IoT Kit Extension Board MicroPython Functions Documentation -v0.2

This document provides a detailed explanation and usage guide for each sensor function available on the IoT extension board for microbit.

Table of Contents

Note: Before executing the code, it is recommened to turn off the microbit built-in led display to free up the ports.

```python
from microbit import *
display.off()
```

PIR Motion Sensor

`read_pir(pin)`

This function returns `True` if the PIR (Passive Infrared) sensor detects motion, otherwise `False`.

Arguments:

- `pin` - The pin connected to the PIR sensor.

Usage:

```
from microbit import *
from iotkit import read_pir

motion_detected = read_pir(pin0)
```

Light Intensity Sensor

`read_light_intensity(light_intensity_pin)`

This function reads the light intensity from the sensor and returns the value as a percentage.

Arguments:

- `light_intensity_pin` - The pin connected to the light intensity sensor.

Usage:

```
from microbit import *
from iotkit import read_light_intensity

light_intensity = read_light_intensity(pin1)
```

Noise Sensor

`read_noise(noise_pin)`

This function reads the noise level from the sensor and returns the value in dB (decibels).

Arguments:

- `noise_pin` - The pin connected to the noise sensor.

Usage:

```
from microbit import *
from iotkit import read_noise
```

```
noise_level = read_noise(pin2)
```

Dust Sensor

`read_dust(v_led, vo)`

This function reads the dust concentration from the sensor and returns the value in µg/m³ (micrograms per cubic meter).

Arguments:

- `v_led` - The pin connected to the dust sensor LED.
- `vo` - The pin connected to the dust sensor output.

Usage:

```
from microbit import *
from iotkit import read_dust

dust_concentration = read_dust(pin3, pin4)
```

SonarBit Distance Sensor

`sonarbit_distance(distance_unit, pin)`

This function reads the distance from the SonarBit sensor and returns the value in the selected unit.

Arguments:

- `distance_unit` - The unit of distance measurement (0 for mm, 1 for cm, 2 for inches).
- `pin` - The pin connected to the SonarBit sensor.

Usage:

```
from microbit import *
from iotkit import sonarbit_distance

distance_mm = sonarbit_distance(0, pin5)
```

```
distance_cm = sonarbit_distance(1, pin5)
distance_inch = sonarbit_distance(2, pin5)
```

Water Level Sensor

`read_water_level(water_level_pin)`

This function reads the water level from the sensor and returns the value as a percentage.

Arguments:

- `water_level_pin` - The pin connected to the water level sensor.

Usage:

```python
from microbit import *
from iotkit import read_water_level

water_level = read_water_level(pin6)
```

Soil Humidity Sensor

`read_soil_humidity(soil_moisture_pin)`

This function reads the soil humidity from the sensor and returns the value as a percentage.

Arguments:

- `soil_moisture_pin` - The pin connected to the soil humidity sensor.

Usage:

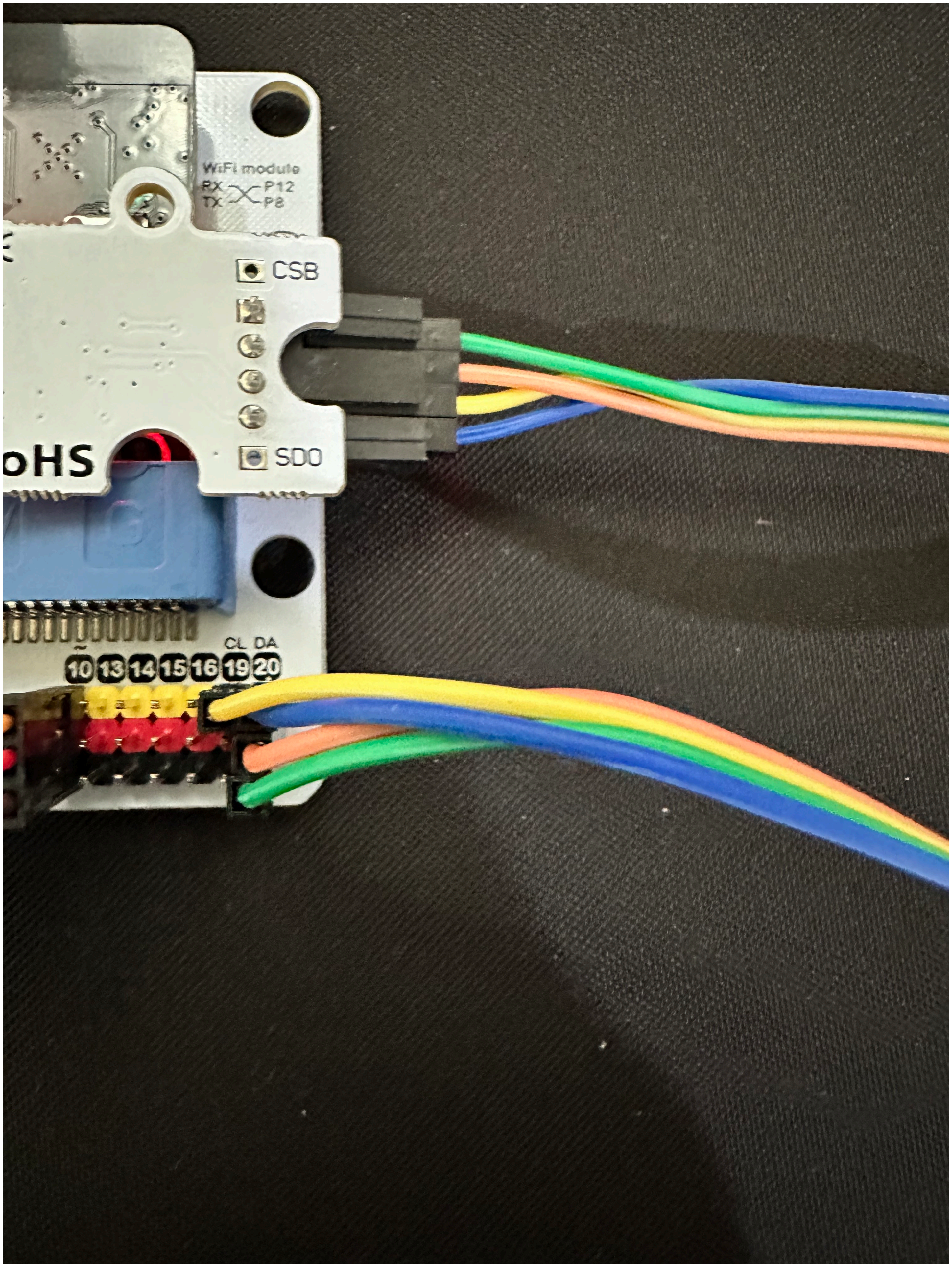
```python
from microbit import *
from iotkit import soil_humidity

soil_humidity = read_soil_humidity(pin7)
```

BME280 Sensor

`read_BME280(state)`

This function reads temperature, humidity, pressure, or altitude from the BME280 sensor and returns the value based on the specified state. Note: The BME280 sensor must be connected to the extension board as shown in the figure.



Arguments:

- `state` - The desired value to read from the BME280 sensor:
  - `0` for temperature (°C)
  - `1` for humidity (%)
  - `2` for pressure (hPa)
  - `3` for altitude (m)

Usage:

```
from microbit import *
from iotkit import read_BME280

temperature = read_BME280(0)
humidity = read_BME280(1)
pressure = read_BME280(2)
altitude = read_BME280(3)
```

Microbit Servo Motor Control

Servo motors can be controlled by sending a specific pulse width to the motor's control wire, which sets the desired angle or position.

Controlling Servo Motors with Micro:bit

The micro:bit board can be used to generate the necessary pulse width to control servo motors. The following code snippet demonstrates how to control a servo motor connected to pin 0 of the micro:bit board:

```
from microbit import *

# Servo control:
# 50 = ~1 millisecond pulse all right
# 75 = ~1.5 millisecond pulse center
# 100 = ~2.0 millisecond pulse all left
pin0.set_analog_period(20)

while True:
    pin0.write_analog(75)
    sleep(1000)
    pin0.write_analog(50)
    sleep(1000)
    pin0.write_analog(100)
    sleep(1000)
```

Code Explanation

1. Import the required `microbit` module:

```
from microbit import *
```

2. Configure the analog period for pin 0:

```
pin0.set_analog_period(20)
```

The `set_analog_period` function sets the period of the PWM signal for the specified pin. In this case, we set the period to 20 milliseconds (ms), which corresponds to a frequency of 50 Hz. This frequency is commonly used for controlling servo motors.

3. Control the servo motor:

The main loop of the program continuously rotates the servo motor to three different positions:

- Center position (1.5 ms pulse):

```
pin0.write_analog(75)
```

- All right position (1 ms pulse):

```
pin0.write_analog(50)
```

- All left position (2 ms pulse):

```
pin0.write_analog(100)
```

Each position is held for 1 second (1000 ms) before moving to the next position.

The table below shows the approximate mapping between the `write_analog` value, pulse width in milliseconds, and position in degrees for a typical servo motor. Note that the actual pulse width to position mapping may vary between different servo motor models, so it's important to consult the datasheet or documentation of the specific servo motor being used for accurate information.

`write_analog` Value	Pulse Width (ms)	Position (degrees)
50	1.0	0°
55	1.1	18°
60	1.2	36°
65	1.3	54°
70	1.4	72°
75	1.5	90° (center)
80	1.6	108°
85	1.7	126°
90	1.8	144°
95	1.9	162°
100	2.0	180°

Keep in mind that these values are approximate and might not precisely match the behavior of your specific servo motor. Always refer to the servo motor's documentation for accurate pulse width to position mapping.

OLED Functions

`initialize_oled()`

This function initializes the OLED display by sending the necessary commands to set it up. It should be called once before any other OLED-related functions are used.

Usage:

```
initialize_oled()
```

`clear_oled()`

This function clears the OLED display by setting all pixels to off.

Usage:

```
clear_oled()
```

`add_text(x, y, text, draw=1)`

This function writes text on the OLED display at the specified position.

Arguments:

- `x` (int): The horizontal position (in characters) where the text should start (0-11).
- `y` (int): The vertical position (in lines) where the text should start (0-3).
- `text` (str): The text to display. The text will be truncated if it exceeds 12 characters.
- `draw` (int, optional): If set to 1 (default), the text will be drawn immediately. If set to 0, the text will be updated in the buffer but not drawn on the display.

Usage:

```
add_text(0, 0, 'Hello, World!')
```

Example usage

The following example demonstrates how to use the OLED functions to display sensor readings from a BME280 sensor:

```
from microbit import *
from iotkit import read_BME280, add_text, initialize_oled, clear_oled

initialize_oled()
clear_oled()

while True:
    sleep(1)
    temperature = read_BME280(0)
    humidity = read_BME280(1)
    pressure = read_BME280(2)
    altitude = read_BME280(3)

    add_text(0, 0, 't: %d' % temperature)
    add_text(0, 1, 'h: %s' % str(humidity))
    add_text(0, 2, 'p: %s' % str(pressure))
    add_text(0, 3, 'a: %s' % str(altitude))
```

In this example, the OLED display is first initialized and cleared. Then, in an infinite loop, sensor readings are obtained and displayed on the OLED screen. The `add_text` function is used to update the display with the latest sensor values.