

Remote Testing Environment Workshop

OSFC 2018

Piotr Król, Michał Żygowski



- Introduction
- Firmware developer routines
- Remote Testing Environment
- Infrastructure
- Hardware setup
- Automation Tests
- Fixing bugs
- Future plans
 - RTE Control API (beta)
 - New RTE revision for RPi
- Q&A and RTE giveaway



Piotr Król

*Founder & Embedded Systems
Consultant*

-  @pietrushnic
-  piotr.krol@3mdeb.com
-  [linkedin.com/in/krolpiotr](https://www.linkedin.com/in/krolpiotr)
-  [facebook.com/piotr.krol.756859](https://www.facebook.com/piotr.krol.756859)



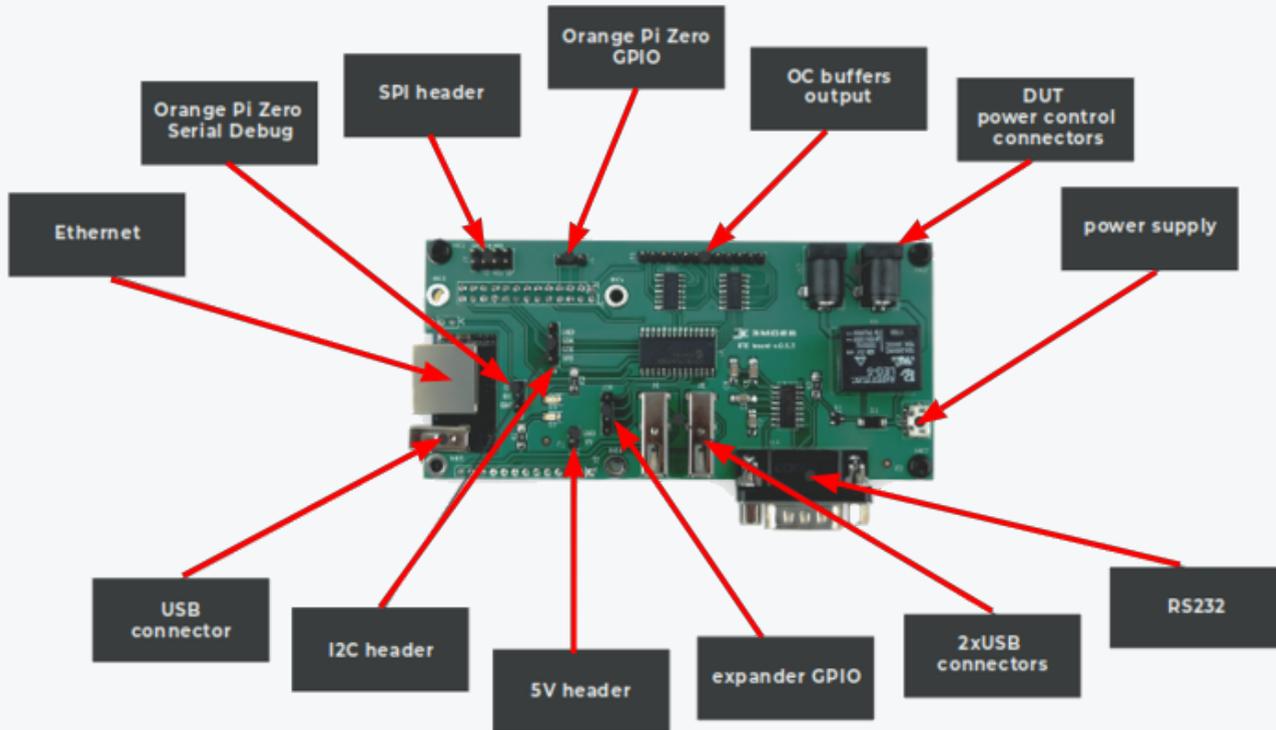
Michał Żygowski

Firmware Engineer

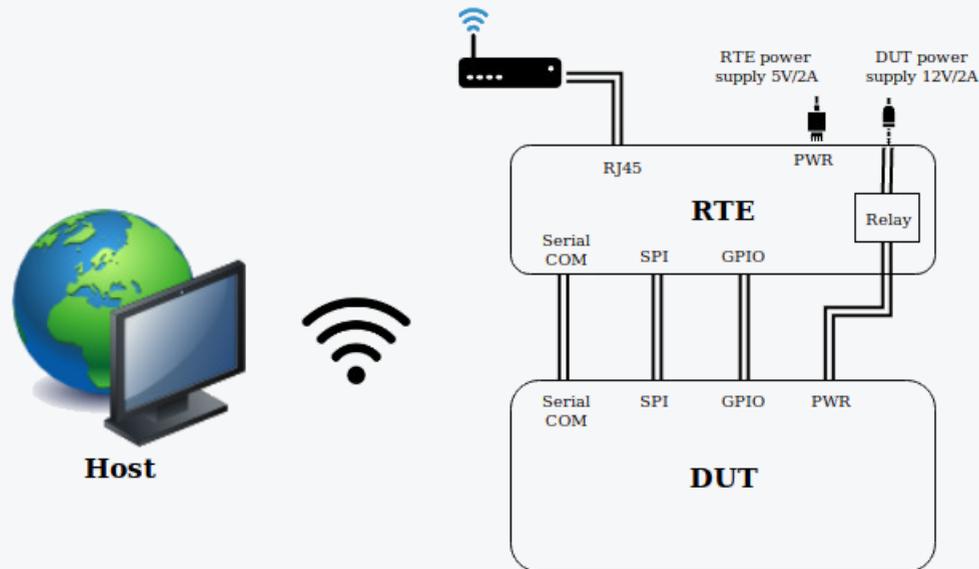
-  michal.zygowski@3mdeb.com
-  [linkedin.com/in/michał-żygowski-88954416b](https://www.linkedin.com/in/michał-żygowski-88954416b)

- Narrowing down serious issue requires a lot of repetitive work:
 - modify & build firmware with potential fix
 - flash affected firmware version
 - boot system
 - verify if the bug still exists
- Problem with automation and remote development

To get rid of these problems we created Remote Testing Environment.

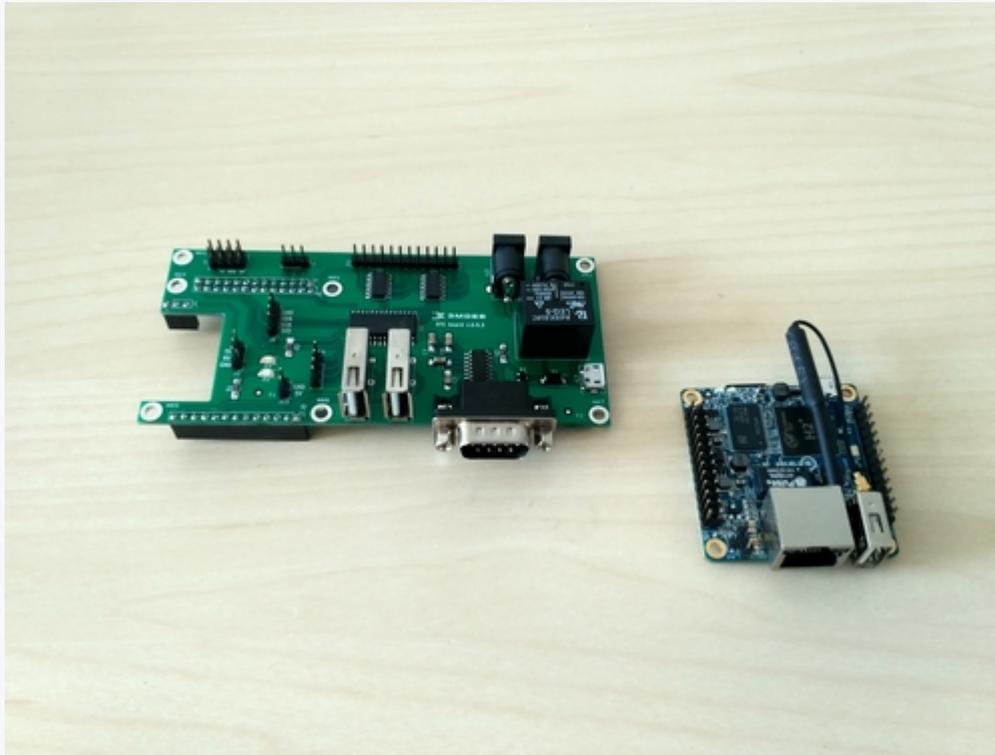


Today, our Devices Under Test are PC Engines APU platforms. For workshop purposes we will need:



Additional infrastructure that can be used is Debian machine with iPXE server for supplementary boot option.

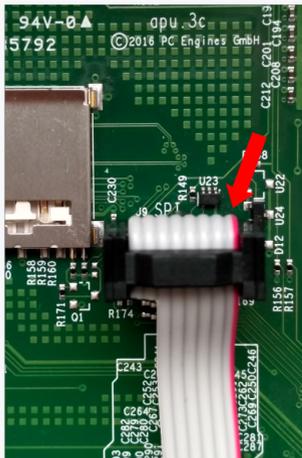
- connect RTE Hat with Orange Pi Zero board (~3min)



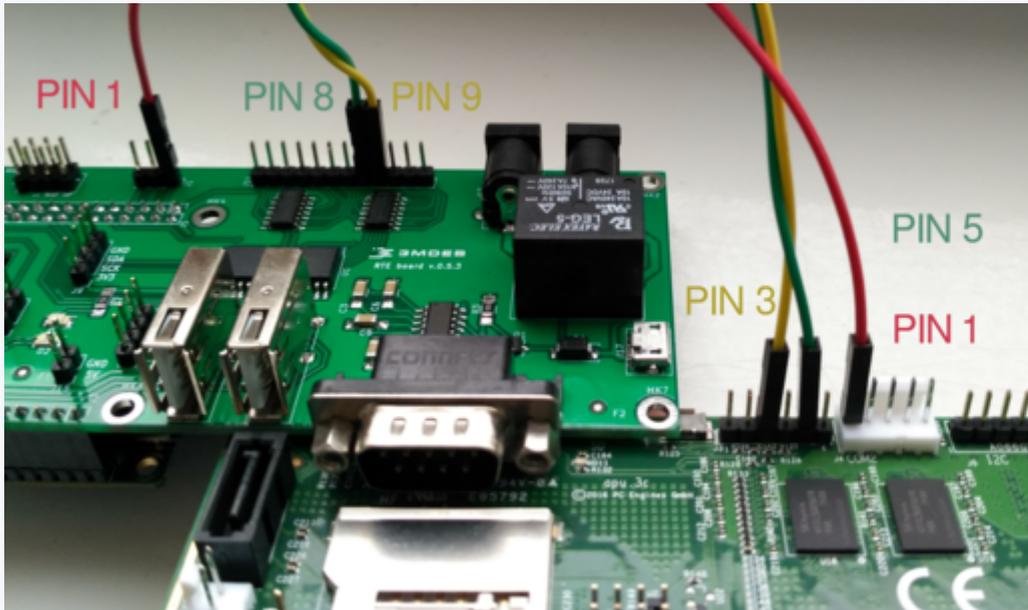
- RS232 interface via a null modem cable (~2min)



- SPI interface via IDC 2x4 pin cable (~3min)



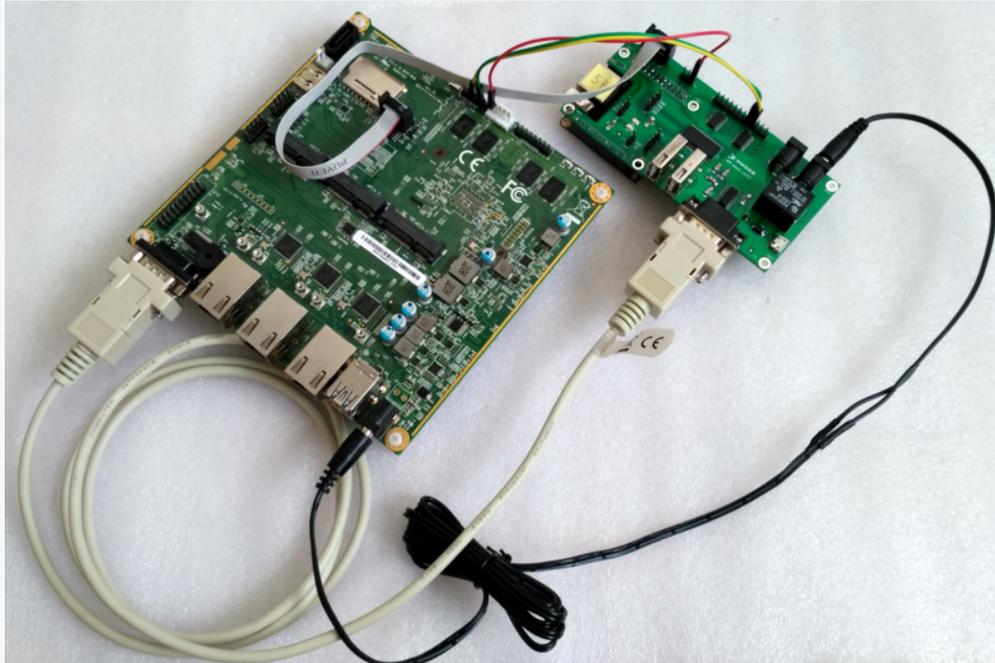
- APU PWR/RST pins with RTE OC buffers (~5min)



RTE header J11 pin	APU2 header J2 pin
9 (OC buffer output)	3 (PWR)
8 (OC buffer output)	5 (RST)
RTE header J11 pin	APU3/4 header J3 pin
9 (OC buffer output)	3 (PWR)
8 (OC buffer output)	5 (RST)
RTE header J1 pin	APU2 header J3 pin
1 (Orange Pi GPIO)	1 (V3)
RTE header J1 pin	APU3/4 header J4 pin
1 (Orange Pi GPIO)	1 (V3)

- DC-DC jack cable (~2min)





- take the connected setup to the power/network section (~10min):
 - power up RTE and APU platforms
 - connect Ethernet cable to Orange Pi Zero

- Generic test automation framework for acceptance test-driven development (ATDD)
- Utilizes the keyword-driven testing approach
- The core is implemented using Python and runs also on Jython (JVM) and IronPython (.NET)
- Bug fixes need tests too - Test-Driven Bug Fixing (TDBF)



Each test suite should contain 4 sections:

```
*** Settings ***
```

```
Library          SSHLibrary
Resource         robotframework/variables.robot
Suite Setup      SSH Connection and Log In    ${rte_ip}
Suite Teardown   Log Out And Close Connections
```

```
*** Variables ***
```

```
${username}      root
${password}      meta-rte
```

```
*** Keywords ***
```

```
Log Out And Close Connections
[Documentation]    Close all telnet and ssh open connections.
Telnet.Close All Connections
SSHLibrary.Close All Connections
```

```
*** Test Cases ***
```

```
RTE: 1.1 SSH connection
    ${ssh_info}=    SSHLibrary.Get Connection
    Should Be Equal As Strings    ${ssh_info.host}    ${rte_ip}
```

First, prepare the environment (~10min):

First, prepare the environment (~10min):

- start Debian RTE Workshop VBox and login
 - password: **rte-workshop**

First, prepare the environment (~10min):

- start Debian RTE Workshop VBox and login
 - password: **rte-workshop**
- open terminal (Activities/Terminal)

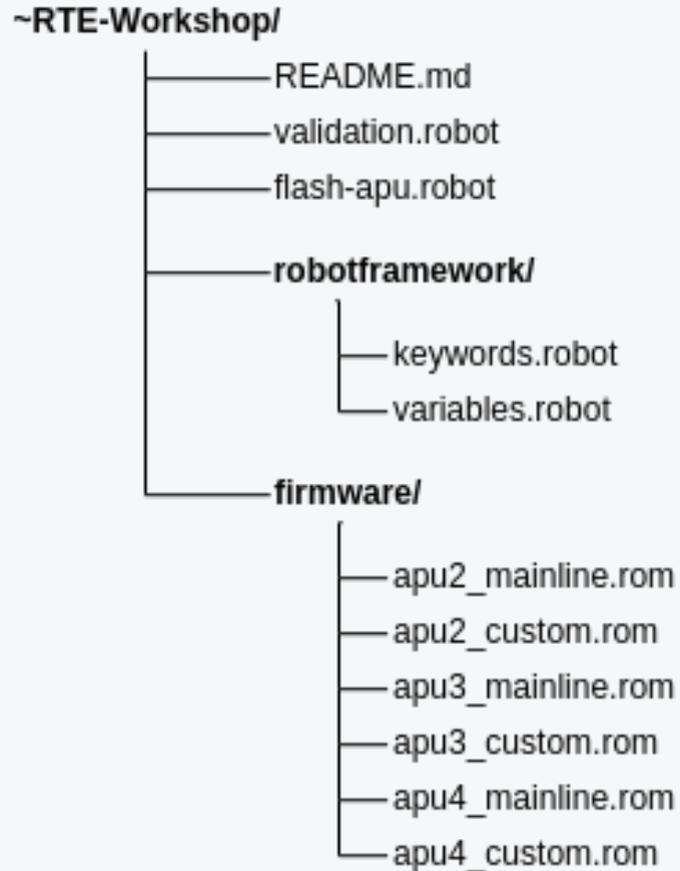
First, prepare the environment (~10min):

- start Debian RTE Workshop VBox and login
 - password: **rte-workshop**
- open terminal (Activities/Terminal)
- change directory: **cd RTE-Workshop**

First, prepare the environment (~10min):

- start Debian RTE Workshop VBox and login
 - password: **rte-workshop**
- open terminal (Activities/Terminal)
- change directory: **cd RTE-Workshop**
- activate virtualenv: **source robot-venv/bin/activate**

```
rte-workshop@debian:~$ cd RTE-Workshop/  
rte-workshop@debian:~/RTE-Workshop$ source robot-venv/bin/activate  
(robot-venv) rte-workshop@debian:~/RTE-Workshop$ █
```



Now we should run script which tests our hardware setup (~5min):

Now we should run script which tests our hardware setup (~5min):

- `robot -v rte_ip:RTE_IP validation.robot`

```
~/RTE-Workshop$ robot -v rte_ip:192.168.3.50 validation.robot
```

where **RTE_IP** is your RTE IP address

Now we should run script which tests our hardware setup (~5min):

- `robot -v rte_ip:RTE_IP validation.robot`

```
~/RTE-Workshop$ robot -v rte_ip:192.168.3.50 validation.robot
```

where **RTE_IP** is your RTE IP address

```
=====
Validation
=====
RTE: 1.1 SSH connection | PASS |
-----
RTE: 2.1 Relay - power on Device Under Test | PASS |
-----
RTE: 3.1 Serial connection | PASS |
-----
```

RTE: 4.1 PWR pin - power off and on Device Under Test

Power Off - platform shutdown

...

Power On - get platform sign of life

....

Output:

```
PC Engines apu4  
coreboot build 20180708  
BIOS version v4.8.0.3  
4080 MB ECC DRAM
```

RTE: 4.1 PWR pin - power off and on Device Under Test

| PASS |

```
-----  
RTE: 4.2 RST pin - reset Device Under Test .
```

```
    Reset - platform reboot
```

```
..
```

```
    Reset - get platform sign of life
```

```
..
```

```
Output:
```

```
PC Engines apu4  
coreboot build 20180708  
BIOS version v4.8.0.3  
4080 MB ECC DRAM
```

```
RTE: 4.2 RST pin - reset Device Under Test
```

```
| PASS |
```

- Possible failures:
 - **ssh timeout** - RTE lacks network connection
 - **NoValidConnections** - bad IP address
 - **Telnet timeout** - check RS232 cable or close established connection

- Possible failures:
 - **ssh timeout** - RTE lacks network connection
 - **NoValidConnections** - bad IP address
 - **Telnet timeout** - check RS232 cable or close established connection
- After each RobotFramework test suite we can track generated log files:

```
-----  
Validation | PASS |  
5 critical tests, 5 passed, 0 failed  
5 tests total, 5 passed, 0 failed  
=====
```

Output: /home/rte-workshop/RTE-Workshop/output.xml
Log: /home/rte-workshop/RTE-Workshop/log.html
Report: /home/rte-workshop/RTE-Workshop/report.html

Try it by yourself by typing: **firefox log.html report.html output.xml**

Flashing process can be started by `flash-apu.robot` script (~5min):

- `robot -v rte_ip:RTE_IP -v fw_file:PATH_TO_FILE flash-apu.robot`
- remember to change `fw_file` firmware path for appropriate APU2/3/4 binary (if you are not sure, check the sign of life from previous tests)

```
(robot-venv) rte-workshop@debian:~/RTE-Workshop$ robot -v rte_ip:192.168.3.50  
-v fw_file:firmware/apu4_mainline.rom flash-apu.robot
```

Flashing process can be started by `flash-apu.robot` script (~5min):

- `robot -v rte_ip:RTE_IP -v fw_file:PATH_TO_FILE flash-apu.robot`
- remember to change `fw_file` firmware path for appropriate APU2/3/4 binary (if you are not sure, check the sign of life from previous tests)

```
(robot-venv) rte-workshop@debian:~/RTE-Workshop$ robot -v rte_ip:192.168.3.50  
-v fw_file:firmware/apu4_mainline.rom flash-apu.robot
```

Possible failures:

- `opening "/tmp/coreboot.rom" failed` - SSH key not authenticated
- `No such file or directory` - bad firmware file path

We have prepared 2 binaries to present the RTE and RobotFramework capability to track down unexpected bugs and flash APUs firmware:

We have prepared 2 binaries to present the RTE and RobotFramework capability to track down unexpected bugs and flash APUs firmware:

I. First, let's try to update our DUT with custom firmware:

We have prepared 2 binaries to present the RTE and RobotFramework capability to track down unexpected bugs and flash APUs firmware:

I. First, let's try to update our DUT with custom firmware:

- `run:robot -v rte_ip:RTE_IP -v fw_file:firmware/apu4_custom.rom flash-apu.robot`

We have prepared 2 binaries to present the RTE and RobotFramework capability to track down unexpected bugs and flash APUs firmware:

I. First, let's try to update our DUT with custom firmware:

- `run:robot -v rte_ip:RTE_IP -v fw_file:firmware/apu4_custom.rom flash-apu.robot`

```
=====
RTE: Flash and validate APUx firmware

    Flashing procedure has started
..
    Flashing procedure has finished
.....
    Sign of Life output:

PC Engines apu4
coreboot build 20180708
BIOS version v0.0.0.0
4080 MB ECC DRAM

RTE: Flash and validate APUx firmware | FAIL |
BIOS version is not correct!
-----
```

We have prepared 2 binaries to present the RTE and RobotFramework capability to track down unexpected bugs and flash APUs firmware:

II. Assumes that we made appropriate changes and build the new binary, flash APU with the newest v4.8.0.3 firmware:

We have prepared 2 binaries to present the RTE and RobotFramework capability to track down unexpected bugs and flash APUs firmware:

II. Assumes that we made appropriate changes and build the new binary, flash APU with the newest v4.8.0.3 firmware:

- `run:robot -v rte_ip:RTE_IP -v fw_file:firmware/apu4_mainline.rom
flash-apu.robot`

We have prepared 2 binaries to present the RTE and RobotFramework capability to track down unexpected bugs and flash APUs firmware:

II. Assumes that we made appropriate changes and build the new binary, flash APU with the newest v4.8.0.3 firmware:

- `run:robot -v rte_ip:RTE_IP -v fw_file:firmware/apu4_mainline.com flash-apu.robot`

```
=====
RTE: Flash and validate APUx firmware

    Flashing procedure has started
..
    Flashing procedure has finished
.....
    Sign of Life output:

PC Engines apu4
coreboot build 20180708
BIOS version v4.8.0.3
4080 MB ECC DRAM

RTE: Flash and validate APUx firmware | PASS |
=====
```

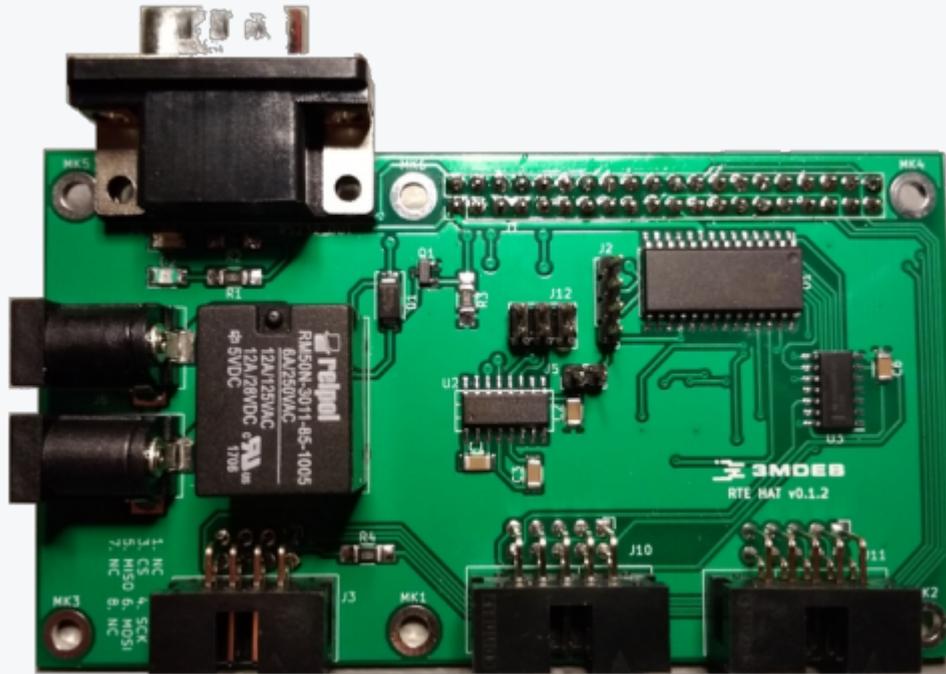
- To open rest API service, type: `firefox RTE_IP:8000` (~3min)

Remote Testing Environment control

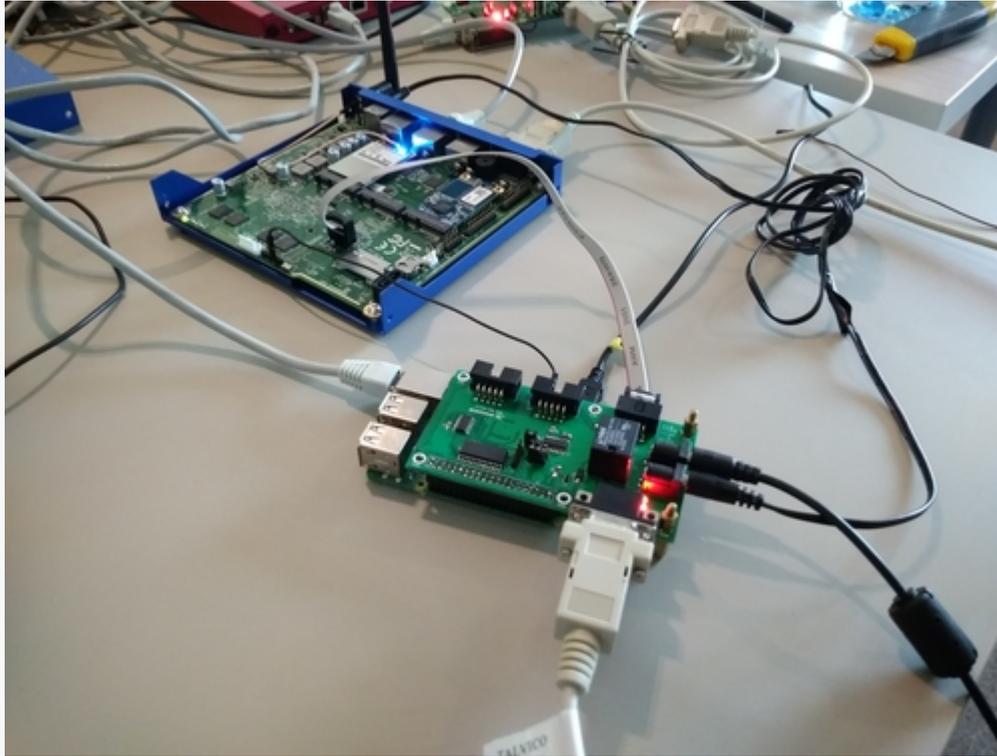
PWR RELAY

File to flash: No file selected.

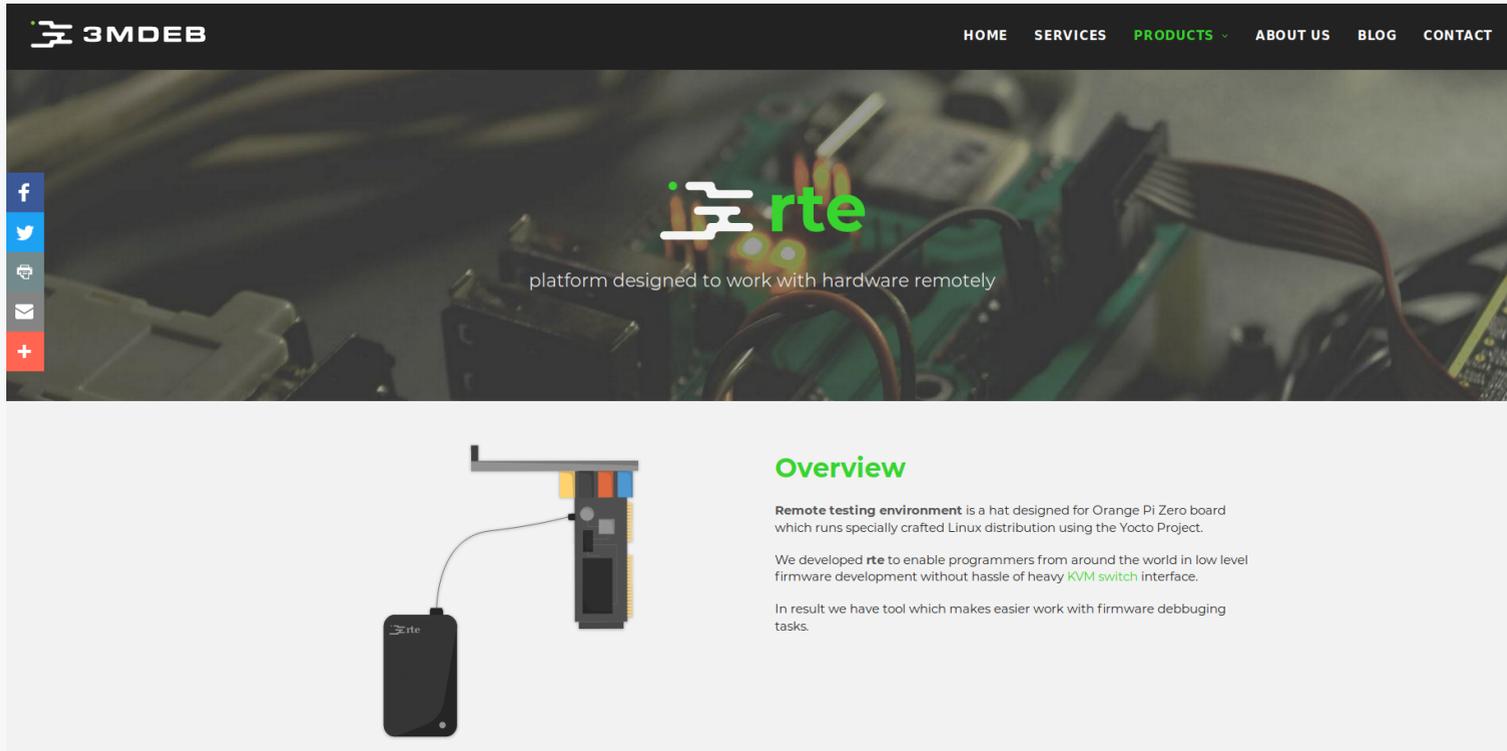
no flashing operation performed



New RTE Hat compatible with 40-pin header RaspberryPi 2/3/Zero



New revision during testing phase



The screenshot shows the RTE website homepage. At the top left is the 3MDEB logo. The top right navigation menu includes links for HOME, SERVICES, PRODUCTS (with a dropdown arrow), ABOUT US, BLOG, and CONTACT. The main header features a background image of a Raspberry Pi board with the RTE logo and the tagline "platform designed to work with hardware remotely". On the left side of the header, there is a vertical stack of social media icons for Facebook, Twitter, GitHub, Email, and a plus sign for more options. Below the header, the "Overview" section is displayed. It includes an image of the RTE hardware, which is a small black device connected to an Orange Pi Zero board. The text in the Overview section describes the "Remote testing environment" as a hat for the Orange Pi Zero board, running a custom Linux distribution based on the Yocto Project. It explains that RTE was developed to facilitate low-level firmware development without the need for a KVM switch interface, making firmware debugging easier.

Overview

Remote testing environment is a hat designed for Orange Pi Zero board which runs specially crafted Linux distribution using the Yocto Project.

We developed **rte** to enable programmers from around the world in low level firmware development without hassle of heavy **KVM switch** interface.

In result we have tool which makes easier work with firmware debbuging tasks.

Q&A and lottery

- Cards received at the entrance will serve as numbers to select the winner of the lottery. **The prizes are three full RTE sets!**
- **Where can I order the RTE for myself?**
The online store is starting soon, so please submit any orders to:

 contact@3mdeb.com

Please visit our RTE website: 3mdeb.com/rte