Antmicro

Protoplaster - docs example

2025-01-08

CONTENTS

CHAPTER ONE

INTRODUCTION

This documentation serves as an example of how individual projects documentation can look like.

The second chapter contains information from the README file.

The last chapter is generated from the sample `test.yml` file which can be found in the README. Its purpose is to demonstrate the documentation generated to describe test procedures used in a project.

CHAPTER TWO

PROTOPLASTER

Copyright (c) 2022-2024 Antmicro

An automated framework for platform testing (Hardware and BSPs).

Currently includes tests for:

- I2C
- GPIO
- Camera
- FPGA

2.1 Installation

```
pip install git+https://github.com/antmicro/protoplaster.git
```

2.2 Usage

```
usage: protoplaster [-h] [-t TEST_FILE] [-g GROUP] [--list-groups] [-o OUTPUT] [--
↪csv CSV] [--csv-columns CSV_COLUMNS] [--generate-docs] [-c CUSTOM_TESTS]

options:
  -h, --help            show this help message and exit
  -t TEST_FILE, --test-file TEST_FILE
                        Path to the test yaml description
  -g GROUP, --group GROUP
                        Group to execute
  --list-groups         List possible groups to execute
  -o OUTPUT, --output OUTPUT
                        A junit-xml style report of the tests results
  --csv CSV             Generate a CSV report of the tests results
  --csv-columns CSV_COLUMNS
                        Comma-separated list of columns to be included in␣
↪generated CSV
  --generate-docs       Generate documentation
  -c CUSTOM_TESTS, --custom-tests CUSTOM_TESTS
```

(continues on next page)

(continued from previous page)

```
                        Path to the custom tests sources
  --report-output REPORT_OUTPUT
                        Proplaster report archive
  --system-report-config SYSTEM_REPORT_CONFIG
                        Path to the system report yaml config file
  --sudo                Run as sudo
```

Protoplaster expects a yaml file describing tests as an input. The yaml file should have a structure specified as follows:

```yaml
base:                     # A group specifier
  i2c:                    # A module specifier
  - bus: 0                # An interface specifier
    devices:              # Multiple instances of devices can be defined in one module
    - name: "Sensor name"
      address: 0x3c  # The given device parameters determine which tests will be␣
↪run for the module
  - bus: 0
    devices:
    - name: "I2C-bus multiplexer"
      address: 0x70
  camera:
  - device: "/dev/video0"
    camera_name: "vivid"
    driver_name: "vivid"
  - device: "/dev/video2"
    camera_name: "vivid"
    driver_name: "vivid"
    save_file: "frame.raw"
additional:
  gpio:
  - number: 20
    value: 1
```

2.2.1 Groups

In the YAML file, you can define different groups of tests to run them for different use cases. In the YAML file example, there are two groups defined: base and additional. Protoplaster, when run without a defined group, will execute every test in each group. When the group is specified with the parameter `-g` or `--group`, only the tests in the specified group are going to be run. You can also list existing groups in the YAML file, simply run `protoplaster --list-groups test.yaml`.

2.3 Base modules parameters

Each base module requires parameters for test initialization. These parameters describe the tests and are passed to the test class as its attributes.

2.3.1 I2C

Required parameters:

- `bus` - i2c bus to be checked
- `name` - name of device to be detected
- `address` - address of the device to be detected on the indicated bus

2.3.2 GPIO

Required parameters:

- `number` - GPIO pin number
- `value` - value written to that pin

Optional parameters:

- `gpio_name` - name of the sysfs GPIO interface after exporting

2.3.3 Cameras

Required parameters:

- `device` - path to the camera device (eg. /dev/video0)
- `camera_name` - expected camera name
- `driver_name` - expected driver name

Optional parameters:

- `save_file` - a path which the tested frame is saved to (the frame is saved only if this parameter is present)

2.3.4 FPGA

Required parameters:

- `sysfs_interface` - path to a sysfs interface for flashing the bitstream to the FPGA
- `bitstream_path` - path to a test bitstream that is going to be flashed

2.4 Writing additional modules

Apart from base modules available in Protoplaster, you can provide your own extended modules. The module should contain a `test.py` file in the root path. This file should contain a test class that is decorated with `ModuleName("")` from the `protoplaster.conf.module` package. This decorator tells Protoplaster what the name of the module is. With this information, Protoplaster can correctly initialize the test parameters. The test class should contain a `name()` method. Its return value is used for the `device_name` field in CSV output.

The description of the external module should be added to the YAML file as for other tests. By default, external modules are expected in the `/etc/protoplaster` directory. If you want to store them in a different path, use the `--custom-tests` argument to set your own path. Individual tests run by Protoplaster should be present in the main class in the `test.py` file. The class's name should start with `Test`, and every test's name in this class should also start with `test`. An example of an extended module test:

```
from protoplaster.conf.module import ModuleName

@ModuleName("additional_camera")
class TestAdditionalCamera:
    """
    {% macro TestAdditionalCamera(prefix) -%}
    Additional camera tests
    -----------------------
    {% do prefix.append('') %}
    This module provides tests dedicated to camera sensors on specific video node:
    {%- endmacro %}
    """

    def test_exists(self):
        """
        {% macro test_exists(device) -%}
          check if the path exists
        {%- endmacro %}
        """
        assert self.path == "/dev/video0"
```

And a YAML definition:

```
---
base:
  additional_camera:
    - path: "/dev/video0"
    - path: "/dev/video1"
```

2.5 Protoplaster test report

Protoplaster provides protoplaster-test-report, a tool to convert test CSV output into a HTML or Markdown table.

```
usage: protoplaster-test-report [-h] [-i INPUT_FILE] -t {md,html} [-o OUTPUT_FILE]

options:
  -h, --help            show this help message and exit
  -i INPUT_FILE, --input-file INPUT_FILE
                        Path to the csv file
  -t {md,html}, --type {md,html}
                        Output type
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        Path to the output file
```

2.6 System report

Protoplaster provides protoplaster-system-report, a tool for obtaining information about system state and configuration. It executes a list of commands and saves their outputs. The outputs are stored in a single zip archive along with an HTML summary.

2.6.1 Usage

```
usage: protoplaster-system-report [-h] [-o OUTPUT_FILE] [-c CONFIG] [--sudo]

options:
  -h, --help            show this help message and exit
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        Path to the output file
  -c CONFIG, --config CONFIG
                        Path to the YAML config file
  --sudo                Run as sudo
```

The YAML config contains a list of actions to perform. A single action is described as follows:

```yaml
report_item_name:
  run: script
  summary:
    - title: summary_title
      run: summary_script
  output: script_output_file
  superuser: required | preferred
  on-fail: ...
```

- run - command to be run
- summary – a list of summary generators, each one with fields:
  - title – summary title

  - run – command that generates the summary. This command gets the output of the original command as stdin. This field is optional; if not specified, the output is placed in the report as-is.

- output - output file for the output of run.
- superuser – optional, should be specified if the command requires elevated privileges to run. Possible values:
  - required – protoplaster-system-report will terminate if the privilege requirement is not met
  - preferred – if the privilege requirement is not met, a warning will be issued and this particular item won't be included in the report
- on-fail – optional description of an item to run in case of failure. It can be used to run an alternative command when the original one fails or is not available.

Example config file:

```
uname:
  run: uname -a
  summary:
    - title: os info
      run: cat
  output: uname.out
dmesg:
  run: dmesg
  summary:
    - title: usb
      run: grep usb
    - title: v4l
      run: grep v4l
  output: dmesg.out
  superuser: required
ip:
  run: ip a
  summary:
    - title: Network interfaces state
      run: python3 $PROTOPLASTER_SCRIPTS/generate_ip_table.py "$(cat)"
  output: ip.out
  on-fail:
    run: ifconfig -a
    summary:
      - title: Network interfaces state
        run: python3 $PROTOPLASTER_SCRIPTS/generate_ifconfig_table.py "$(cat)"
    output: ifconfig.out
```

2.6.2 Running as root

By default, `sudo` doesn't preserve `PATH`. To run `protoplaster-system-report` installed by a non-root user as root, invoke `protoplaster-system-report --sudo`

CHAPTER

THREE

PROTOPLASTER TESTS

Note: This page has been autogenerated from a Protoplaster tests definition file.

To perform hardware/BSP tests and open-source Protoplaster framework has been used.

Running Protoplaster runs the tests described in the following chapters:

3.1 I2C devices tests

This module provides tests dedicated to i2c devices on specific buses:

- /dev/i2c-0:
  - detection test for *Sensor name* on address: `0x3c`
- /dev/i2c-0:
  - detection test for *I2C-bus multiplexer* on address: `0x70`

3.2 Camera sensor tests

This module provides tests dedicated to V4L devices on specific video node:

- /dev/video0:
  - try to capture frame
  - check if the camera sensor name is `vivid`
  - check if the camera sensor driver name is `vivid`
- /dev/video2:
  - try to capture frameand store it to `frame.raw` file
  - check if the camera sensor name is `vivid`
  - check if the camera sensor driver name is `vivid`

3.3 GPIOs tests

This module provides tests dedicated to GPIO on specific pin number

- /sys/class/gpio/gpio20:
  - write 1 and read back to confirm

CHAPTER

FOUR

PROTOPLASTER TESTS REPORT

device name	test name	module	duration	message	status
/dev/i2c-0	addresses	test.py::TestI2C::test_addresses	8ms 383us		passed
/dev/i2c-0	addresses	test.py::TestI2C::test_addresses	7ms 604us		passed
/dev/video0	frame	test.py::TestCamera::test_frame	256ms 241us		passed
/dev/video0	device_name	test.py::TestCamera::test_device	220ms 367us		passed
/dev/video0	driver_nam	test.py::TestCamera::test_driver_	221ms 848us		passed
/dev/video2	frame	test.py::TestCamera::test_frame	241ms 463us		passed
/dev/video2	device_name	test.py::TestCamera::test_device	205ms 827us		passed
/dev/video2	driver_nam	test.py::TestCamera::test_driver_	204ms 421us		passed
/sys/class/gpio/	read_write	test.py::TestGPIO::test_read_wri	12ms 357us		passed

CHAPTER

FIVE

PROTOPLASTER SYSTEM REPORT

Protoplaster provides `protoplaster-system-report`, a tool to obtain information about system state and configuration. It executes a list of commands and saves their outputs. The outputs are stored in a single zip archive together with an HTML summary. An example summary can be found here.

The following config was used to generate the example:

```
uname:
  run: uname -a
  summary:
    - title: os info
      run: cat
  output: uname.out
dmesg:
  run: dmesg
  summary:
    - title: usb
      run: grep usb
    - title: v4l
      run: grep v4l
  output: dmesg.out
  superuser: required
ip:
  run: ip a
  summary:
    - title: Network interfaces state
      run: python3 $PROTOPLASTER_SCRIPTS/generate_ip_table.py "$(cat)"
  output: ip.out
  on-fail:
    run: ifconfig -a
    summary:
      - title: Network interfaces state
        run: python3 $PROTOPLASTER_SCRIPTS/generate_ifconfig_table.py "$(cat)"
    output: ifconfig.out
```