MICROPYTHON FOR ZING

About MIcropython

MicroPython implements the entire Python 3.4 syntax (including exceptions, with, yield from, etc., and additionally async/await keywords from Python 3.5 and some select features from later versions). The following core datatypes are provided: str(including basic Unicode support), bytes, bytearray, tuple, list, dict, set, frozenset, array.array, collections.namedtuple, classes and instances. Builtin modules include os, sys, time, re, and struct, etc. Select ports have support for _thread module (multithreading), socket and ssl for networking, and asyncio. Note that only a subset of Python 3 functionality is implemented for the data types and modules.

MicroPython can execute scripts in textual source form (.py files) or from precompiled bytecode (.mpy files), in both cases either from an on-device filesystem or "frozen" into the MicroPython executable.

MicroPython also provides a set of MicroPython-specific modules to access hardware-specific functionality and peripherals such as GPIO, Timers, ADC, DAC, PWM, SPI, I2C, CAN, Bluetooth, and USB.

Detailed information regarding MICROPYTHON can be found here.

ZING CUSTOM FIRMWARE

Along with all the functionality of the micropython we have created a custom port for our zing board. We used the GENERIC 32 build structure to compile our own custom firmware. The nice part of our custom firmware is that we have compiled a library which includes all the internal ports and I/O sensors which can be controlled directly without knowing the GPIO number by calling the specific function.

ZING LIBRARY

Zing library has all the internal I/O sensors and ports are included in the library which can be called directly from the library. Overall syntax of common zings accessories are kept the same through various versions for forward compatibility meaning if you wrote a program for one particular version you can use the same program for all the other versions of zing.

ZING V1.0 SYNTAX

INTERNAL PORT NUMBERS VARIABLES

SL NO	PORT	SYNTAX	GPIO
1.	A1	PORT_A1	7
2	A2	PORT_A2	6
3	B1	PORT_B1	5
4	B2	PORT_B2	4
5	C1	PORT_C1	11
6	C2	PORT_C2	12
7	D1	PORT_D1	17
8	D2	PORT_D2	18

Each intel gpio pin can be referenced by calling the corresponding syntax. The bellow examples shows how to control the GPIO on different ports using micropython

EXAMPLES

Controlling GPIO on the specific PORT.

```
from zing import PORT_A1, Pin
from time import sleep


port = Pin(PORT_A1, Pin.OUT)

while True:
   port.value(1)
   sleep(1)
   port.value(0)
   sleep(1)
```

Example 1.0.1 : *Controlling the GPIO of A1 port.*

This example shows How to create an LED blinking program using the A1 port.

Reading from a specific GPIO PORT.

```
from zing import PORT_A1, Pin
from time import sleep

port = Pin(PORT_A1, Pin.IN)
while True:
   a1_val = port.value()
   print(a1_val)
   sleep(1)
```

Example 1.0.2 : *Reading value from GPIO of A1 port.*

The above example shows how to read values from a specific GPIO and print the value to the console. pin object is used to control I/O pins (also known as GPIO - general-purpose input/output). Pin objects are commonly associated with a physical pin that can drive an output voltage and read input voltages. The pin class has methods to set the mode of the pin (IN, OUT, etc) and methods to get and set the digital logic level. A pin object is constructed by using an identifier which unambiguously specifies a certain I/O pin. The allowed forms of the identifier and the physical pin that the identifier maps to are port-specific. Possibilities for the identifier are an integer, a string or a tuple with port and pin number.

ADC Read.

```
from zing import PORT_A1, Pin , ADC
from time import sleep

pot = ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)       #Full range: 3.3v

while True:
   pot_value = pot.read()
   print(pot_value)
   sleep(0.1)
```

Example 1.0.3 : Reading the ADC value.

The above example shows how to read ADC value from a specific GPIO and print the value to the console.

Controlling the inbuilt RGB LEDS.

```
from zing import rgb_led

led = rgb_led()
led.set_pixels(1,255,0,0)
led.update_pixels()
```

Example 1.0.4 : *Controlling the internal RGB LEDS.*

The above example shows how to control the internal RGB LEDS of the zing.

Reading the internal battery percentage.

```
from zing import battery

bat = battery()

while True:
   val = bat.get_level()
   print(val)
```

Example 1.0.14 : reading the battery percentage.

The above example shows how to read inbuilt battery percentage and print on the console.

Reading the inbuilt ACCELEROMETER and GYRO.

```
from zing import IMU

imu = IMU()

while True:
   acc = imu.get_acc()
   gyro = imu.get_acc()
   print(f"ACC_X={acc[0]}")
   print(f"ACC_Y={acc[1]}")
   print(f"ACC_Z={acc[2]}")

   print(f"GYRO_X={gyro[0]}")
   print(f"GYRO_Y={gyro[1]}")
   print(f"GYRO_Z={gyro[2]}")
```

Example 1.0.15 : reading the 3-axis acceleration.

The above example shows how to read the inbuilt accelerometer and print on the console.

Controlling the Serial servo.

```
from zing import SerialServo
from time import sleep

servo = SerialServo()

while True:
   servo.angle_adjust(16,100)
   sleep(1)
   servo.angle_adjust(16,1024)
   sleep(1)
```

Example 1.0.16 : Controlling the serial servo motor of zing
The above example shows how to set angle to zing.

Setting up an OLED screen.

```
from zing import OLED

oled=OLED()

while True:
oled.set_text("Free the MALLOCS !!!!", 1)
oled.show()
```

Example 1.0.16 : Setting up an OLED interface.

Controlling the RGB ultrasonic sensor.

```
from zing import RGB_Ultrasonic

us = RGB_Ultrasonic()

while True:
  id = 1
  dis = us.distance_cm() #reading the distance in cm
  print(dis)
  us.rgb_set_color(255,0,0) #seting the rgbof ultrasonic
```

Example 1.0.16 : Setting up a RGB ULTRASONIC sensor.