RT1010Py

User Manual

olimex.com

Rev.1.0 October 2023

Table of Contents

What is RT1010Py

RT1010Py is development board with MIMXRT1011DAE5A Cortex-M7 processor running at 500Mhz i.e. 4 times faster than RP2040.

RT1010Py runs MicroPython tanks to Robert Hammelrath (robert@hammelrath.com)

RT1010Py has these features:

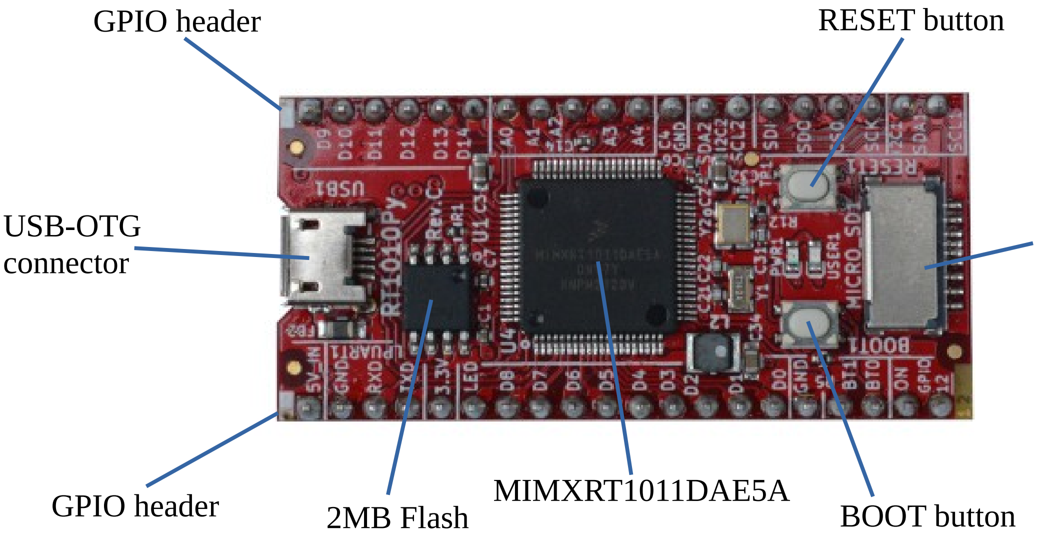
- MIMXRT1011DAE5A
- 128KB on board RAM
- 2MB SPI Flash
- two SPI
- two I2C
- four PWM
- USB 2.0 OTG
- micro SD card connector
- RTC with 32.768 kHz crystal
- RESET button
- BOOT button
- mUEXT connector with 3.3V, GND, I2C, SPI, and UART
- two GPIOs headers spaced at 22.86 mm (0.9”)
- Dimensions: 53.34 x 25.4 mm (2.1 x 1”)

Order codes for RT1010Py and accessories:

RT1010Py	RT1011 board running at 500Mhz with MicroPython
USB-CABLE-A-MICRO-1.8M	USB-A to micro cable
MICRO-SD-16GB-CLASS10	16GB microSD card
RT1010Py-DevKit	evaluation board for RT1010Py with two relays, two UEXT, USB-C
UEXT modules	There are temperature, humidity, pressure, magnetic field, light sensors. Modules with LCDs, LED matrix, Relays, Bluetooth, Zigbee, WiFi, GSM, GPS, RFID, RTC, EKG, sensors and etc.

HARDWARE

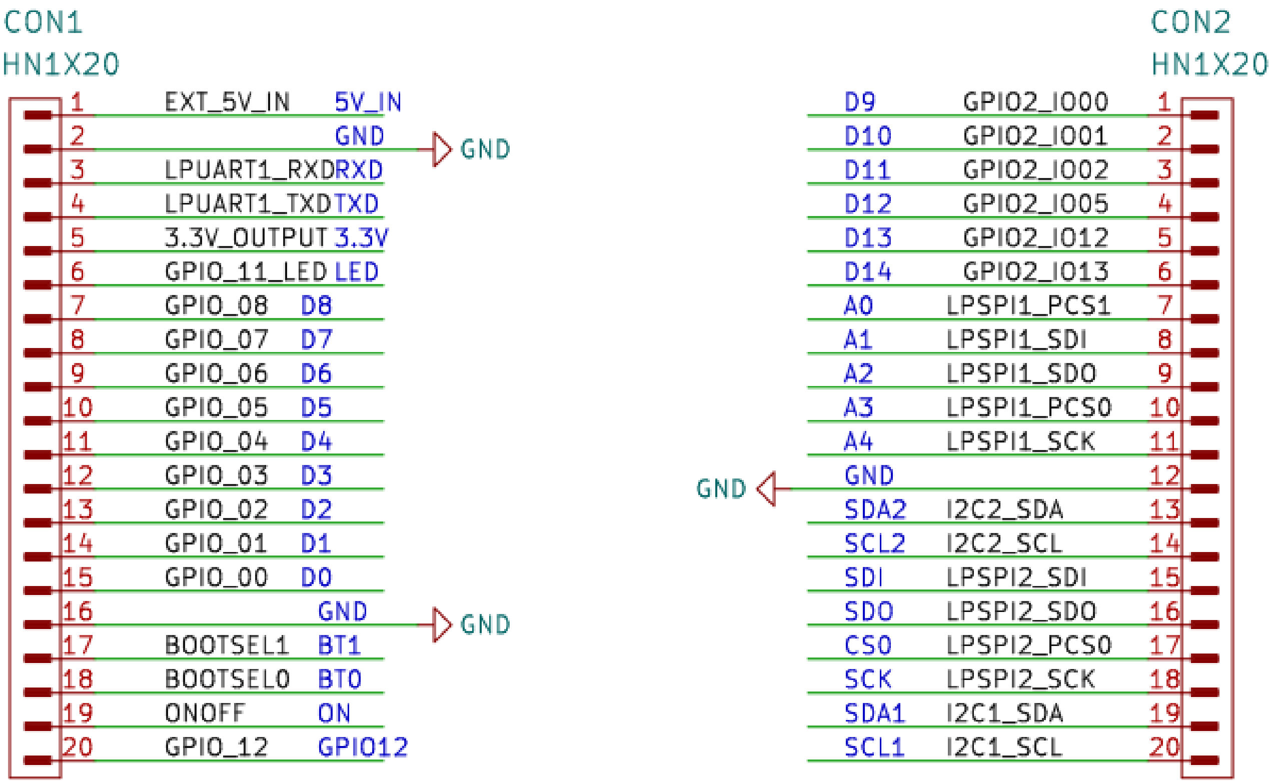
RT1010Py layout:



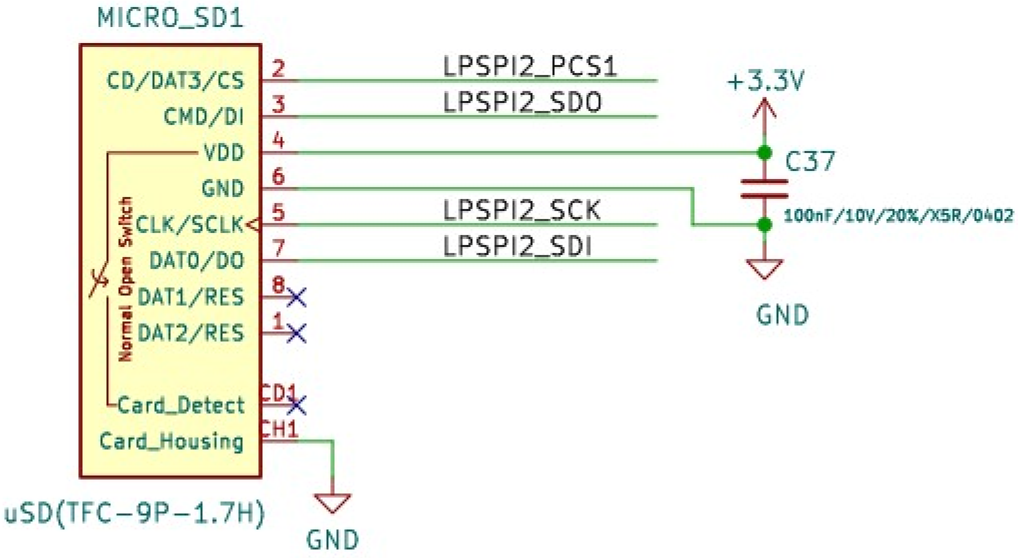
RT1010Py schematics:

RT1010Py latest schematic is on GitHub

GPIO connectors:



SD card connector:

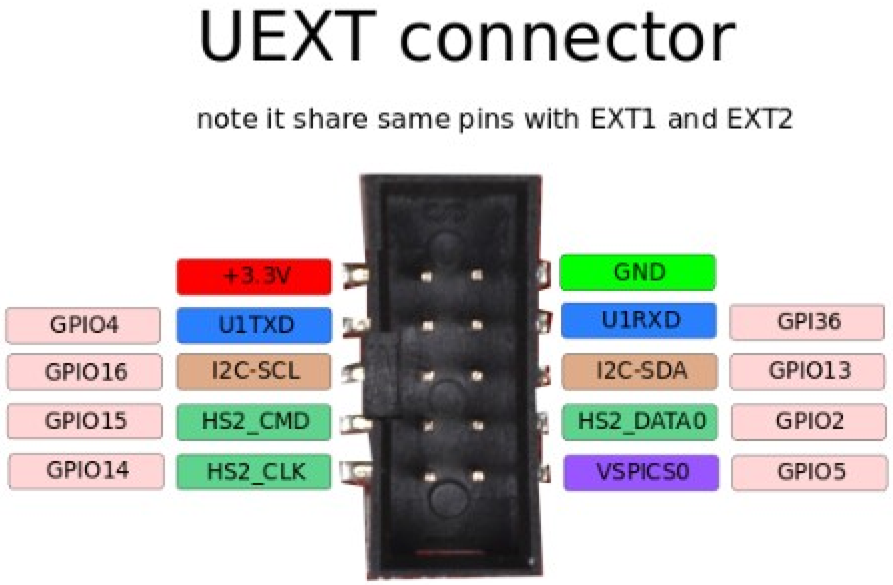


UEXT connector:

UEXT connector stands for Universal EXTension connector and contain +3.3V, GND, I2C, SPI, UART signals.

UEXT connector can be in different shapes.

The original UEXT connector is 0.1” 2.54mm step boxed plastic connector. All signals are with 3.3V levels.



As the boards become smaller and smaller some smaller packages were introduced too beside the original UEXT connector

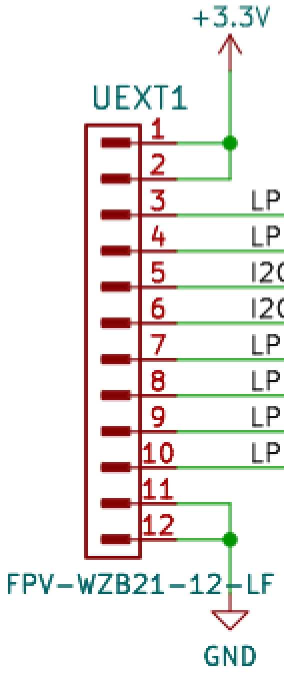
- mUEXT is 1.27 mm step boxed header connector which is with same layout as UEXT

- pUEXT is 1.0 mm single row connector (this is the connector used in RP2040-PICO30)

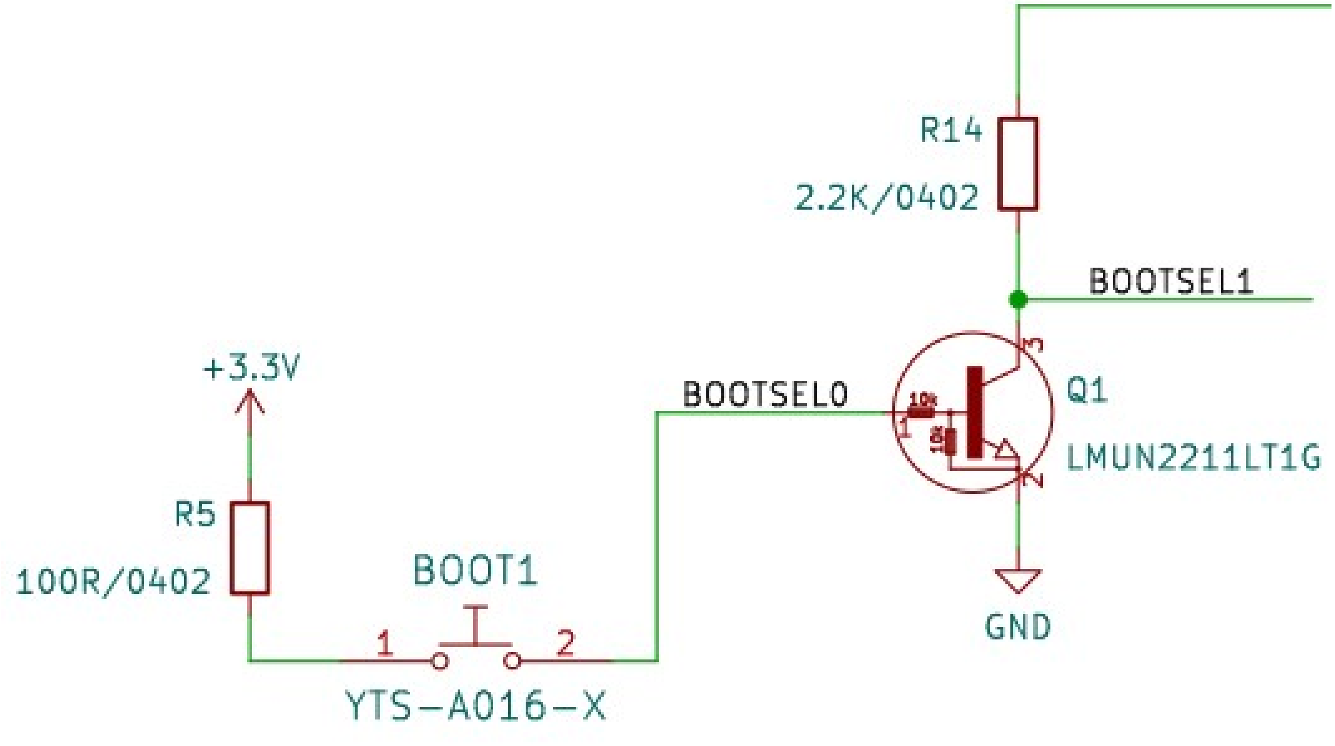
- fUEXT is Flat cable 0.5 mm step connector

Olimex has developed number of MODULES with this connector. There are temperature, humidity, pressure, magnetic field, light sensors. Modules with LCDs, LED matrix, Relays, Bluetooth, Zigbee, WiFi, GSM, GPS, RFID, RTC, EKG, sensors and etc.

RT1010Py fUEXT connector:



RT1010Py boot configuration:



USER / BOOT

released 10 Internal Boot
pressed 01 Serial Downloader

SOFTWARE:

MicroPython bootloader and firmware instalation:

Detailed instructions how to install MicroPython bootloader and firmware are here: https://micropython.org/download/OLIMEX_RT1010/

- Get the files ufconv.py and uf2families.json from the micropython/tools directory, e.g. at https://github.com/micropython/micropython/tree/master/tools.
- Get the NXP program sdphost for your operating system, e.g. from https://github.com/adafruit/tinyuf2/tree/master/ports/mimxrt10xx/sdphost. You can also get them from the NXP web sites.
- Get the UF2 boot-loader package https://github.com/adafruit/tinyuf2/releases/download/0.9.0/tinyuf2-imxrt1010_evk-0.9.0.zip and extract the file tinyuf2-imxrt1010_evk-0.9.0.bin

Now you have all files at hand that you will need for updating.

1. Get the firmware you want to upload from the MicroPython download page.
2. Push and hold the "Boot" button, then press "Reset", and release both buttons.
3. Run the commands:

```
sudo ./sdphost -u 0x1fc9,0x0145 -- write-file 0x20206400 tinyuf2-imxrt1010_evk-0.9.0.bin
sudo ./sdphost -u 0x1fc9,0x0145 -- jump-address 0x20207000
```

Wait until a drive icon appears on the computer (or mount it explicitly), and then run:

```
python3 uf2conv.py <firmware_xx.yy.zz.hex> --base 0x60000400 -f 0x4fb2d5bd
```

You can put all of that in a script. Just add a short wait before the 3rd command to let the drive connect.

4. Once the upload is finished, push Reset again.

Using sudo is Linux specific. You may not need it at all, if the access rights are set properly, and you will not need it for Windows.

Once the generic boot-loader is available, this procedure is only required for the first firmware load or in case the flash is corrupted and the existing firmware is not functioning any more.

MicroPython Firmware

Releases

v1.21.0 (2023-10-05) .hex / [.bin] / [Release notes] (latest)
v1.20.0 (2023-04-26) .hex / [.bin] / [Release notes]
v1.19.1 (2022-06-18) .hex / [.bin] / [Release notes]

Preview builds

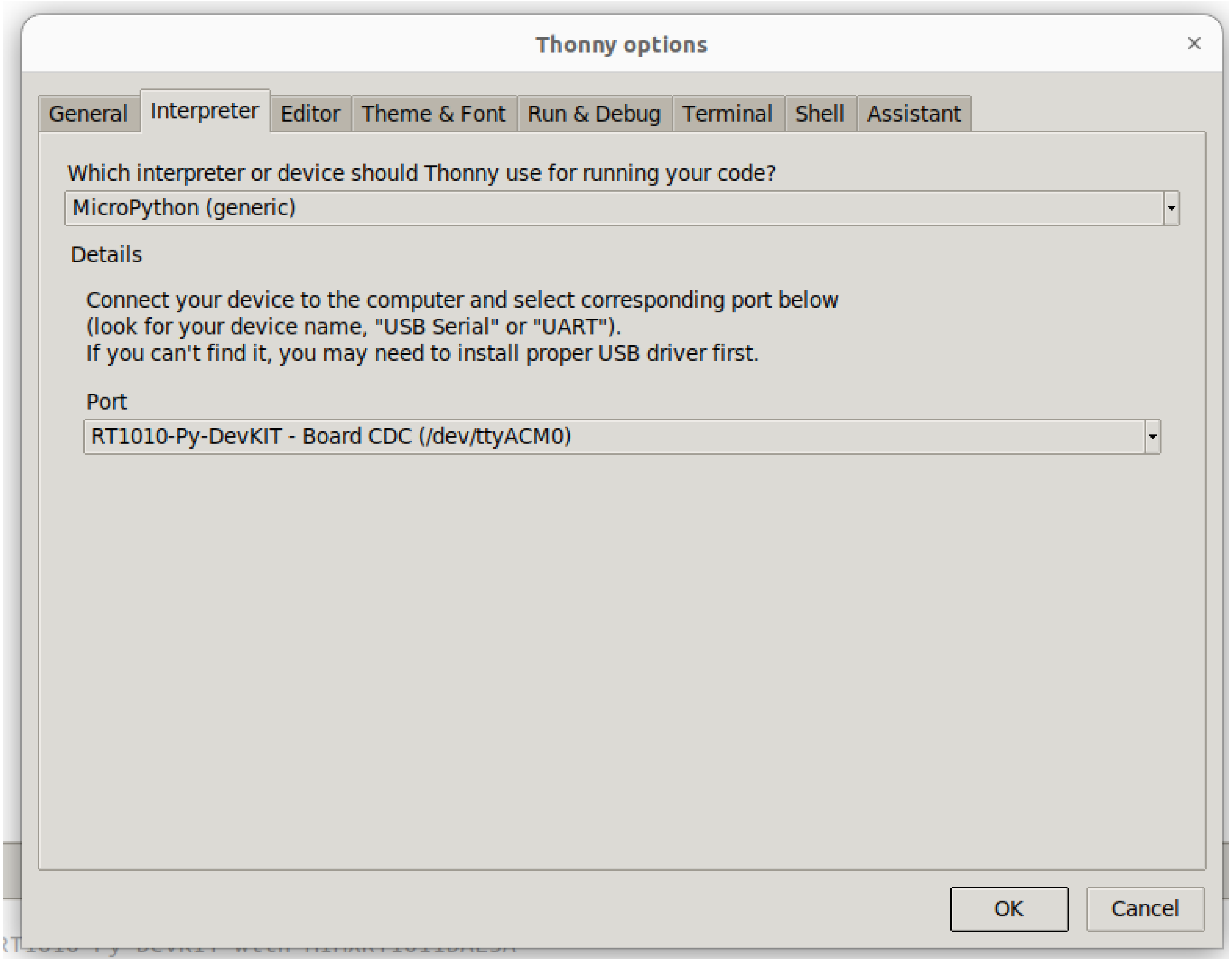
v1.22.0-preview.31.g3883f2948 (2023-10-17) .hex / [.bin]
v1.22.0-preview.30.ge78471416 (2023-10-17) .hex / [.bin]
v1.22.0-preview.27.gc2361328e (2023-10-17) .hex / [.bin]
v1.22.0-preview.24.g51da8cc28 (2023-10-17) .hex / [.bin]
(These are automatic builds of the development branch for the next release)

Working with MicroPython and Thonny:

Install Thonny with:

```
   $ sudo apt install thonny
```

Plug USB cable to RT1010Py and run Thonny. From the Run menu select interpreter:



Which interpreter or device should Thonny use for running your code?

MicroPython (generic)

Details

Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Port

RT1010-Py-DevKIT - Board CDC (/dev/ttyACM0)

Now you are ready to make your first embedded hello world program i.e. to blink an LED.

Python programming with RT1010Py:

Check at what frequency RT1010Py processor is running:

```
import machine

machine.freq()
```

Delay and timing:

```
import time

time.sleep(1)           # sleep for 1 second
time.sleep_ms(500)      # sleep for 500 milliseconds
time.sleep_us(10)       # sleep for 10 microseconds
start = time.ticks_ms() # get millisecond counter
delta = time.ticks_diff(time.ticks_ms(), start) # compute time difference
```

Timers:

```
from machine import Timer

tim0 = Timer(-1)
tim0.init(period=5000, mode=Timer.ONE_SHOT, callback=lambda t:print(0))

tim1 = Timer(-1)
tim1.init(period=2000, mode=Timer.PERIODIC, callback=lambda t:print(1))
```

blinking LED with Timer

```
from machine import Pin, Timer

led = Pin("LED", Pin.OUT)

tim = Timer()

def tick(timer):

    global led

    led.toggle()
```

```
   tim.init(freq=2.5, mode=Timer.PERIODIC, callback=tick)
```

GPIOs:

```
   from machine import Pin

   led = Pin(‘LED’,Pin.OUT)
   led.on()
   led.off()
```

valid GPIO names are: D0..D14, LED, A0..A4 , GPIO_00..GPIO_14, some of them duplicate for instance GPIO_00 is D0

UART:

```
from machine import UART

uart1 = UART(1, baudrate=115200) #Tx and Rx mark
uart1.write('hello')  # write 5 bytes
uart1.read(5)         # read up to 5 bytes

uart2 = UART(2, baudrate=115200) #D5 - Rx, D6 - Tx
uart2.write('hello')  # write 5 bytes
uart2.read(5)         # read up to 5 bytes

uart3 = UART(3, baudrate=115200) #D7 - Rx, D8 - Tx
uart3.write('hello')  # write 5 bytes
uart3.read(5)         # read up to 5 bytes
```

PWM pin assignment:

Pin	Olimex RT1010PY
D0	•
D1	F1/0/B
D2	F1/0/A
D3	F1/1/B
D4	F1/1/A
D5	F1/2/B
D6	F1/2/A
D7	F1/3/B
D8	F1/3/A
D9	•
D10	F1/0/B
D11	F1/0/A
D12	F1/1/B
D13	F1/1/A
D14	•
A0	•
A1	F1/2/B
A2	F1/2/A

Pin	Olimex RT1010PY
A3	F1/3/B
A4	F1/3/A
SDI	F1/3/X
SDO	F1/2/X
CS0	F1/1/X
SCK	F1/0/X

- Fm/n/l: FLEXPWM module m, submodule n, channel l. The pulse at a X channel is always aligned to the period start.

Make breathing LED connected to D1 GPIO:

```
import time
from machine import Pin, PWM

pwm = PWM(Pin('D1'))

pwm.freq(1000)

duty = 0
direction = 1
for x in range(8 * 256):
    duty += direction
    if duty > 255:
        duty = 255
        direction = -1
    elif duty < 0:
        duty = 0
        direction = 1
    pwm.duty_u16(duty * duty)
    time.sleep(0.001)
```

ADC

```
from machine import ADC

adc = ADC(‘A0’)      # create ADC object on ADC pin
adc.read_u16()      # read value, 0-65536 across voltage range 0.0v - 3.3v
```

SPI bus:

Software SPI:

Works on all pins.

```
from machine import Pin, SoftSPI

# construct a SoftSPI bus on the given pins
# polarity is the idle state of SCK
# phase=0 means sample on the first edge of SCK, phase=1 means the second
spi = SoftSPI(baudrate=100000, polarity=1, phase=0, sck=Pin('D1'), mosi=Pin('D2'),
miso=Pin('D3'))

spi.init(baudrate=200000) # set the baudrate

spi.read(10)            # read 10 bytes on MISO
spi.read(10, 0xff)      # read 10 bytes while outputting 0xff on MOSI

buf = bytearray(50)     # create a buffer
spi.readinto(buf)       # read into the given buffer (reads 50 bytes in this case)
spi.readinto(buf, 0xff) # read into the given buffer and output 0xff on MOSI

spi.write(b'12345')     # write 5 bytes on MOSI

buf = bytearray(4)      # create a buffer
spi.write_readinto(b'1234', buf) # write to MOSI and read from MISO into the buffer
spi.write_readinto(buf, buf) # write buf to MOSI and read MISO back into buf
```

Hardware SPI:

There are two Hardware SPIs . They can work up to 30Mhz clock.

The first is available as CS0 SDO SDI SCK, the second is connected to the SD Card with CS1

```
from machine import SPI, Pin

spi = SPI(1, 10000000)
spi.write('Hello World')
```

I2C:

Software I2C:

Software I2C (using bit-banging) works on all output-capable pins

```
from machine import Pin, SoftI2C

i2c = SoftI2C(scl=Pin('D1'), sda=Pin('D2'), freq=100000)

i2c.scan()              # scan for devices

i2c.readfrom(0x3a, 4)   # read 4 bytes from device with address 0x3a
i2c.writeto(0x3a, '12') # write '12' to device with address 0x3a

buf = bytearray(10)     # create a buffer with 10 bytes
i2c.writeto(0x3a, buf)  # write the given buffer to the slave
```

Hardware I2C:

Two hardware I2C are available SDA1/SCL1 and SDA2/SCL2

```
from machine import I2C
i2c = I2C(1, 400_000)
i2c.scan()
i2c.writeto(0x76, b"Hello World")
```

I2S

Example:

```
from machine import I2S, Pin

i2s = I2S(3, sck=Pin('D10'), ws=Pin('D9'), sd=Pin('D11'), mode=I2S.TX,
bits=16,format=I2S.STEREO, rate=44100,ibuf=4000)

i2s.write(buf)             # write buffer of audio samples to I2S device
```

Real time clock (RTC)

Example:

```
from machine import RTC

rtc = RTC()
rtc.datetime((2017, 8, 23, 1, 12, 48, 0, 0)) # set a specific date and time
rtc.datetime() # get date and time
rtc.now() # return date and time in CPython format.
```

SD card

You need sdcard.py driver written with Thonny to RT1010Py:

```
"""
MicroPython driver for SD cards using SPI bus.

Requires an SPI bus and a CS pin.  Provides readblocks and writeblocks
methods so the device can be mounted as a filesystem.

Example usage on pyboard:

    import pyb, sdcard, os
    sd = sdcard.SDCard(pyb.SPI(1), pyb.Pin.board.X5)
    pyb.mount(sd, '/sd2')
    os.listdir('/')

Example usage on ESP8266:

    import machine, sdcard, os
    sd = sdcard.SDCard(machine.SPI(1), machine.Pin(15))
    os.mount(sd, '/sd')
    os.listdir('/')

"""

from micropython import const
import time


_CMD_TIMEOUT = const(100)

_R1_IDLE_STATE = const(1 << 0)
# R1_ERASE_RESET = const(1 << 1)
_R1_ILLEGAL_COMMAND = const(1 << 2)
# R1_COM_CRC_ERROR = const(1 << 3)
# R1_ERASE_SEQUENCE_ERROR = const(1 << 4)
# R1_ADDRESS_ERROR = const(1 << 5)
# R1_PARAMETER_ERROR = const(1 << 6)
_TOKEN_CMD25 = const(0xFC)
_TOKEN_STOP_TRAN = const(0xFD)
_TOKEN_DATA = const(0xFE)


class SDCard:
    def __init__(self, spi, cs, baudrate=1320000):
        self.spi = spi
        self.cs = cs

        self.cmdbuf = bytearray(6)
        self.dummybuf = bytearray(512)
        self.tokenbuf = bytearray(1)
        for i in range(512):
            self.dummybuf[i] = 0xFF
        self.dummybuf_memoryview = memoryview(self.dummybuf)

        # initialise the card
        self.init_card(baudrate)

    def init_spi(self, baudrate):
        try:
            master = self.spi.MASTER
        except AttributeError:
            # on ESP8266
            self.spi.init(baudrate=baudrate, phase=0, polarity=0)
        else:
            # on pyboard
            self.spi.init(master, baudrate=baudrate, phase=0, polarity=0)

    def init_card(self, baudrate):

        # init CS pin
        if self.cs is not None:
            self.cs.init(self.cs.OUT, value=1)

        # init SPI bus; use low data rate for initialisation
        self.init_spi(120000)

        # clock card at least 100 cycles with cs high
        for i in range(16):
            self.spi.write(b"\xff")

        # CMD0: init card; should return _R1_IDLE_STATE (allow 5 attempts)
        for _ in range(5):
            if self.cmd(0, 0, 0x95) == _R1_IDLE_STATE:
                break
        else:
            raise OSError("no SD card")

        # CMD8: determine card version
        r = self.cmd(8, 0x01AA, 0x87, 4)
        if r == _R1_IDLE_STATE:
            self.init_card_v2()
        elif r == (_R1_IDLE_STATE | _R1_ILLEGAL_COMMAND):
            self.init_card_v1()
        else:
            raise OSError("couldn't determine SD card version")

        # get the number of sectors
        # CMD9: response R2 (R1 byte + 16-byte block read)
        if self.cmd(9, 0, 0, 0, False) != 0:
            raise OSError("no response from SD card")
        csd = bytearray(16)
        self.readinto(csd)
        if csd[0] & 0xC0 == 0x40:  # CSD version 2.0
            self.sectors = ((csd[8] << 8 | csd[9]) + 1) * 1024
        elif csd[0] & 0xC0 == 0x00:  # CSD version 1.0 (old, <=2GB)
            c_size = csd[6] & 0b11 | csd[7] << 2 | (csd[8] & 0b11000000) << 4
            c_size_mult = ((csd[9] & 0b11) << 1) | csd[10] >> 7
            self.sectors = (c_size + 1) * (2 ** (c_size_mult + 2))
        else:
            raise OSError("SD card CSD format not supported")
        # print('sectors', self.sectors)

        # CMD16: set block length to 512 bytes
        if self.cmd(16, 512, 0) != 0:
            raise OSError("can't set 512 block size")

        # set to high data rate now that it's initialised
        self.init_spi(baudrate)

    def init_card_v1(self):
        for i in range(_CMD_TIMEOUT):
            self.cmd(55, 0, 0)
            if self.cmd(41, 0, 0) == 0:
                self.cdv = 512
                # print("[SDCard] v1 card")
                return
        raise OSError("timeout waiting for v1 card")

    def init_card_v2(self):
        for i in range(_CMD_TIMEOUT):
            time.sleep_ms(50)
            self.cmd(58, 0, 0, 4)
            self.cmd(55, 0, 0)
            if self.cmd(41, 0x40000000, 0) == 0:
                self.cmd(58, 0, 0, 4)
                self.cdv = 1
                # print("[SDCard] v2 card")
                return
        raise OSError("timeout waiting for v2 card")

    def cmd(self, cmd, arg, crc, final=0, release=True, skip1=False):
        self.set_cs(0)

        # create and send the command
        buf = self.cmdbuf
        buf[0] = 0x40 | cmd
        buf[1] = arg >> 24
        buf[2] = arg >> 16
        buf[3] = arg >> 8
        buf[4] = arg
        buf[5] = crc
        self.spi.write(buf)

        if skip1:
            self.spi.readinto(self.tokenbuf, 0xFF)

        # wait for the response (response[7] == 0)
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            response = self.tokenbuf[0]
            if not (response & 0x80):
                # this could be a big-endian integer that we are getting here
                for j in range(final):
                    self.spi.write(b"\xff")
                if release:
                    self.set_cs(1)
                    self.spi.write(b"\xff")
                return response

        # timeout
        self.set_cs(1)
        self.spi.write(b"\xff")
        return -1

    def readinto(self, buf):
        self.set_cs(0)

        # read until start byte (0xff)
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            if self.tokenbuf[0] == _TOKEN_DATA:
                break
            time.sleep_ms(1)
        else:
            self.set_cs(1)
            raise OSError("timeout waiting for response")

        # read data
        mv = self.dummybuf_memoryview
        if len(buf) != len(mv):
            mv = mv[: len(buf)]
        self.spi.write_readinto(mv, buf)

        # read checksum
        self.spi.write(b"\xff")
        self.spi.write(b"\xff")

        self.set_cs(1)
        self.spi.write(b"\xff")

    def write(self, token, buf):
        self.set_cs(0)

        # send: start of block, data, checksum
        self.spi.read(1, token)
        self.spi.write(buf)
        self.spi.write(b"\xff")
        self.spi.write(b"\xff")

        # check the response
        if (self.spi.read(1, 0xFF)[0] & 0x1F) != 0x05:
            self.set_cs(1)
            self.spi.write(b"\xff")
            return

        # wait for write to finish
        while self.spi.read(1, 0xFF)[0] == 0:
            pass

        self.set_cs(1)
        self.spi.write(b"\xff")
    def write_token(self, token):
        self.set_cs(0)
        self.spi.read(1, token)
        self.spi.write(b"\xff")
        # wait for write to finish
        while self.spi.read(1, 0xFF)[0] == 0x00:
            pass

        self.set_cs(1)
        self.spi.write(b"\xff")

    def readblocks(self, block_num, buf):
        nblocks = len(buf) // 512
        assert nblocks and not len(buf) % 512, "Buffer length is invalid"
        if nblocks == 1:
            # CMD17: set read address for single block
            if self.cmd(17, block_num * self.cdv, 0, release=False) != 0:
                # release the card
                self.set_cs(1)
                raise OSError(5)  # EIO
            # receive the data and release card
            self.readinto(buf)
        else:
            # CMD18: set read address for multiple blocks
            if self.cmd(18, block_num * self.cdv, 0, release=False) != 0:
                # release the card
                self.set_cs(1)
                raise OSError(5)  # EIO
            offset = 0
            mv = memoryview(buf)
            while nblocks:
                # receive the data and release card
                self.readinto(mv[offset : offset + 512])
                offset += 512
                nblocks -= 1
            if self.cmd(12, 0, 0xFF, skip1=True):
                raise OSError(5)  # EIO

    def writeblocks(self, block_num, buf):
        nblocks, err = divmod(len(buf), 512)
        assert nblocks and not err, "Buffer length is invalid"
        if nblocks == 1:
            # CMD24: set write address for single block
            if self.cmd(24, block_num * self.cdv, 0) != 0:
                raise OSError(5)  # EIO

            # send the data
            self.write(_TOKEN_DATA, buf)
        else:
            # CMD25: set write address for first block
            if self.cmd(25, block_num * self.cdv, 0) != 0:
                raise OSError(5)  # EIO
            # send the data
            offset = 0
            mv = memoryview(buf)
            while nblocks:
                self.write(_TOKEN_CMD25, mv[offset : offset + 512])
                offset += 512
                nblocks -= 1
            self.write_token(_TOKEN_STOP_TRAN)

    def ioctl(self, op, arg):
        if op == 4:  # get number of blocks
            return self.sectors

    def set_cs(self, value):
        if self.cs is not None:
            self.cs(value)
```

then you can use it to test the SD card functionality:

```
# Test for sdcard block protocol
# Peter hinch 30th Jan 2016
import os, sdcard, machine
import gc


def sdtest():
    spi = machine.SPI(2)
    spi.init()  # Ensure right baudrate
    cs = machine.Pin(machine.Pin.cpu.GPIO_AD_01, machine.Pin.OUT, value=0)
    sd = sdcard.SDCard(spi, cs)  # Compatible with PCB
    vfs = os.VfsFat(sd)
    os.mount(vfs, "/fc")
    print("Filesystem check")
    print(os.listdir("/fc"))

    gc.collect()
    line = "abcdefghijklmnopqrstuvwxyz\n"
    lines = line * 100  # 2700 chars
    short = "1234567890\n"

    fn = "/fc/rats.txt"
    print()
    print("Multiple block read/write")
    with open(fn, "w") as f:
        n = f.write(lines)
        print(n, "bytes written")
        n = f.write(short)
        print(n, "bytes written")
        n = f.write(lines)
        print(n, "bytes written")

    with open(fn, "r") as f:
        result1 = f.read()
        print(len(result1), "bytes read")

    fn = "/fc/rats1.txt"
    print()
    print("Single block read/write")
    with open(fn, "w") as f:
        n = f.write(short)  # one block
        print(n, "bytes written")

    with open(fn, "r") as f:
        result2 = f.read()
        print(len(result2), "bytes read")

    os.umount("/fc")

    print()
    print("Verifying data read back")
    success = True
    gc.collect()
    if result1 == "".join((lines, short, lines)):
        print("Large file Pass")
    else:
        print("Large file Fail")
        success = False
    if result2 == short:
        print("Small file Pass")
    else:
        print("Small file Fail")
        success = False
    print()
    print("Tests", "passed" if success else "failed")
```

the result will be something like this:

```
>>> sdtest()
Filesystem check
['System Volume Information', 'Agon-CPM2.2', 'basic_examples_tests', 'mos',
'autoexec.txt', 'bbcbasic.bin', 'README.md']

Multiple block read/write
2700 bytes written
11 bytes written
2700 bytes written
5411 bytes read

Single block read/write
11 bytes written
11 bytes read

Verifying data read back
Large file Pass
Small file Pass

Tests passed
```

OneWire driver:

MicroPython can read OneWire devices like SNS-TMP-DS18B20-MAXIM.

The connection should be: Black to GND, Red to +3.3V, Yellow to D12. There must be also 4.7K resistor connected between D12 and +3.3V

```
from machine import Pin
import onewire, time, ds18x20

ow = onewire.OneWire(Pin(12))
ds = ds18x20.DS18X20(ow)
roms = ds.scan()
ds.convert_temp()
time.sleep_ms(750)
for rom in roms:
    print(ds.read_temp(rom))
```

Revision History

Revision 1.0 October 2023 initial