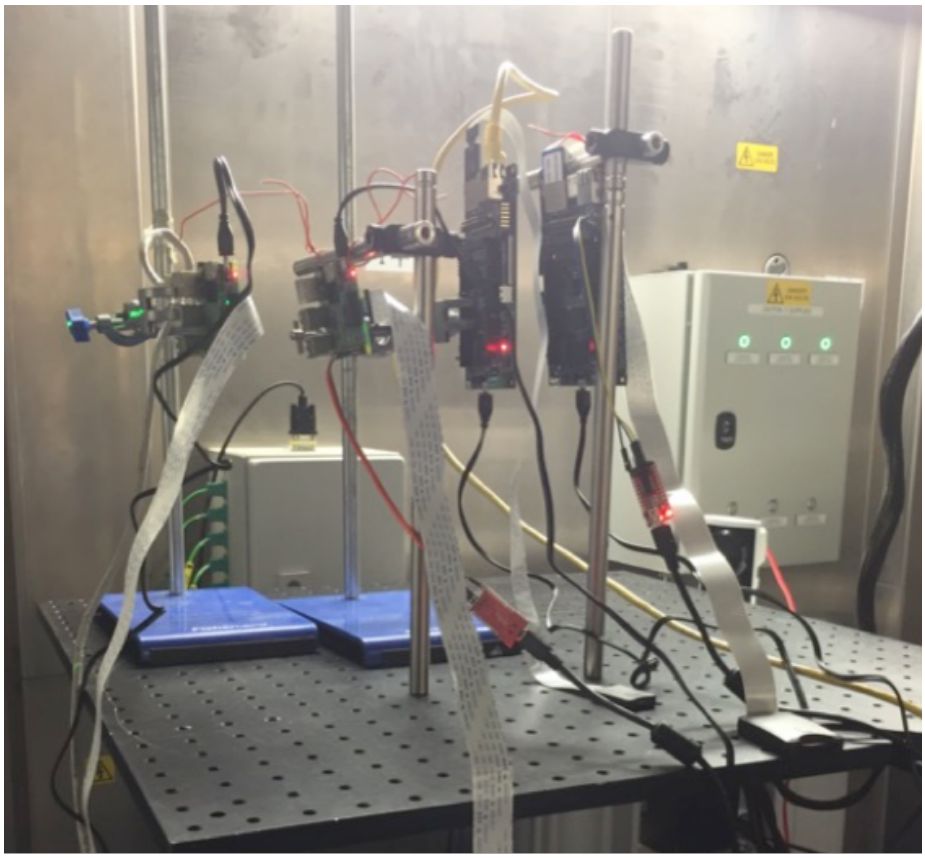
Setup Description and Documentation

Pablo R. Bodmann

August 21, 2021



Contents

1 Setup Overview

Figure 1 shows an abstract view of this setup. We have on the side under radiation the DUT which runs the chosen benchmarks and outside the radiation we have a host computer that starts the application on the DUT, receives and logs them.

The messages can be either the end of a successful execution of an erroneous one. The lack of messages means that either the application has stopped executing or the Linux has crashed.

When there is a System Crash, the host sends a request to the power switch. The power switch turns off and turns the DUT back on in order to force a reboot.

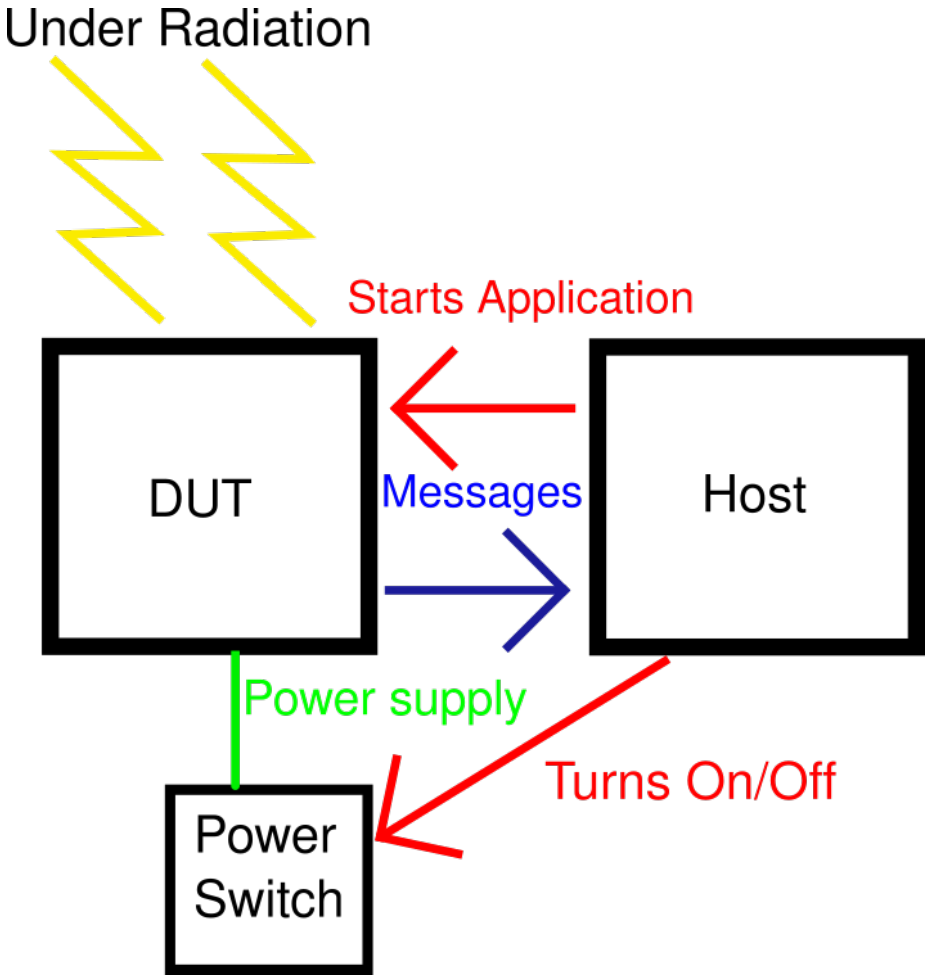


Figure 1: Abstract View of the Setup

2 Software Needed

This setup on works on Linux both on the Host machine and on the DUT

2.1 Host

- python3
- Curl
- Telnet (Client)

2.2 DUT

- Telnet (Server)

3 Controlling Script

This setup uses two scripts, one being the controlling script (script.py) and the other one controlling the power switch (switch.py). The control of the power switch is made by sending a specific HTTP message. Currently, the only two supported power switches are the Lindy IP power switch and Aviosys IP power switch.

The controlling script receives as arguments in this order: DUT IP, Message receiving port, the benchmark name, switch type (lindy, ip_power), switch IP, power switch port where the DUT is connected, sleep time (for waiting for the DUT to boot), DUT username and DUT password.

When adding new benchmarks, remember to add them to the arguments dictionary (arg_dic), to use the IP and port for messaging and the error message size (mess_size_dic), for more information see Section 5.

4 Types of Messages and Timeout

As seen previously in Figure 1, the DUT when running sends periodically messages to the host machine. There are 3 types of messages:

- AA: benchmark terminated correctly
- DD: benchmark terminated with an incorrect output
- CC: comes after the DD if more parts of the output are wrong, i.e., multiples elements on a matrix are wrong. See Section 6 for an example with all types of message messages. This is interpreted as being in the same erroneous run as the first DD message.

The reason for two types of messages for an erroneous run of the benchmark is that the controlling script does not know how many parts of the output were corrupted. Therefore, packaging each incorrect part in a message with a constant size is more preferable and easier to process.

Besides logging the wrong outputs, the controlling script also logs crashes. This is made by waiting for messages and after a time, much larger than the execution of the benchmark, the script attempts to restart the benchmark. If it manages successfully contact the DUT, this is logged as an Application Crash, if not as a System Crash.

5 Benchmark Overall Structure

Algorithm 1: Overall structure of the benchmarks

```
Input [Host IP] [Host Port] <other benchmark arguments>
initialization;
while True do
    A:
    benchmark();
    B:
    test_output();
    if output is correct then
        send AA message;
    else
        send DD and CC messages;
    C:
```

Listing 1 shows the overall structure of a benchmark. All benchmarks receive two arguments besides the ones specific to them. These two are the host IP and port which are used by the controlling script to receive the messages. If there is more than one DUT running, the host may be the same however the port must be different in order not to mix the messages. After these two arguments, come the other arguments specific to the benchmark. For example, the Matrix Multiplication benchmark receives Host IP, Host Port, Input File with the input Matrices, Correct Output File (Template), and Matrix Line Size (The matrices are all square).

Also, it can be noted in Listing 1 the points A, B and C. These are marks not present in the test benchmark but are timing marks needed after in order to calculate the error rate. The timings needed are A-B (benchmark running time) and A-C (benchmark total time).

6 Log Structure

28-05-2021_21-39-30 192.168.1.60 [INFO] starting radiation experiment...
28-05-2021_21-39-30 192.168.1.60 [INFO] BOARD HARD RESET...
28-05-2021_21-40-00 192.168.1.60 [INFO] EXPERIMENT RESUMED

28-05-2021_21-40-01 192.168.1.60 [PL] AA
28-05-2021_21-40-02 192.168.1.60 [PL] DD 000001da0000017e4ca7a64e
28-05-2021_21-40-02 192.168.1.60 [PL] CC 000001db000000664ca72467
28-05-2021_21-40-03 192.168.1.60 [PL] AA
28-05-2021_21-40-33 192.168.1.60 [INFO] TIMEOUT... APP CRASHED
28-05-2021_21-40-50 192.168.1.60 [INFO] EXPERIMENT RESUMED
28-05-2021_21-41-51 192.168.1.60 [PL] AA
28-05-2021_21-42-21 192.168.1.60 [INFO] TIMEOUT... LINUX CRASHED
28-05-2021_21-40-38 192.168.1.60 [INFO] EXPERIMENT RESUMED
28-05-2021_21-40-39 192.168.1.60 [PL] AA

Here is an example of a log for the Matrix multiplication benchmark. On all lines, we can observe the time which the host logged the message. All the lines with [INFO] are generated by the script itself. The ones with [PL] are messages received from the DUT. We can see the AA messages indication a correct execution of the benchmark. In the case of this benchmark, we observed an erroneous execution with two elements wrong. The first one is on the position 0x1da,0x17e with its value being 0x4ca7a64e. The second element wrong is on 0x1db,0x66 and has a value of 0x4ca72467. All messages sent by the DUT are raw bytes, because of speed and message size.

Also, we can observe an Application Crash and a System Crash. The Application Crash is faster to recover than a System Crash because in the latter there is the need to wait for the DUT to boot.