The communication with a Raspi Pico

Rethinking communication with a controller when Micropython runs on it.
By jean-claude.feltes@education.lu

This document illustrates the path I was following when I tried to use a Pico to control a DDS generator I had in my drawer from earlier times (the project was published in ELEKTOR 5/1995) Though I focus on this project, the reflections may be of general interest for other projects. Tell me if you agree or disagree.

1. “Normal” controllers (AVR, Arduino, Teensy…)

My favorite approach was this: To select an action, the computer sends one byte meaning a command. This is received by serial interrupt. The main loop looks if there is any received byte, if yes it calls a subprogram (function) that calls the function corresponding to the command.

If more interaction is needed e.g. to change the value of a variable, things get a little more complicated, as input statements have to be used.

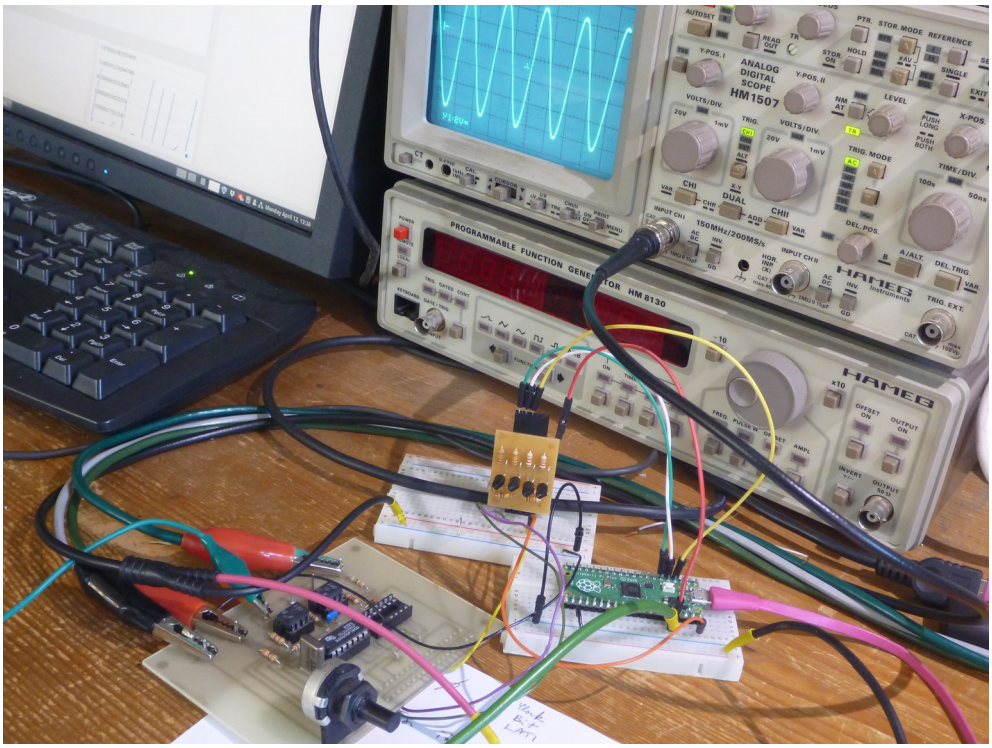
```
Serialinterrupt:
        Ok = Udr
Return


Menu:
        Select Case Ok
                Case"?" : GosubHelp
                Case "v": PrintVersion()
                '...
                Case "%": Gosub Set_calibration_factors
                Case "!": Gosub Set_default_calibration

        End Select
        Ok = 0
Return


Do

      'user action by serial interrupt -> menu
       If Ok > 0 Then Gosub Menu

       '...

Loop
```

2. Trying this with the Pico?

(Or any other controller running Micropython)

One of my first Pico projects was a sine wave generator that should be able to work as a standalone device, but should also be able to be commanded by computer.

https://www.facebook.com/creativelab.lu/photos/a.773074186104088/3836887993056010/



At first I tried to port my communication system to Micropython.

That brought me to a set of questions:

- Is there anything like a serial interrupt in Micropython?
- Would I be able to use my command paradigm in this project
- How could I avoid interferences with the programming environment while developing my scripts? Could I use the same serial connection (over USB) for communication and programming or would I have to use a separate one?
- How would I interrupt a running program?
- How would I structure my computer software for the communication?
- Would it be possible to automatically detect on which port my generator was connected?

3. Rethinking the communication

After some attempts to stick to the old paradigm, I realised that it was as always when you miss something in Python: there is a better and more pythonic way to do it!

It was even so easy to communicate that I wonder why I have searched so long to do it the other way.

As an example to do it from Thonny as programming GUI:

I have a script singen_03.py on the file system of my Pico. This has a function set_frequency to set, well, the frequency as it is not difficult to guess.

Once my Pico is connected and available in Thonny, I can do this from the REPL:

```
>>> from singen_03 import *
>>> set_frequency(400)
399.7803

>>>
```

This way, I can set the frequency to any value, from the computer.

I can use all Micropython functions of my module from remote.

I do not have to write new functions to interprete the incoming commands.

Everything is already there and available.

Well, nearly everything.

What if I do not use Thonny, but my own software to control the Pico?

4. Steps to my own Pico control software

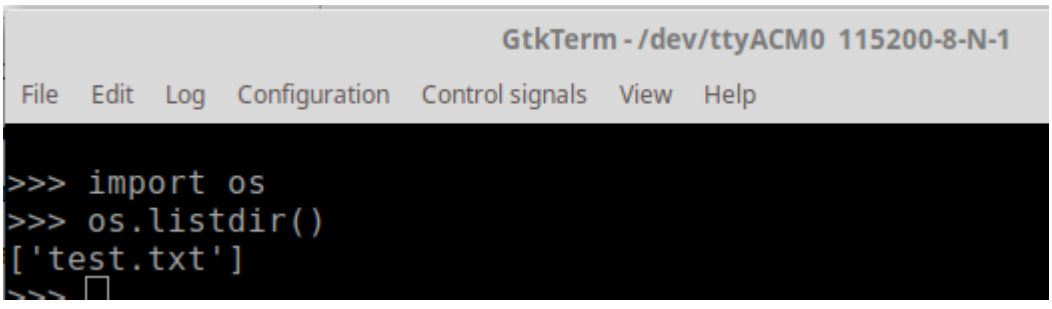
This software should be able to

- connect to the Pico
- interrupt an eventually running program (if necessary)
- send commands (→ this means call functions, as we have seen)
- receive and display messages from the Pico.

A nice to have feature would be to select automatically the right Pico, if there are more Picos and maybe other devices connected.

In principle any serial terminal program can be used instead of Thonny, if it is set to 115200 baud, 8 bits, no parity and connected to the right port.

Here is an example of listing the content of the Pico file system using GtkTerm:



But there may be difficulties if the Pico has a program running, or if it is in the wrong mode.
So it is good to use control codes to set him into the right mode:

Control commands:

```
CTRL-A      -- on a blank line, enter raw REPL mode              b'\x01'
CTRL-B      -- on a blank line, enter normal REPL mode           b'\x02'
CTRL-C      -- interrupt a running program                       b'\x03'
CTRL-D      -- on a blank line, do a soft reset of the board     b'\x04'
CTRL-E      -- on a blank line, enter paste mode                 b'\x05'
```

As I have seen in some programs for communication (ampy for example), it seems to be a good idea to do the CTRL-C command twice.

5. Detecting a connected Pico

In Python, on the computer we can use the serial.tools module to get information about the serial ports and eventually their connectedd hardware:

```
import serial
from serial.tools import list_ports
import time

def print_port_info():
    for port in list_ports.comports():
        print("Device: ", port.device)
        try:
            print ("vid:pid", hex(port.vid), ":", hex(port.pid))
        except:
            print ("vid:pid", port.vid, ":", port.pid)
        print("Serial number: ", port.serial_number)
        print("hiw: ", port.hwid)
        print("name: ", port.name)
        print ("description: ", port.description)
        print ("interface: ", port.interface)
        print("location: ", port.location)
        print("manufacturer", port.manufacturer)
        print("product: ", port.product)
```

From several tests the most characteristic features seem to be:

	ESP8266	Raspi Pico
name	**ttyUSBx**	**ttyACMx**
manufacturer	None	**MicroPython**
interface		Board CDC
description		Board in FS mode - Board CDC

For the Paspi Pico, until refuted, I decided to take manufacturer = “Micropython” as most distinctive feature.

So we can write a function to find all connected Picos:

```
def scan_for_picos(verbose = False):
    '''returns list of USB ports with Raspi Picos connected '''

    picos = []
    for port in list_ports.comports():
        if verbose:
            print("Checking ", port.device)
        if port.manufacturer != None:
            if "MicroPython" in port.manufacturer:
                picos.append(port.device)
    return picos
```

The function returns an array of the serial ports of all connected Picos and can be used like this:

```
picos = scan_for_picos()
print("Picos found:")
for pico in picos:
    print(pico)
```

On my computer this gives:

```
Checking  /dev/ttyS5
Checking  /dev/ttyS4
Checking  /dev/ttyS0
Checking  /dev/ttyUSB0
Checking  /dev/ttyACM0
Checking  /dev/ttyACM1
Picos found:
/dev/ttyACM0
/dev/ttyACM1
```

6. Detecting a special Pico

Once this was done, I felt the need to detect my special Pico with the sinewave program under all the Picos conncted. But how?

From several ideas that could work, I distilled the simplest one:

The Pico should hold a file with just one word of information that would be distinctive.
This file would have to be opened and read locally on the Pico, and the result printed out, that means sent over serial to the computer.

Here is the code to get info about a connected Pico:

```
def get_info_pico(pico, timeout=0.5):
    '''
    Get info stored on the pico in a file info.txt
        This must contain a keyword in the first line:
        The keyword is returned
    '''
    s=serial.Serial(pico, baudrate=115200)
    if s.isOpen()==False:
        s.open()

    # send commands
    s.write(b'\x03\x03')          # Interrupt eventually running program
    s.write(b'\x02')              # Normal REPL
    s.write(b'f = open("info.txt", "r")\r')
    s.write(b't = f.read()\r')
    s.write(b'f.close()\r')
    s.write(b'print(t)\r')

    # Receive answer (with timeout)
    text = ''
    t1 = time.time()
    while True:
        nbbytes = s.inWaiting()
        if nbbytes >0:
            c = s.read(nbbytes)
            c = c.decode("utf-8")
            text += c
        if time.time() - t1 > timeout:
            break

    # Analize answer
    if 'Traceback' in text:                 # Something went wrong
        info = ''
    else:                                   # Filter interesting part
        keyword = 'print(t)'
        pos1 = text.find(keyword)
        info = text[pos1 + len(keyword) +1:]  # keyword found behind print()
        info = info.split()[0]                # take first line only
        info = info.strip()

    return info
```

The function has a few additions to the basic idea that help to get it running flawlessly:

- There is a check if the port is already open, this helps if another program is already using it.
- If a program on the Pico is running in an endless loop, it is interrupted.
- The answer is read with a timeout.
- If there is no file info.txt on the pico there is an error that leads to a reply starting with “Traceback...”. In this case we return an empty string.

- The response is rather verbose, we have to filter out the interesting content.

Once we have this function, it is easy to find the port name of our special pico:

```
def find_pico(keyword, timeout = 0.5):
    '''returns port where a Pico with keyword in first line of
    info.txt is connected'''
    for picoport in scan_for_picos():
        info = get_info_pico(picoport, timeout)
        if info == keyword:
            break
    else:
        picoport = ''
    return picoport
```

So at the start of the computer program, these lines detect the Pico we are looking for:

```
keyword = 'SINGEN'
pico_port = find_pico(keyword, 0.2)
print('Found ', keyword, 'Pico on port',  pico_port)
```

with the output

```
Found  SINGEN Pico on port /dev/ttyACM0
```

7. Example of a command line communication program

As an example, let's have a look at simple program that makes the sine wave generator do a frequency sweep.

First we have to import the needed modules:

```
from scan_picos_01 import scan_for_picos, find_pico
from serial import Serial
import time
```

where scan_picos_01.py holds the functions to scan for picos and to identify a special pico, as described above. The first function of our program finds the pico port and opens it, if it is not already open:

```
def init_pico():
    keyword = 'SINGEN'
    pico_port = find_pico(keyword, 0.05)
    print('Found ', keyword, 'Pico on port',  pico_port)

    ser = Serial(pico_port, baudrate = 115200)
    if ser.open == False:
        ser.open()
    return ser
```

It returns the serial interface for the pico, like ‘/dev/ttyACM0’.

Then we have a function to initialize the generator:

```
def init_singen(ser, timeout = 0.05):
    ser.write(b'\x03\x03')
    get_answer(ser, timeout)
    ser.write(b'from singen_04 import *\r')
    get_answer(ser, timeout)
    time.sleep(0.1)
```

This function interrupts any running program, then commands Micropython to import all functions to control the generator. Thus they are available through the REPL and can be called from remote.

The answer of the controller is caught throuh the get_answer function.

This function captures everything coming from the Pico, until the timeout is over:

```
def get_answer(ser, timeout=0.2, verbose = True):
    t1 = time.time()
    text = ''
    while True:
        nbbytes = ser.inWaiting()
        if nbbytes >0:
            c = ser.read(nbbytes)
            c = c.decode("utf-8")
            text += c
        if time.time() - t1 > timeout:
            break
    if verbose:
        print('RESPONSE: ', text)
    return text
```

The function has two purposes: get information about what the Pico does, and empty the serial receive buffer.

The most important function is the one to set the frequency:

```
def set_frequency(f, timeout = 0.05, verbose = True):
    cmd = b'setf(' + str(f).encode('utf-8') + b')\r'
    if verbose:
        print('CMD: ', cmd)
    ser.write(cmd)
    get_answer(ser, timeout)
```

A main program illustrates how easy it to do everything you want, once these functions are available:

```
if __name__ == "__main__":

    ser = init_pico()
    init_singen(ser)
    time.sleep(0.1)

    for i in range(1,11):
        set_frequency(i*100)
        time.sleep(0.2)

    set_frequency(0)
    ser.close()
```

The program does a frequency sweep 100Hz to 1kHz in steps of 100Hz.

In my example application it may be fine to switch from remote to local once the program on the computer is finished. This can easily be done by resetting the Raspi Pico (For this it is supposed that the generator program on the Pico is started automatically by the main.py file).