Micropython on a Teensy 3.2 ?

By jean-claude.feltes@education.lu

I was trying to build this measuring box (voltages, currents…) with an ESP32. It seemed promising as the ESP32 has a file system, the data could be stored there on the chip. WiFi was not so important, but the ESP32 has got it, and it would be nice to have.
Why not use C for programming? For me, the answer was clear: I like to avoid it because it is ugly. Python is as beautiful and clear as BASIC is.

But the ESP32 disappointed me. It has so many peripherals on a chip for so little money, there has to be a backdraft! And there was.
The ADCs showed a large fluctuation in the values. I succeeded in reducing it by taking an average over 50 values, with a timing adjusted to one 50Hz period to suppress hum. But there still was a very nonlinear behaviour, especially at low input voltages.
Even a Mega8 would do a better job!
For a moment I wanted to go back to this alternative (or use an external ADC), when I saw that the Teensy could be flashed to use Micropython.

This was not so straightforward. You have to compile Micropython. And to do this you have to install the ARM toolchain.
I found a very good tutorial here that shows how to load Micropython on a Teensy3.x:

https://learn.sparkfun.com/tutorials/how-to-load-micropython-on-a-microcontroller-board/teensy-3x

The first attempt was unsuccessful. There was a library conflict between the AVR and ARM toolchain. A quick web search showed that I was not the only one having this problem.
Frustrating!

The next day I tried to build Micropython on another nearly blank computer. And I succeded and could load Micropython to the Teensy.

Happy as I was, I started a serial terminal (GTKterm, 115200 baud) and hit the <Enter> key. There it was, the REPL prompt “>>>”. Wow!
As a first test, I used Python as a calculator:

```
>>> 3+5
8
```

Fantastic! The Teensy could do calculations.

The example code from the tutorial showed how to switch on a LED:

```
import pyb
led = pyb.Pin.board.LED
led.value(1)
```

This worked, too.
But it showed already a difference to the Micropython I was used to.
The Pin class is not found in “machine”, but in “pyb” (like for the pyBoard).

I came back to my main computer and plugged in the Teensy. I wanted to use Thonny as IDE, as it had done a good job for the ESPs.
What? No reaction? OK, keep cool, select the right interpreter and port from the Run menu.
Now the prompt was to be seen, the file system was blank, unlike the ESP32 that showed already a rudimentary main.py and boot.py.

Was there even a filesystem?

```
>>> f = open('data.txt', 'w')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'open' isn't defined
```

That was not really promising!
Well, not too astonishing as there is not a lot of storage space on the Teensy:
256K Flash, 256K RAM, 2K EEPROM.
(Some days after this, I found the answer, see below)

And how should I upload my Python script?
Thonny doesn’t do it.
I remembered having used ampy before Thonny to load scripts to an ESP8266.

A first test was promising:

```
ampy -p /dev/ttyACM0 -b 115200 reset
```

Ok, reset worked.

Now let’s list eventual files:

```
ampy -p /dev/ttyACM0 -b 115200 ls
```

No, this was producing an error.

Was it possible to load a script file and execute it?

I wrote a short blink script into a file test.py:

```
import pyb
led = pyb.Pin.board.LED
while True:
      led.value(1)
      pyb.delay(1000)
      led.value(0)
      pyb.delay(1000)
```

and uploaded it:

```
ampy -p /dev/ttyACM0 -b 115200 run test.py
```

My Teensy was blinking!
But ampy was stuck. Aborting it with Ctrl-C worked, the Teensy continued blinking.

A hard reset stopped the blinking program.
After that (no surprise) my script was gone on the Teensy.

And now?
My conclusion is that for the moment (january 2021) and probably after that (because of the small storage space) I cannot use the Teensy 3.2 as I had used the ESP8266 or ESP32 before: upload script(s) and run a boot.py and main.py and after this scripts imported by main.py.

So I have just the REPL, I can interactively do some things like lightin up a LED etc., but that is not really much.

To use the Teensy, I will have to use this ugly looking language and use lots of {}; that prevent my code from being clear and concise ...

Update (23.1.2021):
No, there is no file system:
https://github.com/micropython/micropython/wiki/Board-Teensy-3.1-3.5-3.6

“Currently, scripts must be compiled into the firmware image. Place your .py files into the teensy/scripts directory and they'll be built into your firmware image.
The sample scripts directory includes a boot.py and main.py. The default main.py flashes the LED twice.

If you want to update a script, you'll need to rebuild and reflash the firmware.”

That makes using a Teensy3 with Micropython to something for hardcore Python enthusiasts, not for “normal” programmers.

Appendix: available commads

```
>>> help()
Welcome to MicroPython!

For online help please visit http://micropython.org/help/.

Quick overview of commands for the board:
  pyb.info()    -- print some general information
  pyb.gc()      -- run the garbage collector
  pyb.delay(n)  -- wait for n milliseconds
  pyb.Switch()  -- create a switch object
                   Switch methods: (), callback(f)
  pyb.LED(n)    -- create an LED object for LED n (n=1,2,3,4)
                   LED methods: on(), off(), toggle(), intensity(<n>)
  pyb.Pin(pin)  -- get a pin, eg pyb.Pin('X1')
  pyb.Pin(pin, m, [p]) -- get a pin and configure it for IO mode m, pull mode p
                   Pin methods: init(..), value([v]), high(), low()
  pyb.ExtInt(pin, m, p, callback) -- create an external interrupt object
  pyb.ADC(pin)  -- make an analog object from a pin
                   ADC methods: read(), read_timed(buf, freq)
  pyb.DAC(port) -- make a DAC object
                   DAC methods: triangle(freq), write(n), write_timed(buf, freq)
  pyb.RTC()     -- make an RTC object; methods: datetime([val])
  pyb.rng()     -- get a 30-bit hardware random number
  pyb.Servo(n)  -- create Servo object for servo n (n=1,2,3,4)
                   Servo methods: calibration(..), angle([x, [t]]), speed([x, [t]])
  pyb.Accel()   -- create an Accelerometer object
                   Accelerometer methods: x(), y(), z(), tilt(), filtered_xyz()

Pins are numbered X1-X12, X17-X22, Y1-Y12, or by their MCU name
Pin IO modes are: pyb.Pin.IN, pyb.Pin.OUT_PP, pyb.Pin.OUT_OD
Pin pull modes are: pyb.Pin.PULL_NONE, pyb.Pin.PULL_UP, pyb.Pin.PULL_DOWN
Additional serial bus objects: pyb.I2C(n), pyb.SPI(n), pyb.UART(n)

Control commands:
  CTRL-A        -- on a blank line, enter raw REPL mode
  CTRL-B        -- on a blank line, enter normal REPL mode
  CTRL-C        -- interrupt a running program
  CTRL-D        -- on a blank line, do a soft reset of the board

For further help on a specific object, type help(obj)
```