensor works, which is described in the BME280 datasheet.
2. Per the datasheet, follow these steps:
    a. The BME280 offers three sensor modes: sleep mode, forced mode and normal mode. These can be selected using the mode[1:0] setting:
        i. **Sleep mode:** No operation, all registers accessible, lowest power, selected after start up.
        ii. **Forced mode:** Perform one measurement, store results and return to sleep mode.
        iii. **Normal mode:** Perpetual cycling of measurements and inactive periods.

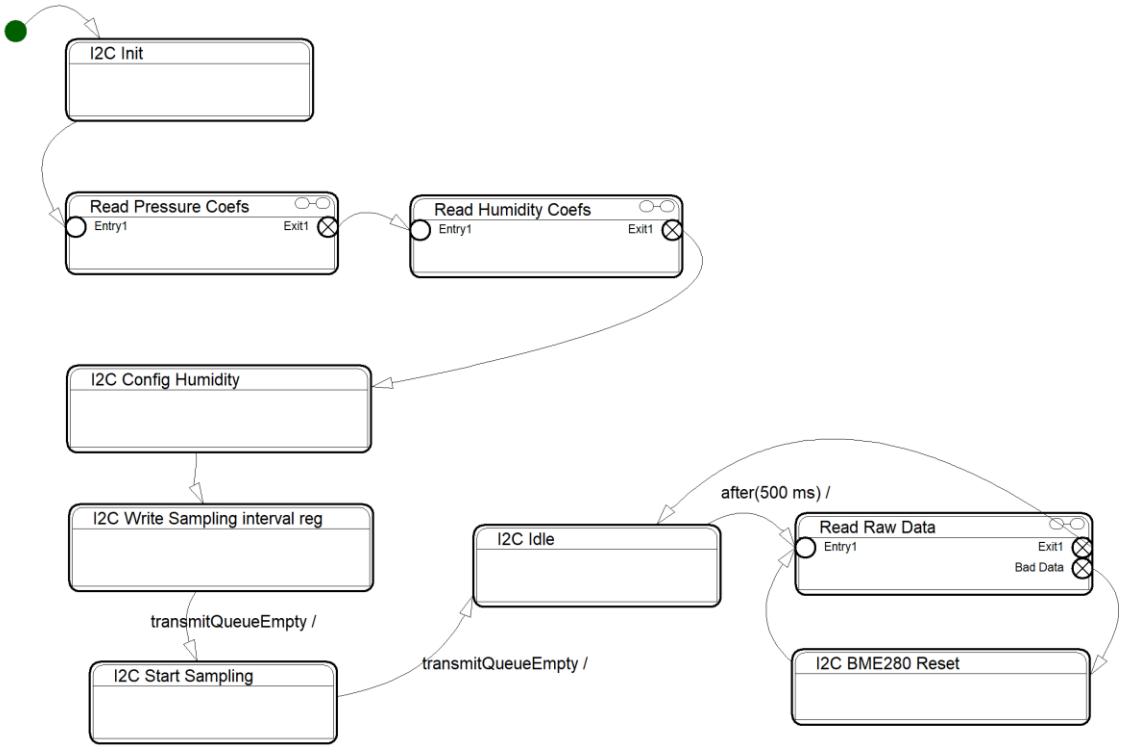


    b. Normal mode comprises an automated perpetual cycling between an active measurement period and an inactive stand-by period.

c. The BME280 measurement period consists of a temperature, pressure, and humidity measurement with selectable oversampling. After the measurement period, the pressure and temperature data can be passed through an optional IIR filter, which removes short-term fluctuations in pressure (for example, caused by slamming a door). For humidity, such a filter is not needed and has not been implemented. The measurement flow is depicted in the diagram below:



3. This flow is achieved using the state chart feature in Embed.
4. The state chart flow is shown below:



SEN0189 turbidity sensor with MCP3008 ADC

This example uses Embed's Extern blocks.

Raspberry Pi target connection

In this experiment, you connect the SPI0 pins to the MCP3008, and then connect the MCP3008 to the SEN0189 sensor.

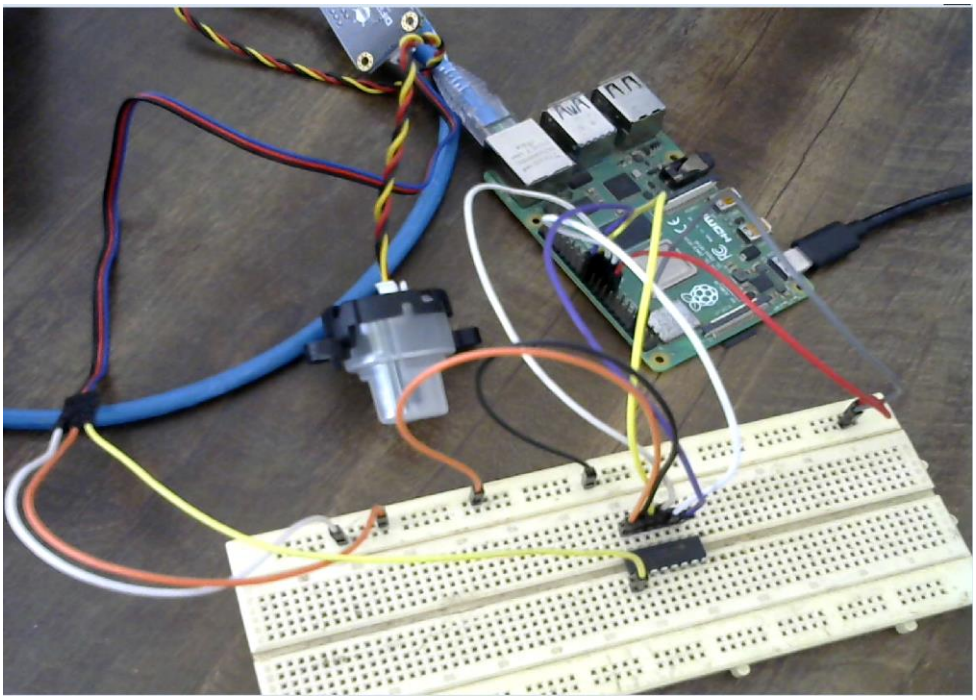


Figure 23: MCP3008 connected to Raspberry Pi and SEN0189.

Embed diagram

MCP3008_SEN0189.vsm

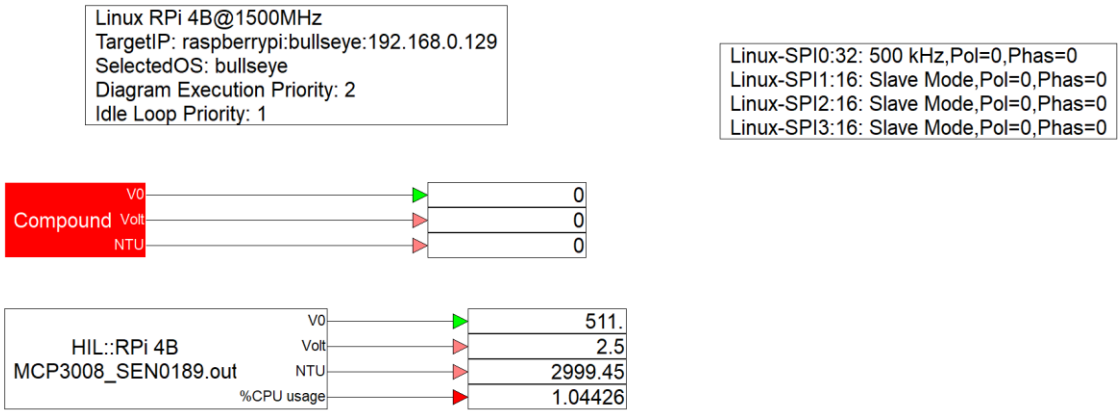


Figure 24: Embed diagram.

Integration steps

1. To create this diagram in Embed, you need to understand how the SEN0189 sensor works, which is described in the SEN0189 datasheet.
2. In this example, **Extern function** blocks are used. You can use **SPI Read** and **SPI Write** blocks to configure

the MCP3008 and take the readings from the ADC sensor.

Capturing video using a camera

This example uses Embed's Extern blocks.

Raspberry Pi target connection

To connect a camera to your Raspberry Pi, use a camera ribbon cable, as shown below:

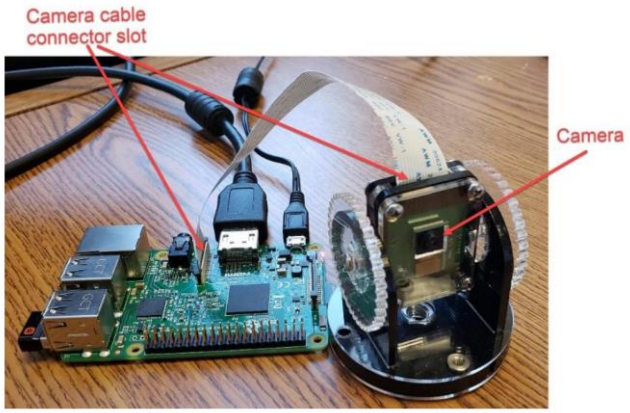


Figure 25: Ribbon cable camera connection.

Embed diagram

Location: Examples > Embedded > Linux > Raspberry Pi > OpenVision > Signal Producer > ReadCamera.vsm

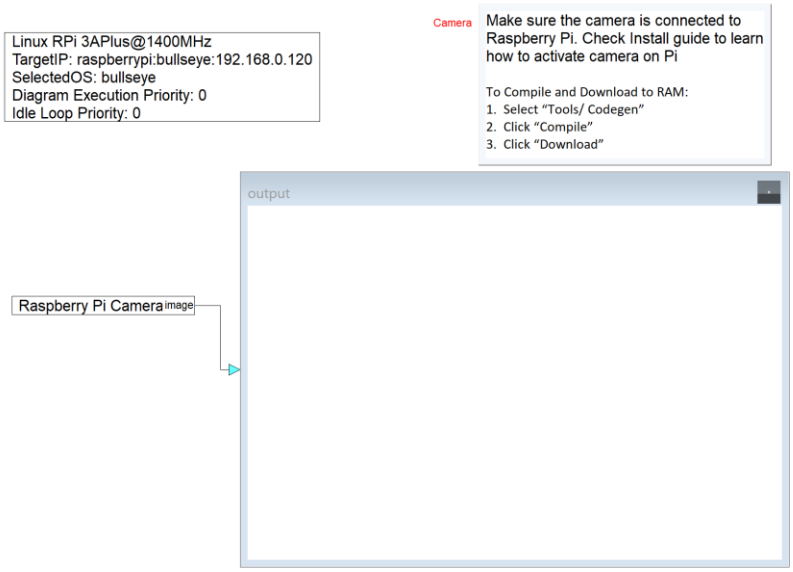
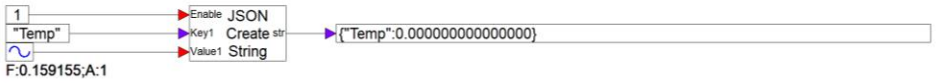


Figure 26: Embed diagram for reading Image on Raspberry Pi.

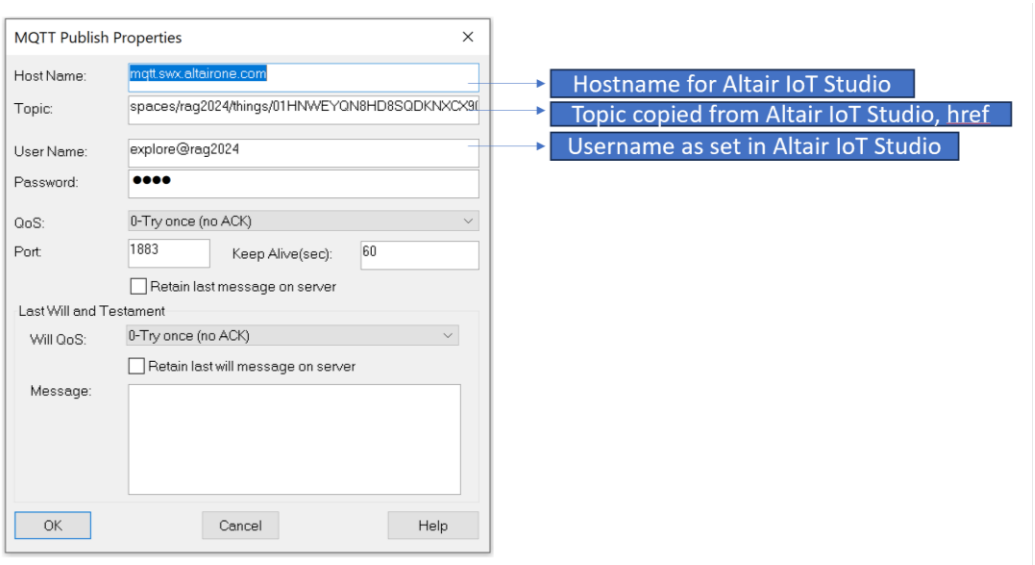
Sending and receiving data on Raspberry Pi to cloud interface using MQTT protocol

1. Open **Altair IoT Studio** and create a new **Things** (if not done earlier).
2. Click the **UID of Things** and go to the **Details** tab. You will find details like Topic/Username/password only here.

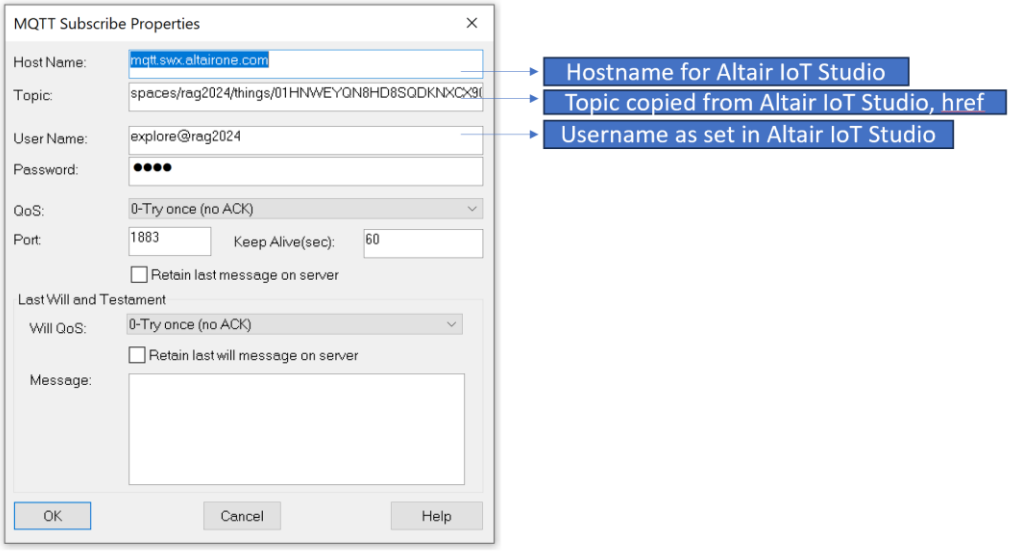
3. Start **Embed**.
4. Open **IoTStudioEmbed2024Pi.vsm**.
5. Create a **JSON String** as follows:



6. Update the **MQTT Publish** block as shown below:

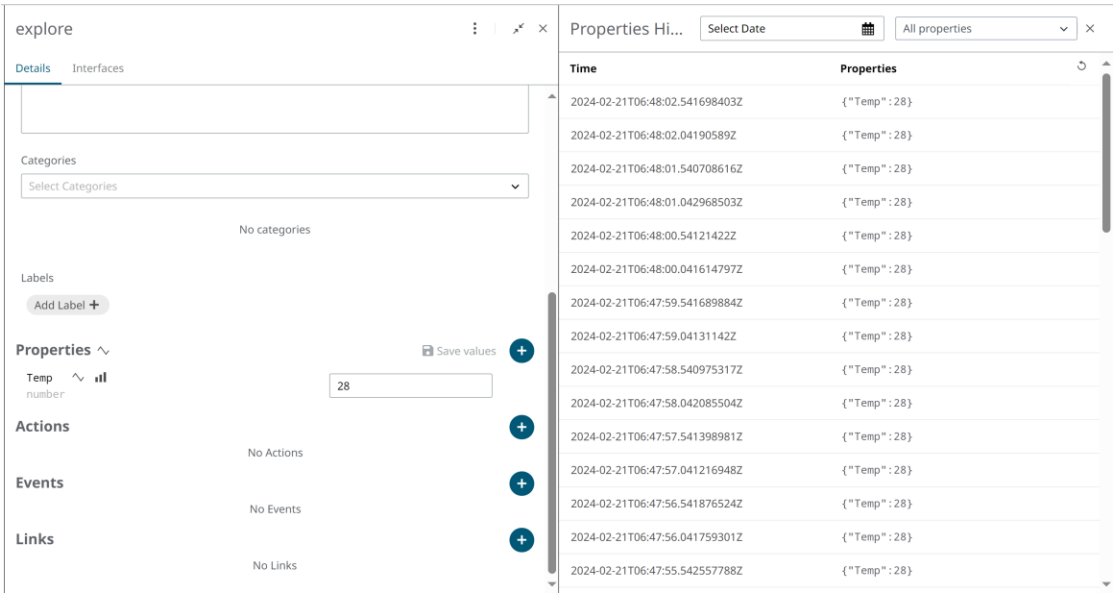


7. Configure the **MQTT Subscribe** block with similar lines:

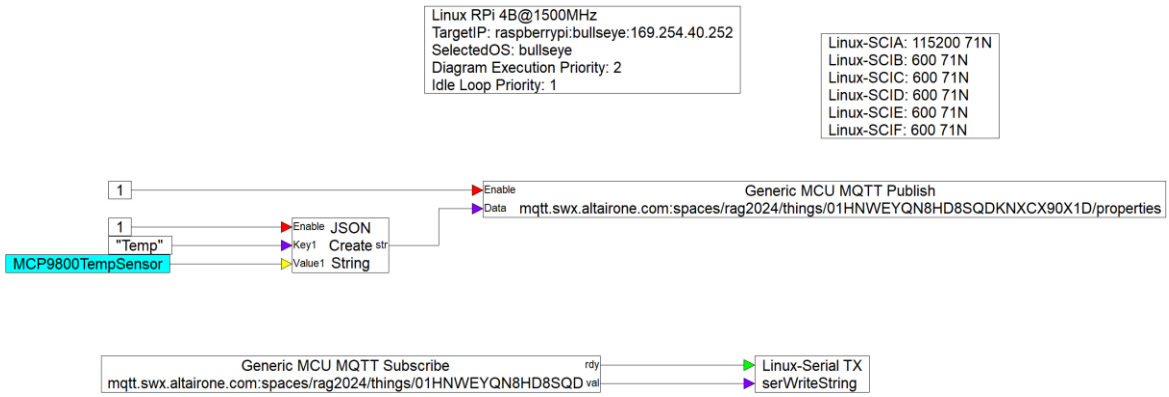


8. Connect the blocks as shown below:

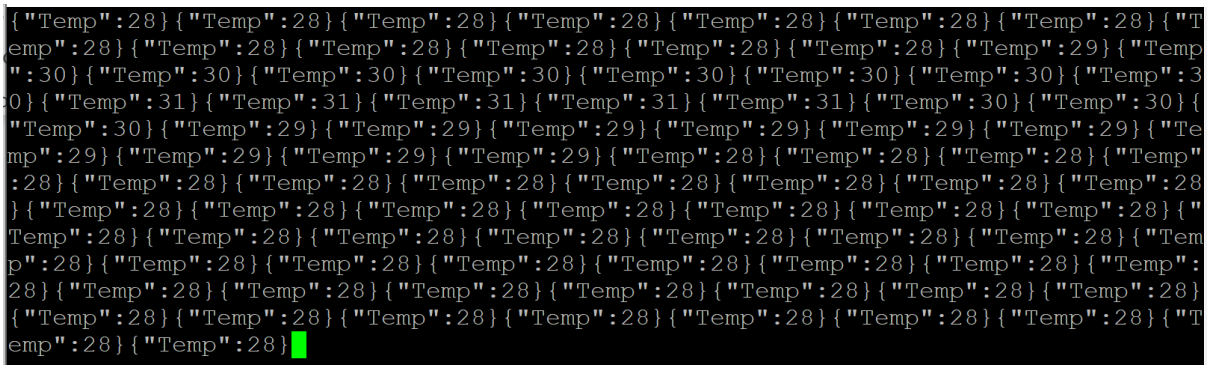
9. Run the Embed diagram and verify the results.



10. You can verify the subscribed data on Raspberry Pi as well using a serial communication.



11. The output on a serial terminal is shown below:



Auto-executing a program on Raspberry Pi power up

It's often useful to be able to restart your Raspberry Pi remotely with a specific OUT file executing. For example, consider a monitoring and control algorithm executing on the Raspberry Pi and communicating with a remote host using MQTT running Embed. If a loss of communication or crash occurs on the Raspberry Pi, you need a way to remotely reboot the Raspberry Pi with the monitoring and control algorithm automatically launched and running. There are several ways to do this: the recommended method is to create a service. When the Raspberry Pi is rebooted or powered on, this service is called by the Systemd initialization system used by the Raspberry Pi OS.

For this example, the service is named myEdge.service and the executable file to be automatically start is named myEdgeAlgorithm.out. The host program is named myHostAlgorithm.vsm.

1. Use **PuTTY** to launch a **Raspberry Pi command window**.
2. Use the **nano editor** to create a **new service file in systemd**:

   sudo nano /etc/systemd/system/LBNL.service

   This command creates the myEdge.service file and allows you to edit its contents.
3. While in the nano editor, add the **service configuration code**:

   ```
   [Unit]
   Description=myEdge.service
   [Service]
   Type=simple
   # ExecStartPre is the action to run before starting our service.  We are using a 15 second delay
   # to ensure there is enough time for the raspberry pi networking to complete its initialization
   ExecStartPre=/bin/sleep 15
   #myEdgeAlgorithm.out is located in the "Downloads" folder, yours may be located somewhere else
   ExecStart=/home/pi/Downloads/ myEdgeAlgorithm.out
   Restart=always
   [Install]
   WantedBy=multi-user.target
   ```

   Press Ctrl X Y Enter keys to exit the nano editor and save the file.
4. Enter the following command to **reload system** to **recognize the new service**:

   Sudo systemctl daemon-reload
5. **Enable service** to run at boot:

   Sudo systemctl enable myEdge.service
6. **Test the service** without rebooting to see if it starts:

   Sudo systemctl start myEdge.service
7. **Check** if the service is running:

   Sudo systemctl status myEdge.service
8. **Test** the service after a reboot:

   Sudo reboot
9. **Check** if the service is running:

   Sudo systemctl status myEdge.service
10. At this time, you can execute the host program myHostAlgorithm.vsm.