Raspberry Pico and I2C

Introduction

I have been using the Raspberry Pi Zero for a number of projects, but recently they are hard to come by. I'm now looking at the Raspberry Pico for a weather station project.

I would like to build a portable weather station for use during the 2024 Total Eclipse, to document temperature and other changes during the few hours of the eclipse, and especially the few minutes of the total eclipse. I have worked on several weather balloon payloads so I have some experience with these types of measurements.

The Pico reminds me of the Arduino. It is less powerful and flexable than the Zero. It has memory limits for logging, but it includes some analog to digital converters.   For a project with limited logging needs, it can be suitable.

Parameters to be measured include Temperature, Pressure, Humidity/Dew Point, Light levels, Wind Speed and direction. My non-expert expectations are that the reduces light energy during the total eclipse will result in temperature and air pressure drops, which may result in winds toward the path of the total eclipse. My expectation is that the moisture content of the air (and dew point) should not change much. I have seen some references to 10 degree temperature drop during the total eclipse, but have not seen references to other changes.

The immediate goal is to use the Raspberry Pico to read a I2C temperature and pressure sensor. A wiser person would install Adafuit's CircuitPython[1] onto the Pico. Instead I will use MicroPython [2]. The sensor used for this initial work is the BMP280 [3]. A MicroPython driver is available on GitHub [4]

Thonny

The Thonny IDE [5] includes the capability of writing MicroPython code to the Raspberry Pico through the USB connection, as well as reading files back from the Pico. It uses a self-discovered com serial port. In addition to transferring and running code on the Pico, it can also set Pico's real time clock.

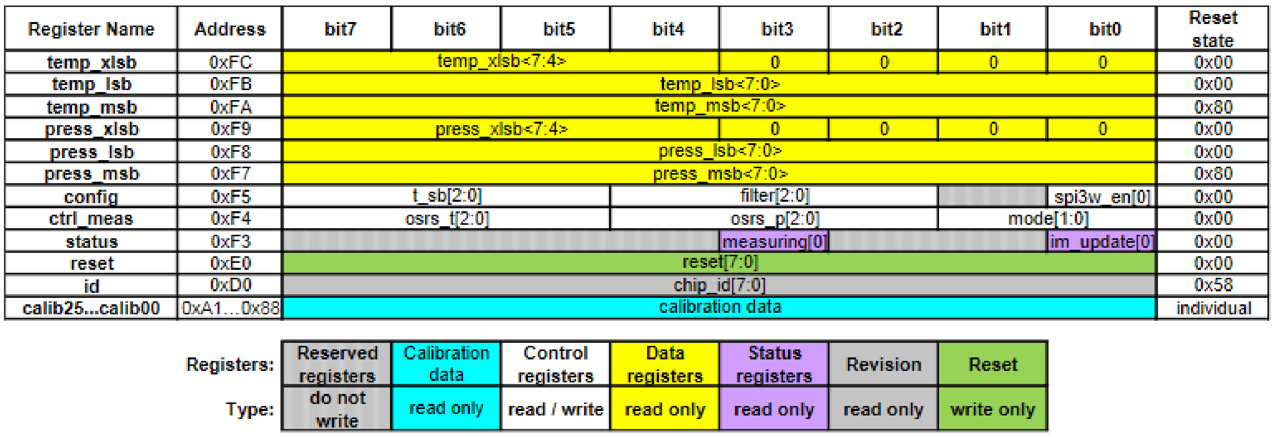
BMP280

The BMP280 is a combined Temperature and Pressure sensor. A link to the data sheet is provided as reference [6]. The Temperature sensor is used to correct the Pressure reading. The Pressure range is 300 to 1100 mbar, suitable to about 30k ft. The Temperature range is -40 to +85 C. The internal ADC is capable of up to 20 bits.   It includes I2C and SPI digital interfaces.

Communication with the unit are conducted by reading and writing the various eight-bit registers. The device can be configured by writing to two registers (CONFIG, and CTRL_MEAS). Three registers contain the temperature and pressure readings respectively (20 bit data). 24 read-only registers contain calibration data. Section 4 of the data sheet describes the registers and Table 18 from the data sheet illustrates the memory map.

Formula to convert ADC readings are described in section 3.11.3 of the datasheet [6], with sample data, results, and intermediate results in section 3.12. These a helpful in verifying an implementation of the formula.

Table 18: Memory map



Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]			▨	spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3	▨				measuring[0]	▨		im_update[0]	0x00
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x58
calib25...calib00	0xA1...0x88	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Revision	Reset
Type:	do not write	read only	read / write	read only	read only	read only	write only

Before using the BMP280.py library, it is helpful to determine the address of the BMP280. From Section 5.2 [6] the address can be 0x76 or 0x77. It can be set by the SDO pin of the SPI interface, but is probably pulled high or low from the circuit configuration. The address can be determined with the machine.I2C.scan() command. The BMP290 specifications say the i2c interface can operate at up to 3.4MHz. The following code is taken from reference [7], though I expect is pretty standard.

```
import machine
# Create I2C object
i2c = machine.I2C(0, scl=machine.Pin(17), sda=machine.Pin(16))
# Print out any addresses found
devices = i2c.scan()
if devices:
    for d in devices:
        print(hex(d))
```

Presumably it will show address of the device. In my case, the Adafruit BMP280 [3] has an address of 0x77.

BMP280.py

A library to use the BMP280 is available on GitHub [4]. It appears to enable the use of all the BMP280 capability listed on the Data Sheet [6], including the test parameters. The following code from reference [8] illustrates the initialization to use the library

```
from machine import Pin,I2C
from bmp280 import *

bus = I2C(0,scl=Pin(1),sda=Pin(0),freq=200000)
bmp = BMP280(bus)
```

However the default I2c address is 0x76, so I replace the last line with:

```
bmp = BMP280(bus,0x77)
```

The Data sheet describes six recommended configurations. These are implemented as follows:

```
# Use cases
BMP280_CASE_HANDHELD_LOW = const(0)
BMP280_CASE_HANDHELD_DYN = const(1)
BMP280_CASE_WEATHER = const(2)
BMP280_CASE_FLOOR = const(3)
BMP280_CASE_DROP = const(4)
BMP280_CASE_INDOOR = const(5)
```

The 16-bit temperature resolution (0.005 deg) is adequate for temperature compensation, and temperature documentation. Higher resolution pressure data may be useful for a dynamic situation like during a total eclipse.

The Handheld cases are not particularly low power modes but permit almost 100 readings/s (DYN) at standard (18-bit) or 10 readings/s (LOW) at high (20-bit) resolution.

My preferences would be standard pressure resolution (18-bit) at 1 reading/s, so there may be an opportunity use a lower power mode. Adjusted standby time of 125ms (mode 010) and filter coefficient of 5 samples (mode 4) will be implemented.

Try:

```
use_case(BMP280_CASE_HANDHELD_DYN)
standby(BMP280_STANDBY_125)
iir(BMP280_IIR_FILTER_4)
```

This should be suitable for reading the sensor at speeds of slower that once/second.

Time-Stamp

The Raspberry Pico includes a built-in real-time clock [9], but it is unpowered. The Thonny IDE sets the clock. In Thonny options, under the “Interpreter” a click box exists to “Synchonize device’s real time clock.” Documentation for the machine.rtc() class in included in the micropython documentation [10].

```
import machine
rtc = machine.RTC()
timestamp = rtc.datetime()
print (timestamp)
tm_str = "%04d-%02d-%02d %02d:%02d:%02d"%(timestamp[0:3] +
                                                timestamp[4:7])
print (tm_str)
```

Data Logging

Data Logging is simply writing data to a file. It can be done at regular intervals, or it can include timestamp information, or both. After acquiring the data, I prefer to open the data file, write the data, and close the file, so if a problem develops, for example loss of power, the data is still available and readable. Separating the data with commas makes it easy to import in a spreadsheet.

```
# Data Logging Example
import machine
import bmp280
import time


bus = machine.I2C(0,scl=machine.Pin(1),sda=machine.Pin(0),freq=200000)
bmp = bmp280.BMP280(bus,0x77)

fname = "temp_log.csv"
rtc = machine.RTC()

bmp.use_case(bmp280.BMP280_CASE_INDOOR)

file = open(fname, "a")
file.write(" ‘Timestamp’,’Temp(C)’,’Pres(bar)’ \n")
file.close()


while True:
    timestamp = rtc.datetime()
    tm_str = "%04d-%02d-%02d %02d:%02d:%02d"%(timestamp[0:3] +
                                            timestamp[4:7])
    print (tm_str)

    pressure=bmp.pressure
    p_bar=pressure/100000

    temperature=bmp.temperature
    print("Temperature: {} C".format(temperature))
    print("Pressure: {} Pa, {} bar".format(pressure,p_bar))
    print()

    file=open(fname,"a")
    file.write("%s, %6.2f, %7.5f \n" % (tm_str, temperature, p_bar))
    file.close()
    time.sleep(15)
```

The data files shown with Thonny by clicking on the File parameter under the View menu item. Clicking on the file name will import the file into Thonny, where it can be saved as a file on the computer.

Conclusion

It is fairly easy to use micropython on the Raspberry Pico to read I2C sensors and log the data.

References:

[1] Adafruit Circuit Python
https://learn.adafruit.com/getting-started-with-raspberry-pi-pico-circuitpython

[2] MicroPython
https://www.raspberrypi.com/documentation/microcontrollers/micropython.html

[3] BMP280
https://learn.adafruit.com/adafruit-bmp280-barometric-pressure-plus-temperature-sensor-breakout

[4] BMP280 micropython library
https://github.com/dafvid/micropython-bmp280

[5] Thonny IDE
https://thonny.org/

[6] BMP280 Datasheet
https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001.pdf

[7] Pico I2C scan code
https://www.digikey.com/en/maker/projects/raspberry-pi-pico-rp2040-i2c-example-with-micropython-and-cc/47d0c922b79342779cdbd4b37b7eb7e2

[8] Micropython bmp280 Example
https://electrocredible.com/bmp280-raspberry-pi-pico-micropython-guide/

[9] Real Time Clock
https://forums.raspberrypi.com/viewtopic.php?t=321271#p1923176

[10] machine.rtc()
https://docs.micropython.org/en/latest/library/machine.RTC.html

I2C examples
https://electrocredible.com/raspberry-pi-pico-i2c-tutorial-examples/
https://microcontrollerslab.com/raspberry-pi-pico-i2c-communication/

Raspberry Pico Documentation
https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-python-sdk.pdf
https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf

Micropython
https://micropython.org/
https://docs.micropython.org/en/latest/
https://docs.micropython.org/en/latest/reference/index.html

micropython bmp280 example
https://electrocredible.com/bmp280-raspberry-pi-pico-micropython-guide/

Micropython machine.I2C reference
https://docs.micropython.org/en/latest/library/machine.I2C.html