Arm Automotive Solutions

Version v2.2

Arm Ltd.

May 14, 2026

CONTENTS

Note

For Arm Zena CSS v2.2 and later documentation, see arm-zena-css.docs.arm.com.

CHAPTER

ONE

TEST AUTOMATION

1.1 User Guide

Our Test Automation uses pytest for extensive plugin support meeting industry level standard and configurable YAML based system for platform setup.

It supports multiple execution targets:

- FVP (Fast Models)
- FPGA (Remote hardware via SSH)

1.1.1 Features

- Telnet-based multi-terminal session handling for Arm FVP integration (extendable to FPGA and SoC environments).
- Pytest-based test execution with modular plugin extensions.
- Log-based result tracking and verification, including ANSI/log filtering utilities.
- Reusable command execution utilities with prompt detection and automatic login handling.
- Platform-based configuration system using properties and YAML templates.

1.1.2 Getting Started

This section provides a detailed, step-by-step guide for setting up an automated test environment for Arm Automotive Software Reference Stack.

Each run creates a parent folder inside the central `logs` directory. The folder name follows this format: `logs/<target>_<platform>_<timestamp>/`

Inside that folder, normalized log files are generated depending on the target type.

For **FVP** platforms, the following files are created:

- **Boot log** - e.g. `<platform>_<timestamp>.log`
- **Telnet console logs** - e.g. `telnet_<console name>_<port number>.log` (one per console/port)
- **Command outputs** - e.g. `cmd_output.txt` for captured command results

For **FPGA** platforms, the following files are created:

- **Boot log** - e.g. `<platform>_boot_<timestamp>.log` Captures stdout and stderr of the remote boot command.

- **Remote run logs directory** - i.e. `remote_run_logs` which contains the downloaded contents of the remote `RUN_DIR`. This includes UART log files and any additional artifacts generated during FPGA boot.

This consistent layout provides a clear way to inspect results, share logs, and debug issues.

1.1.3 Installation

Ensure Python **3.10 or higher** is installed.

1. Create Python environment Create and activate a virtual environment, then install dependencies in editable mode:

```
$ python3 -m venv venv
$ source venv/bin/activate
```

After activating the environment, install the Test Automation framework using:

```
$ cd test_automation
$ pip install -e .
```

1.1.4 Add New Test Cases

Test files should be placed under the `tests/` directory and follow the naming convention `test_*.py`.

Test Structure

A typical test file should contain:

1. A `pytest` test function or test class
2. Usage of built-in fixtures provided by the framework
3. Validation using framework helpers

Example:

```
def test_basic_example():
    assert True
```

> **Note**
>
> Refer to FVP and FPGA sections for target specific fixtures and helpers.

Refer to the target specific documentation for detailed setup and execution steps:

FVP Platform

Prerequisites

To run tests locally on FVP, make sure to have:

1. FVP binary

   The required Fast Models (FVP) executable must be installed on the host machine. When running pytest, the path to the binary must be passed explicitly through: `--fvp-binary /opt/arm/FVP/models/Linux64_GCC-9.3/FVP`

2. Crypto Library

   For targeting the **RD-Aspen FVP**, `Crypto.so` plugin is needed. The test will try to auto-detect this file under the FVP installation.

3. Build Images

   Tests require prebuilt images from the software reference stack build. The directory path provided to `--build-dir` must contain the required image files (such as `rse-rom-image.img`, `ap-flash-image.img`, and `.wic`) needed to boot the platform.

   These images may come from a local build or downloaded from any source. The images must be present under the directory passed to `--build-dir`.

```
$ pytest -s tests/test_sample.py \
  --config ./my_platform_config.yaml \
  --build-dir ~/my-reference-stack/build/tmp/deploy/images/rd-aspen \
  --fvp-binary /opt/arm/FVP \
  --platform fvp_rd_aspen
```

Configuration

Follow all the steps in *Prerequisites* and *Installation* before running tests.

Note

Before testing, ensure the `.wic` image name in the default YAML configuration matches the correct build architecture.

For baremetal builds, update this argument accordingly in the YAML file:

```
- "-C ros.virtio_block0.image_path=${BUILD_DIR}/\
   baremetal-image-fvp-rd-aspen.wic"
```

To run tests for other platform variants such as `RD-Aspen Cfg2`, some additional parameters may need to be updated or modified in the YAML configuration. For example, enabling an additional UART cluster terminal and adjust related entries in `required_terminals`, `prompts`, and `port_map` to include `terminal_uart_si_cluster1`.

Other variant-specific changes (like additional telnet ports or prompt patterns) should be updated as needed before running tests. For instance:

```
--parameter css.smb.si.terminal_uart_si_cluster1.start_telnet=0
```

Update or extend arguments like the above based on the specific platform variant being tested, such as RD-Aspen Cfg1 or Cfg2.

Running Sample Test

1. Default config:

```
$ pytest -s tests/test_sample.py \
  --config test_automation/configs/standalone_config.yaml \
  --build-dir /tmp/images/ \
  --fvp-binary /tmp/FVP \
  --platform fvp_rd_aspen
```


A successful sample test run should produce output similar to the following:

```
============================================================␣
↪test session starts␣
↪============================================================
platform linux -- Python 3.10.x, pytest-x.x.x, pluggy-x.x.x --
↪<python install dir>/bin/python3
cachedir: .pytest_cache
rootdir: <rootdir path>
configfile: pyproject.toml
plugins: timeout-x.x.x

collected 1 item

tests/test_sample.py::test_sample PASSED

============================================================␣
↪1 passed in 205.51s (0:03:25)␣
↪============================================
```

The sample test typically completes within 3-4 minutes on a standard host system (but may vary depending on host system performance).

2. Custom config:

```
$ pytest -s tests/test_sample.py \
  --config ./my_platform_config.yaml \
  --build-dir /tmp/images/ \
  --fvp-binary /tmp/FVP \
  --platform fvp_rd_aspen
```

Note

- The `--platform` argument must exactly match the platform value in the config file (e.g., `fvp_rd_aspen`).
- The value should contain "rd_aspen" if tests are for RD-Aspen (e.g., `fvp_rd_aspen`).
- Default logging level is INFO. To enable DEBUG logs, add `--debug-logs`.
- For more detailed output during debugging, extra pytest arguments can be used: `-rs --setup-show -vv`.
- If you don't see INFO logs after using verbose command line options, use `-o log_cli=true --log-cli-level=INFO`.
- Environment variables can be used:
  - `FVP_BINARY` instead of `--fvp-binary`
  - `FVP_CRYPTO_PATH` for location of `Crypto.so`

Add New Test Cases for FVP

FVP tests use built-in fixtures such as `platform_base_obj` and related helpers to interact with the platform via console-based command execution and prompt-based validation.

Example:

```
class TestMyExample:
   def test_my_example(self, platform_base_obj, platform_name):
      code, output = platform_base_obj.mgr.execute_command_with_prompt_capture(
         port=platform_base_obj.default_console,
         command="echo hello",
         timeout=10
      )
      assert code == 0
      assert "hello" in output
```

Note

- Always use `session_manager.wait_for_prompt_in_log()` or `session_manager.execute_command_with_prompt_capture()`.
- Match terminal names as in the YAML config.
- `conftest.py` provides global fixtures — no need to redeclare managers in each test.

FPGA Platform

Prerequisites

To run tests on FPGA hardware, make sure to have:

1. SSH access

   Make sure to have SSH access to the remote host where the FPGA is connected. The framework interacts with the FPGA through this host (remote execution).

2. FPGA host specification

   The FPGA hostname must be provided when running pytest using:

   `--host <hostname>`

3. Payload archive availability

   The local payload archive specified in the YAML configuration must be available. This archive should contain FPGA bootable images generated from the software reference stack build.

Configuration

FPGA platforms are defined in a YAML configuration file and must include:

- `type: fpga`
- `username`
- `port` (optional)
- `local_payload_path`

- `remote_payload_path`
- `boot_cmd`

The host name is not read from YAML and must be provided through the command line using `--host <fpga_hostname>`.

Running Sample FPGA Test

Follow all the steps in *Prerequisites* and *Installation* before running tests. The following command runs the sample FPGA boot test using the default FPGA config:

```
$ pytest -s test_automation/tests/test_fpga_boot.py \
  --config test_automation/test_automation/configs/fpga_config.yaml \
  --platform <platform_name> \
  --host <fpga_hostname>
```

A successful sample FPGA boot test run should produce output similar to the following:

```
============================================================ test session starts␣
↪============================================================
...

test_automation/tests/test_fpga_boot.py
SETUP    S fpga_device
      tests/test_fpga_boot.py::TestFPGABoot::test_boot_artifacts_present␣
↪(fixtures used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_sample_command_check_in_uart␣
↪(fixtures used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_uart_role_mapping (fixtures␣
↪used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_scp_boot_completed (fixtures␣
↪used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_detect_scp_safety_uart␣
↪(fixtures used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_detect_rse_uart (fixtures␣
↪used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_any_uart_contains_el3_exit␣
↪(fixtures used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_detect_linux_uart (fixtures␣
↪used: fpga_device, request).
      tests/test_fpga_boot.py::TestFPGABoot::test_uart_mapping_summary␣
↪(fixtures used: fpga_device, request).
TEARDOWN S fpga_device

================================================ 9 passed in 545.37s (0:09:05)␣
↪================================================
```

The sample test typically completes within 9-10 minutes on a standard host system (but may vary depending on host system performance).

During execution, the framework:

- Establishes an SSH connection.
- Uploads the payload archive.

- Launches the configured boot command.
- Detects the RUN_DIR.
- Logs in automatically after detecting the login prompt.
- Downloads the complete RUN_DIR logs after execution.

Add New Test Cases for FPGA

FPGA tests use the `fpga_device` fixture and validate behavior using boot artifacts, UART logs, and remote execution utilities.

Example:

```
class TestMyFPGAExample:
    def test_boot_artifacts_present(self, fpga_device):
        run_dir = fpga_device.get_run_dir()
        assert run_dir
        bootlog = fpga_device.log_path()
        assert bootlog.exists()
        assert bootlog.stat().st_size > 0
```

Note

- Refer to the common "Add New Test Cases" section for generic test structure guidance.
- Use `fpga_device` for FPGA-specific test flows.
- Prefer existing FPGA helper methods such as `get_run_dir()`, `log_path()`, `run_sample_command()`, and `get_login_uart()` where applicable.
- Use `fpga_device.net` helpers for UART log scanning and remote log inspection.

Note

- The `--platform` argument must match the platform defined in the config file.
- Default logging level is INFO.
- To enable DEBUG logs, add `--debug-logs`.
- Extra pytest arguments can be used: `-rs --setup-show -vv`.

1.2 API Documentation

1.2.1 Modules

This section explains about the core classes and their relationships in the Test Automation.

test_automation

Submodules

test_automation.cli

Functions

main()	Entry point for the Test Automation CLI.

Module Contents

test_automation.cli.**main**()

Entry point for the Test Automation CLI.

Returns
None.

test_automation.configs

Submodules

test_automation.configs.config

Central loader for config, with optional BUILD_IMAGE_DIR and FVP_BINARY substitution. Supports both list- and mapping-style 'platforms' sections in YAML.

Classes

Config	Central config loader to load YAML, set environment variable and expose

Functions

_expand(obj)	Expand ${ENV} in all strings.
_set_env_vars(build_dir, fvp_binary)	Set process environment variables for FVP configuration.
_load_yaml_expanded(path)	Load YAML configuration from a file and expand environment variables.
_normalize_platforms(data)	Normalize the `platforms` section of the YAML configuration.
_dict_to_obj(obj)	Recursively convert dictionaries to SimpleNamespace and lists to lists

Module Contents

test_automation.configs.config.**_expand**(*obj*)

Expand ${ENV} in all strings.

Parameters
obj (*Any*) – Python objects like strings, lists, and dictionaries are transformed.

Returns
The environment variables and user-home markers expanded wherever applicable.

Return type
Any

test_automation.configs.config.**_set_env_vars**(*build_dir*, *fvp_binary*)

Set process environment variables for FVP configuration.

Parameters

- **build_dir** (*Optional[str]*) – Path to set as BUILD_DIR. If None, the variable is not set.
- **fvp_binary** (*Optional[str]*) – Path to set as FVP_BINARY. If None, the variable is not set.

Returns
None

Return type
None

test_automation.configs.config.**_load_yaml_expanded**(*path*)

Load YAML configuration from a file and expand environment variables.

Parameters
path (*str*) – Path to a YAML configuration file.

Returns
Parsed and expanded YAML content as a dictionary.

Raises

- **FileNotFoundError** – If the YAML file cannot be found.
- **yaml.YAMLError** – If the YAML content cannot be parsed.

Return type
Dict[str, Any]

test_automation.configs.config.**_normalize_platforms**(*data*)

Normalize the platforms section of the YAML configuration.

Ensures platforms are returned as a dictionary keyed by platform name, regardless of whether the YAML used mapping or list style.

Parameters
data (*Dict[str, Any]*) – Parsed and expanded YAML configuration.

Returns
Mapping of platform name to platform configuration dictionary.

Return type
Dict[str, Dict[str, Any]]

test_automation.configs.config.**_dict_to_obj**(*obj*)

Recursively convert dictionaries to SimpleNamespace and lists to lists of converted entries. Scalars are returned unchanged.

Parameters
obj (*Any*) – object to convert (dict/list/scalar).

Returns
converted object: SimpleNamespace for dicts, lists for lists.

Return type
Any

class test_automation.configs.config.**Config**(*path*, *build_dir=None*, *fvp_binary=None*)

Central config loader to load YAML, set environment variable and expose platforms as a dict keyed by name.

Parameters

- **path** (*str*)
- **build_dir** (*Optional[str]*)
- **fvp_binary** (*Optional[str]*)

platforms: Dict[str, Dict[str, Any]]

_raw

path

build_dir = None

fvp_binary = None

get_platform(*name*)

Return the configuration dictionary for a specific platform.

Parameters
name (*str*) – Platform name key as defined in the YAML `platforms`.
Returns
The platform configuration mapping for `name`.
Raises
KeyError – If `name` is not present in *platforms*.
Return type
Dict[str, Any]

to_dict()

Return the full expanded YAML content as a dictionary.

Returns
A shallow copy of the expanded YAML content.
Return type
Dict[str, Any]

get_platform_object(*name*)

Return an object (attributes) of the platform config.

Parameters
name (*str*) – Platform key under 'platforms' in the YAML.
Returns
SimpleNamespace (recursive) representing the platform.
Raises
KeyError – If platform not present.
Return type
Any

test_automation.targets

Submodules

test_automation.targets.fpga

Submodules

test_automation.targets.fpga.autofpganetworking

Auto FPGA networking manager.

This module provides an SSH based session manager used to boot and interact with an FPGA target.

Attributes

logger	
RUN_DIR_RE	
_UART_LOG_NAME_RE	
_PROMPT_RE	
_LOGIN_MARKER	

Classes

SSHConfig	SSH connection settings for the FPGA host.
_OutputTarget	Destination for execution output.
_OutputBuffer	Accumulates execution output text between recv() calls
AutoFPGANetworking	SSH based networking manager for FPGA targets.

Functions

_read_all(stdout)	Read and decode all data from a Paramiko stdout-like stream.

Module Contents

test_automation.targets.fpga.autofpganetworking.**logger**

test_automation.targets.fpga.autofpganetworking.**RUN_DIR_RE**

test_automation.targets.fpga.autofpganetworking.**_UART_LOG_NAME_RE**

test_automation.targets.fpga.autofpganetworking.**_PROMPT_RE**

test_automation.targets.fpga.autofpganetworking.**_LOGIN_MARKER = 'login:'**

test_automation.targets.fpga.autofpganetworking.**_read_all**(*stdout*)

Read and decode all data from a Paramiko stdout-like stream.

Parameters

stdout – Stream object returned by Paramiko exec_command.

Returns

Decoded text (UTF-8 with replacement on decode errors).

Return type
str

class test_automation.targets.fpga.autofpganetworking.**SSHConfig**

SSH connection settings for the FPGA host.

Parameters

- **host** – Remote hostname or IP.
- **username** – SSH username.
- **password** – Optional password used for password or interactive auth.
- **port** – SSH port.
- **connect_timeout_s** – Connection/auth timeout in seconds.

host: str

username: str

password: str | None = None

port: int = 22

connect_timeout_s: int = 20

class test_automation.targets.fpga.autofpganetworking.**_OutputTarget**

Destination for execution output.

Parameters

- **log_file** – local boot log file handle
- **on_line** – optional callback invoked per output record

log_file: any

on_line: Callable[[str], None] | None = None

class test_automation.targets.fpga.autofpganetworking.**_OutputBuffer**

Accumulates execution output text between recv() calls until complete log records can be flushed.

text: str = ''

class test_automation.targets.fpga.autofpganetworking.**AutoFPGANetworking**(*cfg*, **args*, ***kwargs*)

Bases: *test_automation.utils.networking_base.BaseNetworkManager*

SSH based networking manager for FPGA targets.

Provides remote session management, command execution, log streaming, and console interaction utilities required to control and monitor an FPGA platform during automated testing.

Parameters
cfg (SSHConfig)

_resolve_platform_name(*args*, *kwargs*)

Resolve platform_name from kwargs or first positional arg.
Parameters
- **args** – Positional args passed to the constructor.

- **kwargs** – Keyword args passed to the constructor.

Returns
Platform name as string.

Return type
str

_resolve_log_paths(*kwargs*)

Resolve LogPathOptions from kwargs (log_paths) or overrides legacy.

Parameters
kwargs – Keyword args passed to the constructor.

Returns
Structured log path configuration.

Return type
test_automation.targets.fpga.fpga_runtime_options.LogPathOptions

cfg

platform_name = ''

_override_log_dir

_override_log_prefix = ''

_client: paramiko.SSHClient | None = None

_connected = False

_last_run_dir: str | None = None

_run_dir_lock

_run_dir_event

_run_ts: str | None = None

_run_local_dir: pathlib.Path | None = None

_remote_run_logs_local_dir: pathlib.Path | None = None

_boot_log_local: pathlib.Path | None = None

_run_thread: threading.Thread | None = None

_run_exit_status: int | None = None

_run_chan: paramiko.Channel | None = None

_initialize_ssh_client()

Create and configure a Paramiko SSH client.

Returns
Configured SSH client instance.

Return type
paramiko.SSHClient

_authenticate_with_password(*client*)

Attempt SSH authentication using password-based authentication.

Parameters
client (*paramiko.SSHClient*) – Initialized SSH client.

Raises

- **paramiko.AuthenticationException** – If password auth fails.
- **Exception** – For other SSH transport/connection errors.

Return type
None

_handle_interactive_prompts(*_title*, *_instructions*, *prompt_list*)

Paramiko keyboard-interactive callback.

For prompts containing the word "password" (case-insensitive), the configured password is returned; otherwise an empty response is used.

Parameters
- **_title** – Prompt title (unused).
- **_instructions** – Prompt instructions (unused).
- **prompt_list** – List of (prompt, echo) tuples.

Returns
List of response strings matching `prompt_list` length

_authenticate_with_interactive(*client*)

Attempt SSH authentication using keyboard-interactive authentication.

Parameters
client (*paramiko.SSHClient*) – Initialized SSH client.

Raises
RuntimeError – If the SSH transport cannot be obtained.

Return type
None

connect()

Establish an SSH connection to the FPGA host.

The method first attempts standard password authentication and falls back to keyboard based authentication when required by the server.

Raises
RuntimeError – If SSH connectivity or authentication fails.

Return type
None

disconnect()

Close SSH connection and clear internal state.

This closes the Paramiko client (if present) and resets the connected flag.

Return type
None

_ensure_ssh_client()

Ensure the SSH client is connected.

Raises
RuntimeError – If the SSH client is not connected.

Return type
None

_upload_binary(*src*, *remote_archive*)

Upload an archive to the remote host using SFTP.

Parameters
- **src** (*pathlib.Path*) – Local archive path.
- **remote_archive** (*str*) – Destination path on the remote host.

Raises
RuntimeError – If SSH is not connected.

Return type
None

`_build_extract_cmd`(*src_name*, *remote_base_dir*)

Build an extraction command for a supported archive type.

Parameters
- **src_name** (*str*) – Archive file name (not full path).
- **remote_base_dir** (*str*) – Remote directory to extract into.

Returns
Shell command string used for extraction.

Raises
RuntimeError – If the archive extension is unsupported.

Return type
str

`_detect_fpga_workdir`(*remote_base_dir*)

Detect a top-level directory created by extracting the archive and lists directories under `remote_base_dir` and uses the most recently modified entry as a candidate work directory. If no directory is found, the base directory is returned.

Parameters
remote_base_dir (*str*) – Remote extraction base directory.

Returns
Detected work directory path.

Raises
RuntimeError – If SSH is not connected.

Return type
str

`copy_payloads_to_remote`(*local_archive*, *remote_base_dir*)

Upload an archive to the remote host, extract it, and return workdir.

Parameters
- **local_archive** (*str*) – Local path to an archive containing FPGA binary.
- **remote_base_dir** (*str*) – Remote directory where the archive will be uploaded and extracted.

Returns
The top-level extracted directory if one is detected, otherwise `remote_base_dir`.

Raises
- **RuntimeError** – If SSH is not connected.
- **FileNotFoundError** – If `local_archive` does not exist.
- **RuntimeError** – If extraction fails.

Return type
str

`_ensure_local_run_dirs`()

Ensure local directories exist for boot logs and downloaded RUN_DIR logs.

Creates: - <CWD>/logs/<platform>_<ts>/ - <CWD>/logs/<platform>_<ts>/remote_run_logs/ - <CWD>/logs/<platform>_<ts>/<platform>_boot_<ts>.log

Return type
None

`property boot_log_path: pathlib.Path | None`

Return the local boot log file path.

Returns
Path to the local boot log file, if initialized.

Return type
Optional[pathlib.Path]

property local_run_dir: pathlib.Path | None

Return the local run directory created for this execution.

Returns
Path to the local run directory, if initialized.

Return type
Optional[pathlib.Path]

property local_remote_run_logs_dir: pathlib.Path | None

Return the local directory where remote RUN_DIR logs are downloaded.

Returns
Path to the local remote logs directory, if initialized.

Return type
Optional[pathlib.Path]

property last_run_dir: str | None

Return the most recently detected remote `RUN_DIR`.

Returns
Remote RUN_DIR string, if detected.

Return type
Optional[str]

_create_full_cmd(*env_setup_cmds*, *remote_cmd*, *remote_workdir*)

Compose the full remote command string.

Parameters
- **env_setup_cmds** (*Sequence[str]*) – Sequence of shell commands to prepare the environment (e.g. `source ...`).
- **remote_cmd** (*str*) – Main command to execute on the remote host.
- **remote_workdir** (*Optional[str]*) – Optional remote working directory to `cd` into before executing `remote_cmd`.

Returns
Fully composed command string suitable for Paramiko exec.

Return type
str

_open_exec_channel(*full_cmd*)

Open a PTY-enabled Paramiko session channel and execute `full_cmd`.

Parameters
full_cmd (*str*) – Command string returned by *_create_full_cmd()*.

Returns
Paramiko channel used to stream output and obtain exit value.

Raises
RuntimeError – If SSH is not connected or transport is missing.

Return type
paramiko.Channel

_write_and_process_line(*text*, *log_file*, *on_line*)

Write output text to the boot log, invoke callback, and detect RUN_DIR.

Parameters
- **text** (*str*) – Text chunk/record to write.
- **log_file** – Open file handle to write to.
- **on_line** (*Optional[Callable[[str], None]]*) – Optional callback invoked with the written text.

Return type
None

_flush_complete_records(*sink*, *buffer*)

Flush complete newline-terminated records from the output buffer.

Parameters
- **sink** (_OutputTarget) – Output sink configuration.
- **buffer** (_OutputBuffer) – Output buffer to flush.

Return type
None

_process_output_log(*chunk*, *sink*, *buffer*)

Decode a received byte chunk and append it to the output buffer.

Parameters
- **chunk** (*bytes*) – Bytes read from Paramiko channel.
- **sink** (_OutputTarget) – Output sink configuration.
- **buffer** (_OutputBuffer) – Output buffer to append to.

Return type
None

_finalize_run_output(*exec_stream*, *sink*, *buffer*)

Drain remaining output after process termination and record exit status.

Parameters
- **exec_stream** (*paramiko.Channel*) – Paramiko channel.
- **sink** (_OutputTarget) – Output sink configuration.
- **buffer** (_OutputBuffer) – Output buffer.

Return type
None

_read_available_output_once(*exec_handle*, *sink*, *buffer*)

Read and process available output once (non-blocking).

Parameters
- **exec_handle** (*paramiko.Channel*) – Paramiko channel.
- **sink** (_OutputTarget) – Output sink configuration.
- **buffer** (_OutputBuffer) – Output buffer.

Return type
None

_monitor_execution_output(*, *exec_handle*, *on_line*)

Monitor output from a Paramiko channel and stream to boot log.

This method runs until the remote command exits and writes all received output into the local boot log file.

Parameters
- **exec_handle** (*paramiko.Channel*) – Paramiko channel that is executing remote command.
- **on_line** (*Optional[Callable[[str], None]]*) – Optional callback invoked for each flushed log record.

Return type
None

start_remote_run(*run_spec=None*, *on_line=None*, ***kwargs*)

Start the remote run command in a background thread and stream output to the local boot log.

Parameters
- **run_spec** (*Optional[*test_automation.targets.fpga.fpga_runtime_options.RemoteRunSpec*]*) – Optional structured run

specification. If not provided, legacy keyword arguments are used to construct it.
- **on_line** (*Optional[Callable[[str], None]]*) – Optional callback invoked for each flushed output.
- **kwargs** – Backward-compatible keyword arguments.

Raises
RuntimeError – If *connect()* was not called first.
Return type
None

wait_for_run_dir(*timeout_s=180*)

Wait until RUN_DIR is detected from the remote output stream.

Parameters
timeout_s (*int*) – Maximum time to wait in seconds.
Returns
Detected RUN_DIR path.
Raises
TimeoutError – If RUN_DIR is not detected within timeout.
Return type
str

wait_remote_exit(*timeout_s=None*)

Wait for the remote run thread to complete and return exit status.

Parameters
timeout_s (*Optional[int]*) – Optional timeout for joining the run thread.
Returns
Exit status if available, otherwise None.
Return type
Optional[int]

find_uart_login(*run_dir*, *, *tail_lines=120*)

Search UART logs under the given RUN_DIR for a login prompt.

This method scans all UART_log_vuart_* files located under run_dir and inspects the last tail_lines lines of each log for the presence of a login marker (e.g., "login:"). If a match is found, the corresponding VUART index, PTS number, and log path are returned.

Parameters
- **run_dir** (*str*) – Remote RUN_DIR path containing UART log files.
- **tail_lines** (*int*) – Number of lines from the end of each UART log to inspect. Defaults to 120.

Returns
Tuple (vuart, pts, uart_log_path) if a login prompt is detected; otherwise None.
Return type
Optional[Tuple[int, int, str]]
Raises
RuntimeError – If the SSH client is not connected.

send_to_vuart_link_sftp(*run_dir*, *payload*, **kwargs*)

Write payload to RUN_DIR/vuart_<vuart>__dev_pts_<pts> using SFTP. This helper writes directly to the VUART link to avoid shell quoting issues.

Parameters
- **run_dir** (*str*) – Remote RUN_DIR path.
- **payload** (*str*) – Text to write to the VUART link.
- **kwargs** – Backward-compatible keyword args.

Raises
- **TypeError** – If vuart or pts are missing.

- **RuntimeError** – If SSH is not connected.

Return type
None

wait_for_shell_prompt(*uart_log_path*, *options=None*, ***kwargs*)

Wait until a root shell prompt is visible in the specified UART log.

This method periodically tails the UART log file and checks for a shell prompt pattern. It stops when the prompt is detected or when the timeout expires.

Parameters
- **uart_log_path** (*str*) – Remote UART log file path.
- **options** (*Optional[test_automation.targets.fpga.fpga_runtime_options.ShellPromptWaitOptions]*) – Optional structured wait configuration. If not provided, values may be supplied via backward-compatible keyword arguments.
- **timeout_s** – (kwarg) Maximum time in seconds to wait for the shell prompt. Defaults to 120.
- **poll_s** – (kwarg) Polling interval in seconds between log checks. Defaults to 1.0.
- **tail_lines** – (kwarg) Number of log lines to inspect on each poll. Defaults to 200.

Returns
None

Raises
- **RuntimeError** – If the SSH client is not connected.
- **TimeoutError** – If the shell prompt is not detected within the timeout.

Return type
None

wait_login_and_send_root(*options=None*, ***kwargs*)

Wait for a UART login prompt, then wait for a shell prompt.

This method scans available UART logs for a login marker, sends the `root` username to the detected VUART link, and then waits until a shell prompt becomes available.

Parameters
- **options** (*Optional[test_automation.targets.fpga.fpga_runtime_options.LoginWaitOptions]*) – Optional structured login wait configuration. If not provided, values may be supplied via backward-compatible keyword arguments.
- **timeout_s** – (kwarg) Maximum time in seconds to wait for the login prompt. Defaults to 900.
- **poll_s** – (kwarg) Polling interval in seconds between login checks. Defaults to 2.0.
- **tail_lines** – (kwarg) Number of UART log lines to inspect during each poll. Defaults to 150.
- **shell_prompt_timeout_s** – (kwarg) Maximum time in seconds to wait for the shell prompt after sending `root`. Defaults to 120.

Returns
Tuple `(uart_log_path, vuart, pts)` identifying the UART log file and its associated VUART and PTS numbers.

Returns
Tuple[str, int, int]

Raises
- **RuntimeError** – If the SSH client is not connected.
- **TimeoutError** – If the login prompt or shell prompt is not detected within the timeout.

Return type
Tuple[str, int, int]

`run_uart_command`(*cmd*, ***kwargs*)

Send a command to a specific UART by writing to its VUART link.

The command is appended with a newline and written to the corresponding `RUN_DIR/vuart_<vuart>__dev_pts_<pts>` link.

Parameters
- **cmd** (*str*) – Command string to send to the UART.
- **vuart** – (kwarg) VUART index.
- **pts** – (kwarg) PTS number associated with the VUART.

Returns
None

Raises
RuntimeError – If the SSH client is not connected.

Return type
None

`list_uart_logs`()

List UART log files under the current `RUN_DIR`.

Returns
Sorted list of tuples `(vuart, pts, uart_log_path)`.

Raises
RuntimeError – If SSH is not connected or RUN_DIR is unknown.

Return type
List[Tuple[int, int, str]]

`read_uart_log`(*uart_log_path*)

Read the full contents of a UART log file.

Parameters
uart_log_path (*str*) – Remote UART log file path.

Returns
UART log contents (UTF-8 decoded with replacement).

Raises
RuntimeError – If SSH is not connected.

Return type
str

`tail_uart_log`(*uart_log_path*, *, *lines=200*)

Read the tail of a UART log file.

Parameters
- **uart_log_path** (*str*) – Remote UART log file path.
- **lines** (*int*) – Number of lines from the end of the file to return.

Returns
UART log tail (UTF-8 decoded with replacement).

Raises
RuntimeError – If SSH is not connected.

Return type
str

`find_uarts_with_strings`(*strings*, *, *tail_lines=300*)

Find UART logs whose tail contains any of the provided strings.

Parameters
- **strings** (*List[str]*) – List of substrings to search for.
- **tail_lines** (*int*) – Number of tail lines to inspect for each UART log.

Returns
List of tuples `(vuart, pts, uart_log_path)` that matched.
Raises
RuntimeError – If SSH is not connected.
Return type
List[Tuple[int, int, str]]

start_uart_keepalive(***kwargs*)

This can help keep UART sessions active and stimulate console output while waiting for prompts.
Return type
None

stop_uart_keepalive()

Stop the UART keepalive thread if it is running.
Return type
None

map_uarts_by_predicate(*predicate*, ***, *tail_lines=300*)

Apply a predicate to UART log text and return matching UARTs.
Parameters
- **predicate** (*Callable[[str], bool]*) – Callable `predicate(text) -> bool`.
- **tail_lines** (*int*) – Number of lines from the UART log tail to inspect.

Returns
List of tuples `(vuart, pts, uart_log_path)`.
Return type
List[Tuple[int, int, str]]

download_run_dir_logs()

Download all files under the current remote `RUN_DIR` to local logs.

The destination directory is: `<CWD>/logs/<platform>_<ts>/remote_run_logs/`
Returns
Local destination path for downloaded remote logs.
Raises
RuntimeError – If SSH is not connected or RUN_DIR is unknown.
Return type
pathlib.Path

_sftp_get_dir_recursive(*sftp*, *remote_dir*, *local_dir*)

Recursively download a remote directory using SFTP.
Parameters
- **sftp** (*paramiko.SFTPClient*) – Active Paramiko SFTP client.
- **remote_dir** (*str*) – Remote directory path to download.
- **local_dir** (*pathlib.Path*) – Local directory to write downloaded files into.

Return type
None

static _shell_escape(*s*)

Escape a string for safe inclusion in a single-quoted shell literal.
Parameters
s (*str*) – Input string.
Returns
Single-quoted, shell-safe literal.
Return type
str

test_automation.targets.fpga.fpga_controller

Attributes

logger	

Classes

FPGAPlatformConfig	Structured configuration for an FPGA platform session.
FPGAController	FPGA controller implementing the Device interface.

Functions

load_fpga_controller_from_yaml(yaml_path platform_name)	Create an FPGAController instance from a YAML configuration file.

Module Contents

test_automation.targets.fpga.fpga_controller.**logger**

class test_automation.targets.fpga.fpga_controller.**FPGAPlatformConfig**

Structured configuration for an FPGA platform session.

Parameters

- **platform_name** – Logical platform name used for log directory naming.
- **host** – Remote hostname or IP address of the FPGA host.
- **username** – SSH username.
- **password** – Optional SSH password for password or interactive auth.
- **port** – SSH port.
- **connect_timeout_s** – SSH connection/auth timeout in seconds.
- **ready_timeout_s** – Timeout (seconds) for readiness checks.
- **login_timeout_s** – Timeout (seconds) for UART login detection and establishing a shell prompt.
- **hpc_env_setup** – Sequence of shell commands used to set up environment before launching the remote boot command.
- **remote_cmd** – Remote command used to start FPGA boot/run.
- **remote_payload_path** – Optional remote path to upload/extract a payload archive into.
- **local_payload_path** – Optional local archive path to upload to the remote host.
- **log_dir** – Optional override base directory for logs.

- **log_prefix** – Optional override prefix for log directory naming.

platform_name: str

host: str

username: str

password: str | None

port: int

connect_timeout_s: int

ready_timeout_s: int

login_timeout_s: int

hpc_env_setup: Sequence[str]

remote_cmd: str

remote_payload_path: str | None = None

local_payload_path: str | None = None

log_dir: str | None = None

log_prefix: str | None = None

class test_automation.targets.fpga.fpga_controller.**FPGAController**(*cfg*)

Bases: *test_automation.utils.device.Device*

FPGA controller implementing the Device interface.

This controller manages the lifecycle of a remote FPGA session over SSH. It is responsible for connecting to the FPGA host, optionally uploading required binaries, launching the configured remote command, monitoring system readiness, handling automated UART login, and exposing runtime state and logs to the test framework.

Parameters
cfg (FPGAPlatformConfig)

cfg

net

_running = False

_login_uart: Tuple[str, int, int] | None = None

init()

Establish an SSH connection to the FPGA host.

Returns
None
Raises
RuntimeError – If SSH connectivity/authentication fails.
Return type
None

`start()`

Start the FPGA boot workflow and login as root.

This method: - Launches the command in the background on the remote host - Waits for a login prompt on the UART - Sends the root login - Marks the controller as running

Returns
None

Raises
- **RuntimeError** – If not connected or login UART cannot be established.
- **TimeoutError** – If login prompt or shell prompt is not detected within configured timeout.

Return type
None

`stop()`

Stop the FPGA run and collect logs.

This method attempts to wait briefly for remote command to exit, then downloads the current RUN_DIR logs into:

```
<CWD>/logs/<platform>_<timestamp>/remote_run_logs/
```

Finally, it disconnects the SSH session regardless of outcome.

Returns
None

Return type
None

`reset()`

Reset the FPGA by stopping and restarting the controller.

This is equivalent to calling `stop()`, `init()`, and `start()`

in sequence.

Returns
None

Return type
None

`wait_ready`(*timeout_s=None*)

Wait until the FPGA run directory is created.

This method blocks until remote command creates the RUN_DIR on the remote host, indicating that the FPGA boot has progressed far enough to be considered ready.

Parameters
timeout_s (*Optional[int]*) – Maximum time to wait in seconds. If None, a default timeout is used.

Returns
None

Return type
None

`is_running()`

Check whether the FPGA controller is currently running.

Returns
True if the controller is running, False otherwise.

Return type
bool

get_ports()

Return exposed service ports for the FPGA. FPGA platforms do not currently expose network service ports.

Returns
An empty mapping.

Return type
Dict[str, int]

log_path()

Return the primary local boot log path.

Returns
Path to the boot log file, or a fallback logs directory if unavailable.

Return type
pathlib.Path

get_run_dir()

Return the remote run directory.

Returns
Path to the RUN_DIR on the remote host, or `None` if unavailable.

Return type
Optional[str]

get_login_uart()

Return UART details for the active login session.

Returns
A tuple of `(uart_log_path, vuart, pts)`.

Raises
RuntimeError – If the login UART has not been established.

Return type
Tuple[str, int, int]

run_sample_command(*command*)

Execute a command on the logged-in UART session.

Parameters
command (*str*) – Shell command to execute.

Returns
`None`

Return type
None

test_automation.targets.fpga.fpga_controller.**load_fpga_controller_from_yaml**(*yaml_path*, *platform_name*, *host=None*)

Create an FPGAController instance from a YAML configuration file.

Parameters

- **yaml_path** (*pathlib.Path*) – Path to the YAML configuration file.
- **platform_name** (*str*) – Name of the platform entry to load.
- **host** (*Optional[str]*) – Optional override host name/IP. If not provided, the YAML platform object's `host` attribute is used.

Returns
Constructed FPGAController instance.

Raises
ValueError – If host cannot be resolved.

Return type
FPGAController

test_automation.targets.fpga.fpga_runtime_options

Runtime option dataclasses used by the FPGA networking and controller layers for log paths, remote command execution, and UART login handling.

Classes

LogPathOptions	Log destination overrides.
RemoteRunSpec	Structured specification for executing a remote command.
ShellPromptWaitOptions	Configuration options for waiting on a shell prompt in a UART log.
LoginWaitOptions	Configuration options for waiting on a UART login prompt and shell prompt.

Module Contents

class test_automation.targets.fpga.fpga_runtime_options.**LogPathOptions**
Log destination overrides.

Parameters
- **log_dir** – Optional base directory for logs (default: <cwd>/logs).
- **log_prefix** – Optional prefix for generating timestamped log folders.

log_dir: str | None = None

log_prefix: str | None = None

class test_automation.targets.fpga.fpga_runtime_options.**RemoteRunSpec**
Structured specification for executing a remote command.

Parameters
- **env_setup_cmds** – Sequence of shell commands used to prepare the environment before running the main remote command.
- **remote_cmd** – The primary command to execute on the remote host.
- **remote_workdir** – Optional remote working directory. If provided, the execution flow will change into this directory before running `remote_cmd`.

env_setup_cmds: Sequence[str]

remote_cmd: str

remote_workdir: str | None = None

class test_automation.targets.fpga.fpga_runtime_options.**ShellPromptWaitOptions**
Configuration options for waiting on a shell prompt in a UART log.

Parameters

- **timeout_s** – Maximum time to wait for detection of login prompt.
- **poll_s** – Polling interval in seconds between login prompt checks.
- **tail_lines** – Number of lines from the end of the UART log inspected when searching for the login marker.

timeout_s: **int = 120**

poll_s: **float = 1.0**

tail_lines: **int = 200**

class test_automation.targets.fpga.fpga_runtime_options.**LoginWaitOptions**

Configuration options for waiting on a UART login prompt and shell prompt.

Parameters

- **timeout_s** – Maximum time to wait for detection of login prompt.
- **poll_s** – Polling interval in seconds between login prompt checks.
- **tail_lines** – Number of lines from the end of the UART log inspected when searching for the login marker.
- **shell_prompt_timeout_s** – Maximum time to wait for the shell prompt after sending the login username.

timeout_s: **int = 900**

poll_s: **float = 2.0**

tail_lines: **int = 150**

shell_prompt_timeout_s: **int = 120**

test_automation.targets.fvp

Submodules

test_automation.targets.fvp.autofvpnetworking

Module to manage telnet sessions

Attributes

logger	

Classes

TelnetSessionManager	Manage multiple telnet sessions for exposed telnet ports of FVP.

Module Contents

test_automation.targets.fvp.autofvpnetworking.**logger**

class test_automation.targets.fvp.autofvpnetworking.**TelnetSessionManager**(*config=None*, *platform=None*, ***options*)

Manage multiple telnet sessions for exposed telnet ports of FVP.

1. Read FVP boot log to get terminal-port mappings
2. Spawn pexpect based telnet session for each terminal
3. Continuously log for each telnet session's output
4. Check for login and shell prompts, send commands and redirect outputs.
5. Drain logs and clean up sessions on demand.

Parameters
platform (*Optional[str]*)

test_name: str | None

_init_load_platform_cfg(*config*, *platform*)

Load the platform configuration from the provided configuration object.

Parameters
- **config** – Configuration object exposing get_platform().
- **platform** (*str*) – Platform key to retrieve from the configuration.

Returns
Platform configuration dictionary returned by get_platform().

Raises
KeyError – If the platform key is not found.

Return type
dict

_init_load_timeouts(*platform_cfg*, *platform*)

Initialize login and shell prompt timeouts from platform configuration.

Parameters
- **platform_cfg** (*dict*) – Platform configuration dictionary containing timeout settings.
- **platform** (*str*) – Platform name (used for error messages).

Raises
KeyError – If required timeout keys are missing.

Return type
None

_init_load_prompts(*platform_cfg*, *platform*)

Initialize default and per-terminal prompt strings from configuration.

Parameters
- **platform_cfg** (*dict*) – Platform configuration dictionary containing prompt mappings.
- **platform** (*str*) – Platform name (used for error messages).

Raises
KeyError – If default prompt keys are missing.

Return type
None

`_init_required_terminals`(*platform_cfg*, *platform*)

Initialize the list of required terminal names.

Parameters
- **platform_cfg** (*dict*) – Platform configuration dictionary.
- **platform** (*str*) – Platform name (used for error messages).

Raises
KeyError – If `required_terminals` is missing or empty.

Return type
None

`_init_port_map`(*platform_cfg*)

Initialize terminal-to-port mapping for the current platform.

Parameters
platform_cfg (*dict*) – Platform configuration dictionary containing port mappings.

Return type
None

`_init_paths_and_state`(*base_log_root*, *timeout*, *encoding*)

Initialize filesystem paths, log directories, and runtime state.

Parameters
- **base_log_root** (*str*) – Base directory path for all telnet log files.
- **timeout** (*int*) – Default timeout in seconds for telnet interactions.
- **encoding** (*str*) – Encoding used for log and I/O operations.

Return type
None

`_get_log_path`(*port*, *terminal_name=None*)

Get the log file path for a telnet session.

The log filename is constructed as `<terminal_name>_<port>.log` when a terminal name is provided. If no terminal name is available, it falls back to `telnet_<port>.log`.

Parameters
- **port** (*int*) – Telnet port number
- **terminal_name** (*Optional[str]*) – Optional terminal name

Returns
Full path to the log file

Return type
str

`create_and_start_session`(*port*, *terminal_name=None*)

Spawn a telnet session and start logging simultaneously.

Parameters
- **port** (*int*) – Telnet port of the telnet session.
- **terminal_name** (*Optional[str]*) – Optional name for prompt mapping.

Return type
None

`get_port`(*terminal_name*)

Return the port number for a terminal name, or None if not registered.

Parameters
terminal_name (*str*) – Terminal name to query.

Returns
Port number if known, else None.
Return type
Optional[int]

`pause_logging`(*port*)

Pause the background logging thread for a port.

This allows direct expect() calls on the session without the logging thread consuming the output.
Parameters
port (*int*) – Port number to pause logging for.
Return type
None

`resume_logging`(*port*)

Resume the background logging thread for a port.
Parameters
port (*int*) – Port number to resume logging for.
Return type
None

`_log_telnet_session`(*port*, *session*)

Continuously log data from the session into its logfile, logs errors and exits cleanly when the session or logfile is closed.
Parameters
- **port** (*int*) – Port number of the telnet session.
- **session** (*pexpect.spawn*) – Active pexpect telnet session instance.

Return type
None

`_log_iteration`(*port*, *session*)

One non-blocking expect iteration from the original logic.
Parameters
- **port** (*int*)
- **session** (*pexpect.spawn*)

Return type
bool

`_get_terminal_name`(*port*, *terminal_name*)

Resolve the terminal name for a given port.

Returns the explicit *terminal_name* if provided. Otherwise, uses *_term_for_port()* to resolve the name.
Parameters
- **port** (*int*)
- **terminal_name** (*Optional[str]*)

Return type
Optional[str]

`wait_for_prompt_in_log`(**args*, ***kwargs*)

Waits for a prompt/pattern in the telnet log file.
- If `regex` is True: `prompt` is treated as a regex pattern.
- If `regex` is False: `prompt` is treated as a literal string.
- **If `regex` is None: a heuristic decides if `prompt` looks like** a regex (auto-detect).

Parameters

- **args** – Positional arguments (port, prompt, timeout).
- **kwargs** – Keyword arguments used to pass options (port=..., prompt=..., timeout=...).

Returns
True if the prompt/pattern is found in the log, otherwise False.

Return type
bool

_extract_wait_args(*args*, *kwargs*)

Extract and validate wait_for_prompt_in_log arguments.

This helper pops known keyword arguments, applies positional args, validates there are no unknown kwargs, and ensures `port` and `prompt` are present.

Parameters
- **args** (*tuple*) – Positional args passed through.
- **kwargs** (*dict*) – Keyword args passed through (will be mutated).

Returns
Tuple (port, prompt, timeout, regex, ignore_case).

Raises
TypeError – on invalid or missing arguments.

Return type
tuple[int, str, int, bool | None, bool]

_pop_wait_kwargs(*kwargs*)

Pop recognized keyword arguments for the wait helper.

Recognized keys: port, prompt, timeout, regex, ignore_case.

Parameters
kwargs (*dict*) – Keyword arguments dict (mutated by popping).

Returns
(port, prompt, timeout, regex, ignore_case).

Return type
tuple[int | None, str | None, int, bool | None, bool]

_apply_positional_args(*args*, *current*)

Apply positional arguments to (port, prompt, timeout).

Parameters
- **args** (*tuple*) – Positional args tuple.
- **current** (*tuple[int | None, str | None, int]*) – Tuple (port, prompt, timeout).

Returns
Updated (port, prompt, timeout).

Raises
TypeError – If more than three positional args are provided.

Return type
tuple[int | None, str | None, int]

_check_for_regex(*text*)

Determine if a string looks like a regex.

Checks for common regex markers such as groups, classes, escapes, quantifiers and anchors.

Parameters
text (*str*) – User given string.

Returns
True if the string likely represents a regex pattern.

Return type
bool

_compile_prompt_pattern(*port*, *prompt*, *options*)

Compile the prompt into a regex.Pattern.

Parameters

- **port** (*int*) – Port number (used only for logging).
- **prompt** (*str*) – Prompt text or regex pattern.
- **options** (*tuple[bool, bool]*) – Tuple (regex, ignore_case).

Returns

Compiled pattern, or `None` on error.

Return type

re.Pattern | None

_poll_until_found(*wait_params*)

Poll the log file until the pattern is found or timeout occurs.

Parameters

wait_params (*tuple[int, str, int, bool, str, re.Pattern]*) – Tuple (port, prompt, timeout, regex, log_path, prompt_pattern).

Returns

True if match found, otherwise False.

Return type

bool

_is_timeout_reached(*now*, *deadline*)

Return True if the current time has passed the deadline.

Parameters

- **now** (*float*) – Current time (seconds since epoch).
- **deadline** (*float*) – Deadline timestamp.

Returns

True if timeout reached, otherwise False.

Return type

bool

_update_debug_status_if_needed(*debug_params*)

Update debug status if enough time has passed since last debug.

Parameters

debug_params (*tuple[int, float, float, int, float, float]*) – Tuple (port, now, started_at, timeout, last_debug_time, debug_interval).

Returns

Updated last debug time.

Return type

float

_file_contains(*path*, *regex*)

Return matched text if pattern is found, else False.

Parameters

- **path** (*str*) – Path to the log file to be scanned.
- **regex** (*re.Pattern*) – Compiled regular expression pattern to search for.

Returns

The matched text if the pattern is found, else `False`.

_debug_wait_status(*port*, *started_at*, *timeout*)

Log a periodic debug message while waiting for a prompt.

Parameters

- **port** (*int*) – Telnet port number associated with the session.
- **started_at** (*float*) – Timestamp when waiting started.
- **timeout** (*int*) – Total timeout value in seconds.

Returns
None
Return type
None

`login_if_needed`(*port*, *login_prompt=None*, *shell_prompt=None*)

Ensure a shell is ready on the given port, logging in if required.

Parameters
- **port** (*int*) – Telnet port number.
- **login_prompt** (*Optional[str]*) – Optional login prompt regex/string override.
- **shell_prompt** (*Optional[str]*) – Optional shell prompt regex/string override.

Returns
`True` if a shell prompt is observed; otherwise `False`.
Return type
bool

`_resolve_prompts`(*term_name*, *login_prompt*, *shell_prompt*)

Resolve effective login and shell prompts for a terminal.

Parameters
- **term_name** (*Optional[str]*) – Logical terminal name, if known.
- **login_prompt** (*Optional[str]*) – Optional login prompt override.
- **shell_prompt** (*Optional[str]*) – Optional shell prompt override.

Returns
Tuple where the first item may be `None` if no login is expected.
Return type
Tuple[Optional[str], str]

`_handle_login_path`(*port*, *login_prompt*, *shell_prompt*)

Handle the login flow if a login prompt is expected and observed.

Parameters
- **port** (*int*) – Telnet port number.
- **login_prompt** (*Optional[str]*) – Login prompt regex/string, if any.
- **shell_prompt** (*str*) – Expected shell prompt regex/string.

Returns
`True` if shell prompt is reached; else `False`.
Return type
bool

`_await_shell_after_login`(*port*, *shell_prompt*)

Wait for the shell prompt after credentials were submitted.

Parameters
- **port** (*int*) – Telnet port number.
- **shell_prompt** (*str*) – Expected shell prompt regex/string.

Returns
`True` if the shell prompt appears; else `False`.
Return type
bool

`_handle_no_login_path`(*port*, *shell_prompt*)

Ensure the shell is present when a login prompt is not expected/found.

Parameters
- **port** (*int*) – Telnet port number.
- **shell_prompt** (*str*) – Expected shell prompt regex/string.

Returns
`True` if the shell prompt is found; else `False`.

Return type
bool

`_term_for_port`(*port*)

Resolve the logical terminal name for a given telnet port.

Parameters
port (*int*) – Telnet port number.

Returns
The terminal name mapped to `port` or `None` if unknown.

Return type
Optional[str]

`execute_command_with_prompt_capture`(*port*, *command*, *timeout=None*)

Execute a shell command on the specified Telnet session, wait for its completion marker, and capture both exit code and output.

Parameters
- **port** (*int*) – Telnet port number associated with the active session.
- **command** (*str*) – The shell command to be executed.
- **timeout** (*Optional[int]*) – Optional timeout in seconds for command execution. If not provided, the default session timeout is used.

Returns
A tuple `(exit_code, output)` where: - `exit_code` is the integer exit status of the executed command. - `output` is the cleaned textual output of the command.

Raises
RuntimeError – If the Telnet session is not alive or terminal mapping cannot be resolved.

Return type
Tuple[int, str]

`_get_session_and_term`(*port*)

Fetch the live pexpect session and its logical terminal name.

Parameters
port (*int*) – Telnet port number whose session is required.

Returns
Tuple of `(session, terminal_name)`.

Raises
RuntimeError – If the session is not alive or the terminal name cannot be resolved.

Return type
Tuple[pexpect.spawn, str]

`_get_prompts_for_term`(*term_name*)

Resolve the effective login and shell prompts for a terminal.

Parameters
term_name (*str*) – Logical terminal name.

Returns
Tuple `(login_prompt, shell_prompt)`; the first may be `None` when no login is expected.

Raises
RuntimeError – If no shell prompt can be determined.

Return type
Tuple[Optional[str], str]

`_ensure_shell_ready`(*port*, *prompts*, *timeout*)

Ensure a shell is ready on the session, logging in if needed.

Parameters

- **port** (*int*) – Telnet port number.
- **prompts** (*Tuple[Optional[str], str]*) – Tuple (login_prompt, shell_prompt).
- **timeout** (*int*) – Seconds to wait for prompts.

Returns
`True` if a shell prompt is confirmed; else `False`.

Return type
bool

_create_full_command(*command*)

Build the final command and its completion marker.

Parameters
command (*str*) – Shell command to execute.

Returns
Tuple `(full_cmd, marker_literal, marker_regex)` where `marker_literal` is the plain prefix (e.g. `"CMD_DONE_"`) and `marker_regex` captures the exit code.

Return type
Tuple[str, str, re.Pattern]

_send_and_try_echo(*port*, *full_cmd*, *marker_literal*)

Send the full command and wait for the marker echo.

Parameters
- **port** (*int*) – Telnet port number.
- **full_cmd** (*str*) – Composite shell command to send.
- **marker_literal** (*str*) – Plain marker prefix expected in the echo.

Returns
`None`.

Return type
None

_await_completion(*port*, *expect*, *timeout*)

Wait for the command execution to complete by detecting either the completion marker or the shell prompt in the session output.

Parameters
- **port** (*int*) – Telnet port number of the session.
- **expect** (*Tuple[str, re.Pattern]*) – Tuple (shell_prompt, marker_regex).
- **timeout** (*int*) – Maximum number of seconds to wait for completion.

Returns
A tuple `(exit_code, error_buffer)`.

Return type
Tuple[Optional[int], Optional[str]]

_expect_marker_or_prompt(*port*, *expect*, *timeout*)

Waits for either the marker or the shell prompt.

:returns
[("marker", session) when marker matched (use session.match)] ("prompt", buffer) when shell prompt matched (session.before) ("timeout", buffer) on timeout (session.before)

Parameters
- **port** (*int*)
- **expect** (*Tuple[str, re.Pattern]*)
- **timeout** (*int*)

Return type
Tuple[str, object]

_parse_exit_from_match(*session*, *marker_regex*)

Extract the command exit code from the session match object.

Parameters

- **session** (*pexpect.spawn*) – The pexpect session whose last match contains the exit code.
- **marker_regex** (*re.Pattern*) – Compiled regex pattern that captures the exit code from the completion marker.

Returns

A tuple `(exit_code, None)`. Falls back to scanning the session buffer if the marker match cannot be parsed.

Return type

Tuple[int, None]

_parse_exit_from_buffer(*buffer*, *marker_regex*, *port*)

Parse the exit code from a plain buffer when the shell prompt appears.

Parameters

- **buffer** (*str*) – The text preceding the shell prompt.
- **marker_regex** (*re.Pattern*) – Compiled regex pattern used to locate and extract the exit code.
- **port** (*int*) – Telnet port number of the session (used for logging).

Returns

A tuple `(exit_code, error_buffer)`: - `exit_code` is the parsed integer exit status, or `None` if the marker was missing or invalid. - `error_buffer` is the raw buffer when no marker is found, otherwise `None`.

Return type

Tuple[Optional[int], Optional[str]]

_finalize_output_and_persist(*raw_output*, *meta*)

Clean command output, remove any buffer, and copy the result to file.

Parameters

- **raw_output** (*str*) – Raw session output captured before the prompt or marker.
- **meta** (*Tuple[str, re.Pattern, int, str]*) – Tuple (full_cmd, marker_regex, port, command).

Returns

The cleaned command output string.

Return type

str

_ensure_shell_or_login(*port*, *prompts*, *timeout*)

Ensure a usable shell, logging in first if necessary.

Parameters

- **port** (*int*) – Telnet port of the session.
- **prompts** (*Tuple[Optional[str], str]*) – Tuple (login_prompt, shell_prompt).
- **timeout** (*int*) – Seconds to wait for shell prompt if needed.

Returns

`True` if shell is ready, else `False`.

Return type

bool

_expect_pattern(*session*, *pattern*, *timeout*)

Wait for a pattern, returning whether it matched and any partial buffer.

Parameters

- **session** (*pexpect.spawn*) – Active pexpect session.
- **pattern** (*str*) – Regex/string to expect.

- **timeout** (*int*) – Seconds to wait for a match.

Returns
Tuple `(matched, buffer_on_timeout)` where `matched` is `True` if the pattern matched; otherwise `False` and `buffer_on_timeout` contains `session.before`.

Return type
Tuple[bool, str]

_already_at_shell(*port*, *shell_prompt*)

Check quickly whether we are already at the shell prompt.

Parameters
- **port** (*int*) – Telnet port of the session.
- **shell_prompt** (*str*) – Expected shell prompt regex/string.

Returns
`True` if the shell prompt is present, else `False`.

Return type
bool

_login_path_then_shell(*port*, *prompts*, *timeout*)

Attempt the login path, then wait for the shell prompt.

Parameters
- **port** (*int*) – Telnet port of the session.
- **prompts** (*Tuple[str, str]*) – Tuple (login_prompt, shell_prompt).
- **timeout** (*int*) – Seconds to wait for the shell prompt if needed.

Returns
`True` if shell is ready, else `False`.

Return type
bool

_after_saw_login_send_root_and_wait_shell(*port*, *shell_prompt*)

Send credentials after seeing the login prompt and wait for the shell.

Parameters
- **port** (*int*) – Telnet port of the session.
- **shell_prompt** (*str*) – Expected shell prompt regex/string.

Returns
`True` if shell prompt appears, else `False`.

Return type
bool

_no_login_prompt_then_wait_shell(*port*, *shell_prompt*, *timeout*)

If login prompt is not observed, wait directly for the shell prompt.

Parameters
- **port** (*int*) – Telnet port of the session.
- **shell_prompt** (*str*) – Expected shell prompt regex/string.
- **timeout** (*int*) – Seconds to wait for the shell prompt.

Returns
`True` if shell prompt appears, else `False`.

Return type
bool

_wait_for_shell_with_timeout(*port*, *shell_prompt*, *timeout*)

Wait for a shell prompt within a time span.

Parameters
- **port** (*int*) – Telnet port of the session.
- **shell_prompt** (*str*) – Expected shell prompt regex/string.
- **timeout** (*int*) – Seconds to wait for the shell prompt.

Returns
True if shell prompt appears, else False.
Return type
bool

_fallback_exit_from_buffer(*buffer*, *marker_regex*, *default=1*)

Try to extract the exit code from a buffer using the marker regex.
Parameters
- **buffer** (*str*) – Text captured before the prompt/marker.
- **marker_regex** (*re.Pattern*) – Compiled regex that captures the exit code in group 1 (e.g. CMD_+DONE_(\d+)).
- **default** (*int*) – Exit code to return when no code can be parsed.

Returns
Parsed integer exit code, or default if unavailable.
Return type
int

_clean_exec_output(*raw_output*, *full_cmd*, *marker_regex*)

Remove noise from raw command output and return a clean string.
Parameters
- **raw_output** (*str*) – Unfiltered text collected before the marker/prompt.
- **full_cmd** (*str*) – Exact command string that was sent (for echo removal).
- **marker_regex** – Compiled regex for the completion marker line.

Returns
Cleaned command output with noise removed.
Return type
str

_detect_test_name()

Decide a test name for logging, preferring an explicit name.
Returns
Resolved test name or "<no test>".
Return type
str

_preferred_test_name()

Return the preferred explicit test name when available.
Returns
Test name if set, else None.
Return type
Optional[str]

_first_test_func_from_stack()

Scan the call stack and return the first test_* function name.
Returns
Name of the first function starting with test_, or None if no such frame exists.
Return type
Optional[str]

_persist_cmd_output(*port*, *command*, *output*)

Append a command's output to the persistent command log file.
Parameters
- **port** (*int*) – Telnet port from which the command was executed.
- **command** (*str*) – Exact shell command that was run.
- **output** (*str*) – Final cleaned output to be written.

Returns
None
Return type
None

`_check_prompt`(*term*, *port*)

Attempt to log in on required port, then wait for its shell prompt.
Parameters
- **term** (*str*) – Terminal name.
- **port** (*int*) – Port number.

Return type
None

`_normalize_terminal`(*raw*)

Normalize a terminal key into the canonical `terminal_*` name.
Parameters
raw (*str*) – Raw terminal key from the FVP driver.
Returns
Lowercased name prefixed with `terminal_` if missing.
Return type
str

`start_telnet_sessions_after_fvp_ready`(*fvp_driver*, *log_file_path=''*)

With LocalFVP: poll get_ports() briefly so multiple terminals can appear, then spawn Telnet sessions for required terminals; if none match, fall back to all.
Parameters
- **fvp_driver** – Controller exposing terminal ports.
- **log_file_path** (*str*) – Path to FVP boot log.

Return type
None

`_resolve_log_path`(*fvp_driver*, *log_file_path*)

Resolve the boot log path using the driver if possible.
Parameters
- **fvp_driver** – FVP driver which may implement `log_path()`.
- **log_file_path** (*str*) – Fallback path to use if the driver raises or does not implement the method.

Returns
Resolved log path or "<unknown>" if none is available.
Return type
str

`_discover_ports`(*fvp_driver*)

Poll the driver for terminal→port mappings until they stabilize.

The poll runs up to `min(timeout, 15)` seconds, returning the most recent normalized mapping. It stops early if all required terminals are present.
Parameters
fvp_driver – Driver exposing `get_ports() -> Dict[str, int]`.
Returns
Dict mapping normalized terminal names to ports.

`_spawn_required_then_fallback`(*norm*)

Spawn Telnet sessions for required terminals, or all as a fallback.
Parameters
norm (*Dict[str, int]*) – Mapping of normalized terminal names to ports.

Returns
Mapping of terminal names to ports that actually started.
Return type
Dict[str, int]

`_wait_for_started_prompts`(*started*)

Wait for prompts on each started terminal. Spawns a thread per started terminal that checks the prompt, then joins all threads before returning.

Parameters
started (*Dict[str, int]*) – Mapping of terminal name to telnet port that has been started.
Returns
None
Return type
None

`close_sessions`()

Gracefully close all active telnet sessions and their log files.

Returns
None
Return type
None

`set_log_dir`(*path*)

Set the base directory for telnet logs and initialize output file.

Creates the directory if needed and ensures the command output file exists. Raises a `RuntimeError` on permission or OS errors.

Parameters
path (*str*) – New base directory for logs.
Returns
None
Raises
RuntimeError – If the directory or file cannot be created.
Return type
None

`drain_console_to_tail`(*session*, *timeout*)

Consume any pending console output to start from a clean state.

param session: The pexpect session to read from. param timeout: Maximum time to spend draining the console.

Parameters
session (*pexpect.spawn*)
Return type
None

`run_cmd`(*port*, *cmd*, *timeout=None*)

Execute a command on the target console and return its output.

Port
Telnet port number of the target console.
Parameters
- **cmd** (*str*) – Command string to execute.
- **timeout** (*Optional[int]*) – Maximum time to wait for command completion.
- **port** (*int*)

Returns
Command output with leading and trailing whitespace removed.

Raises
RuntimeError – If the command returns a non-zero status code.
Return type
str

test_automation.targets.fvp.fvp_controller

Attributes

logger	

Exceptions

FVPError	Base exception for FVP related errors.
FVPStartError	Raised when the FVP fails to start.
FVPTimeout	Raised when the FVP does not become ready within
FVPStopped	Raised when the FVP process is unexpectedly stopped.

Classes

LocalFVP	Local FVP controller to start/stop/reset an FVP process using parameters

Module Contents

test_automation.targets.fvp.fvp_controller.**logger**

exception test_automation.targets.fvp.fvp_controller.**FVPError**

Bases: RuntimeError

Base exception for FVP related errors.

exception test_automation.targets.fvp.fvp_controller.**FVPStartError**

Bases: *FVPError*

Raised when the FVP fails to start.

exception test_automation.targets.fvp.fvp_controller.**FVPTimeout**

Bases: *FVPError*

Raised when the FVP does not become ready within the expected timeout.

exception test_automation.targets.fvp.fvp_controller.**FVPStopped**

Bases: *FVPError*

Raised when the FVP process is unexpectedly stopped.

class test_automation.targets.fvp.fvp_controller.**LocalFVP**(*platform_config*, **args*, ***options*)

Bases: *test_automation.utils.device.Device*

Local FVP controller to start/stop/reset an FVP process using parameters from YAML.

Parameters
platform_config (*dict*)

_init_required_fields(*platform_config*)

Populate fields derived directly from the platform configuration.

Parameters
platform_config (*dict*) – Parsed platform configuration mapping.
Returns
None
Return type
None

_init_optional_and_telnet(*platform_config*, *args*, *options*)

Resolve optional log settings and initialize the Telnet manager.

Parameters
- **platform_config** (*dict*) – Parsed platform configuration mapping.
- **args** (*tuple*) – Legacy positional args. If present, the first element may be a TelnetSessionManager instance.
- **options** (*dict*) – Keyword options that may include: * **log_dir** (str, optional) – Override log directory path. * **log_prefix** (str, optional) – Override log filename prefix. * **telnet_log_dir** (str, optional) – Directory for Telnet logs. * **telnet_manager** (TelnetSessionManager, optional) – Telnet session manager instance. Defaults to a new instance.

Returns
None
Return type
None

_init_runtime_state()

Initialize mutable runtime state and atexit hook.

Return type
None

init()

Initialize any pre-provisioning steps, currently no-op.

Return type
None

start()

Start the FVP process with the configured parameters.

Raises
FVPStartError – If the FVP process fails to start.
Return type
None

_cleanup_threads_and_state()

Join read thread, clear flags/state.

Return type
None

_signal(*sig*)

Send a signal to the FVP process or its process group.

Parameters
sig (*int*) – Signal number (e.g. `signal.SIGTERM`).
Returns
`None`
Return type
None

`_terminate_proc()`

Terminate the FVP process normally, then forcefully if needed.
Return type
None

`stop()`

Stop the FVP process and clean up telnet sessions. Ensures the reader thread exits, the process is terminated, and all telnet sessions are closed.
Return type
None

`reset(telnet_delay_s=0)`

Restart the FVP process so the device returns to a known-good state. If it's running, stop then start; otherwise just start.
Parameters
telnet_delay_s (*int*) – Optional delay in seconds to wait
Return type
None

`wait_ready(timeout_s=None, telnet_delay_s=0)`

Block until the FVP is ready or the timeout expires.
Parameters
- **timeout_s** (*Optional[int]*) – Timeout in seconds. Defaults to configured *ready_timeout_s*.
- **telnet_delay_s** (*int*) – Optional delay in seconds to wait before starting telnet sessions after FVP is ready.

Raises
FVPTimeout – If the FVP is not ready within the timeout.
Return type
None

`is_running()`

Return True if the FVP process is currently running.
Return type
bool

`get_ports()`

Return mapping of terminal names to telnet ports.
Return type
Dict[str, int]

`log_path()`

Get the path to the current boot log file.
Raises
FVPError – If start() has not been called yet.
Return type
pathlib.Path

_build_cmd()

Construct the final command string for launching the FVP.

Return type
str

_make_log_path()

Return type
pathlib.Path

_process_stdout_line(*line*, *pattern*, *first_port_seen*)

Process one line of FVP stdout, update ports, and set readiness state.

Parameters
- **line** (*str*) – A single line from the FVP process stdout.
- **pattern** (*re.Pattern[str]*) – Compiled regex to match terminal/port info.
- **first_port_seen** (*bool*) – Whether a port was already discovered before.

Returns
True if this line contained the first discovered port, otherwise the prior first_port_seen value.

Return type
bool

_read_loop()

Read process output, write to boot log, and capture telnet port mappings. Sets ready after the first port is discovered.

Return type
None

_atexit()

Ensure the FVP process and sessions are stopped on exit.

Return type
None

test_automation.targets.fvp.plugin

Attributes

logger

Functions

_login_primary(mgr, plat)	Log in on the primary console using the session manager's prompt maps.
_export_env(pytestconfig, plat)	Optionally export environment variables for FVP execution.
_find_crypto_lib(base_dir)	Search recursively for Crypto.so under the model base directory.
make_fvp_bundle(platform_dict, cfg)	Build and return a driver instance for the fvp platform.

Module Contents

test_automation.targets.fvp.plugin.**logger**

test_automation.targets.fvp.plugin.**_login_primary**(*mgr*, *plat*)

Log in on the primary console using the session manager's prompt maps.

Parameters

- **mgr** (test_automation.targets.fvp.autofvpnetworking.TelnetSessionManager) – Active TelnetSessionManager that manages console sessions.
- **plat** (*dict*) – Platform configuration mapping.

Raises

AssertionError – If the expected prompts are not observed on the primary console during login.

Returns

None.

Return type

None

test_automation.targets.fvp.plugin.**_export_env**(*pytestconfig*, *plat*)

Optionally export environment variables for FVP execution.

Sets FVP_BINARY from --fvp-binary or existing environment, and attempts to locate Crypto.so under the model base directory. If the provided binary path does not exist, the process exits.

Parameters

- **pytestconfig** – Pytest configuration object used to read command-line options.
- **plat** (*dict*) – Platform configuration mapping (context only; not modified).

Returns

None.

Raises

SystemExit – If an invalid FVP binary path is supplied.

Return type

None

test_automation.targets.fvp.plugin.**_find_crypto_lib**(*base_dir*)

Search recursively for Crypto.so under the model base directory.

Parameters

base_dir (*pathlib.Path*)

Return type

str

test_automation.targets.fvp.plugin.**make_fvp_bundle**(*platform_dict*, *cfg*)

Build and return a driver instance for the fvp platform.

Parameters

- **platform_dict** – Platform configuration mapping as loaded from YAML.
- **cfg** – Global configuration object and forwarded to the telnet session manager for context.

Returns
A `DriverBundle`.

Return type
test_automation.targets.registry.DriverBundle

test_automation.targets.registry

Attributes

logger	
PLATFORM_REGISTRY	

Classes

DriverBundle	Container for a platform instance.

Functions

register_platform(kind)	Decorator to register a factory for platform under the given kind.
get_factory(kind)	Fetch a registered platform factory by kind.

Module Contents

`test_automation.targets.registry.`**`logger`**

`test_automation.targets.registry.`**`PLATFORM_REGISTRY`**

`test_automation.targets.registry.`**`register_platform`**(*kind*)

Decorator to register a factory for platform under the given kind.

Parameters
kind – Platform kind (e.g., `"fvp"`, `"fpga"`).

Returns
Decorator that registers the factory and returns it unchanged.

`test_automation.targets.registry.`**`get_factory`**(*kind*)

Fetch a registered platform factory by kind.

Parameters
kind – Platform kind (case-insensitive).

Returns
Registered factory callable, or `None` if not found.

`class test_automation.targets.registry.`**`DriverBundle`**(*driver*, *manager=None*, *login_primary=None*, *export_env=None*)

Container for a platform instance.

Parameters

- **driver** – Platform driver instance with `start()`, `wait_ready()`, `stop()` (and optionally `init()`).
- **manager** – Optional session/console manager (e.g., telnet, ssh, serial).
- **login_primary** – Optional callable for logging into the primary.
- **export_env** – Optional callable for exporting environment variables.

Variables

- **driver** – Platform driver instance.
- **manager** – Session/console manager object.
- **login_primary** – Callable for primary console login.
- **export_env** – Callable for exporting environment variables.

driver

manager = None

login_primary = None

export_env = None

test_automation.utils

Submodules

test_automation.utils.auto_platform_base

Attributes

logger	

Classes

AutoTestPlatformBase	Base helpers and common attributes for automated platform tests.

Module Contents

test_automation.utils.auto_platform_base.**logger**

class test_automation.utils.auto_platform_base.**AutoTestPlatformBase**(*telnet_mgr*, *platform_config*, *cli_platform*)

Base helpers and common attributes for automated platform tests.

Variables

- **mgr** – Telnet/session manager used to resolve console ports.

- **platform** – Platform name (lowercased) from config.
- **cli_platform** – Platform name provided via CLI (lowercased).
- **target** – Target kind from config.
- **config_data** – Raw platform configuration dictionary.
- **rse_port** – Port handle for the RSE console.
- **scp_port** – Port handle for the SCP console.
- **default_console** – Port handle for the default/primary console.
- **secure_world_ap_console** – Port handle for the Secure World AP console.
- **si_cluster0** – Port handle for Safety Island cluster 0.
- **si_cluster1** – Port handle for Safety Island cluster 1.
- **si_cluster2** – Port handle for Safety Island cluster 2.

Parameters

- **platform_config** (*dict*)
- **cli_platform** (*str*)

mgr

platform

cli_platform

target = ''

config_data

rse_port

scp_port

default_console

secure_world_ap_console

test_automation.utils.device

Classes

Device	Abstract device manager API required for both local and

Module Contents

class test_automation.utils.device.**Device**

Bases: abc.ABC

Abstract device manager API required for both local and hardware farm device connections.

Returns

None.

abstract init()

Prepare resources but do not power on/run yet.

Returns
None.

Return type
None

abstract start()

Power on the DUT.

Returns
None.

Return type
None

abstract stop()

Power off the DUT and clean up the resources.

Returns
None.

Return type
None

abstract reset()

Power cycle DUT. Power on and off as per the current state.

Returns
None.

Return type
None

abstract wait_ready(*timeout_s=None*)

Block until the DUT is ready (e.g., exported telnet ports detected).

Returns
None.

Parameters
timeout_s (*Optional[int]*)

Return type
None

abstract is_running()

True if the model is alive.

Returns
`True` if the device is running, else `False`.

Return type
bool

abstract get_ports()

Return discovered terminal name -> telnet port mappings.

Returns
Dictionary mapping terminal names to telnet port numbers.

Return type
Dict[str, int]

abstract log_path()

Return path to the primary boot log.

Returns
Path object pointing to the boot log file.

Return type
pathlib.Path

test_automation.utils.io_utils

Attributes

logger	
DEFAULT_TIMEOUT	

Functions

write_file(platform_base_obj, path, value[, timeout])	Write a value to a file using shell redirection.
read_file(platform_base_obj, path[, timeout])	Read the contents of a file using the `cat` command.
write_to_port(mgr, port, path, value)	Write value to file over a specific port.
read_file_from_port(mgr, port, path[, timeout])	Read file content using cat command over a specific port.
check_if_file_exists(mgr, port, path)	Check if a file exists on the target.
read_int(mgr, port, path)	Read integer value from file using cat command.

Module Contents

`test_automation.utils.io_utils.`**`logger`**

`test_automation.utils.io_utils.`**`DEFAULT_TIMEOUT = 120`**

`test_automation.utils.io_utils.`**`write_file`**(*platform_base_obj*, *path*, *value*, *timeout=DEFAULT_TIMEOUT*)

Write a value to a file using shell redirection.

Parameters

- **`platform_base_obj`** – Platform base object providing console access.
- **`path`** (*str*) – Path to the file to write to.
- **`value`** (*Union[str, int]*) – Value to write to the file.
- **`timeout`** (*int*) – Command timeout in seconds.

Returns
output of the command execution

`test_automation.utils.io_utils.`**`read_file`**(*platform_base_obj*, *path*, *timeout=DEFAULT_TIMEOUT*)

Read the contents of a file using the `cat` command.

Parameters

- **`platform_base_obj`** – Platform base object providing console access.
- **`path`** (*str*) – Path to the file to read from.

- **timeout** (*int*) – Command timeout in seconds.

Returns
Contents of the file

Return type
str

test_automation.utils.io_utils.**write_to_port**(*mgr*, *port*, *path*, *value*)

Write value to file over a specific port.

Parameters

- **mgr** – Manager object to run command
- **port** – Console port to use
- **path** (*str*) – File path to write to
- **value** (*Union[str, int]*) – Value to write

Returns
output of the command execution

test_automation.utils.io_utils.**read_file_from_port**(*mgr*, *port*, *path*, *timeout=120*)

Read file content using cat command over a specific port.

Parameters

- **mgr** – Manager object to run command
- **port** – Console port to use
- **path** (*str*) – File path to read from
- **timeout** (*int*) – Command timeout in seconds

Returns
Contents of the file

Return type
str

test_automation.utils.io_utils.**check_if_file_exists**(*mgr*, *port*, *path*)

Check if a file exists on the target.

Parameters

- **mgr** – Manager object to run command
- **port** – Console port to use
- **path** (*str*) – File path to read from

Returns
True if file exists, False otherwise

Return type
bool

test_automation.utils.io_utils.**read_int**(*mgr*, *port*, *path*)

Read integer value from file using cat command.

Parameters

- **mgr** – Manager object to run command

- **port** – Console port to use
- **path** (*str*) – File path to read from

Returns
Integer value read from the file

Return type
int

test_automation.utils.logfiltering

Module to remove all extra and unnecessary characters from log files.

Attributes

ANSI_ESCAPE	
ORPHAN_CSI	
CONTROL_CHARS	
_CLEANER	

Classes

AnsiStrippingStream	A wrapper that strips ANSI/control sequences before writing.

Functions

strip_ansi_and_controls(text)	Removes ANSI escape codes, orphan CSI sequences, and stray control

Module Contents

test_automation.utils.logfiltering.**ANSI_ESCAPE**

test_automation.utils.logfiltering.**ORPHAN_CSI**

test_automation.utils.logfiltering.**CONTROL_CHARS**

test_automation.utils.logfiltering.**_CLEANER**

test_automation.utils.logfiltering.**strip_ansi_and_controls**(*text*)

Removes ANSI escape codes, orphan CSI sequences, and stray control characters.

Parameters
text (*str*) – Raw string to clean.

Returns
Cleaned string without ANSI/control sequences.

Return type
str

class test_automation.utils.logfiltering.**AnsiStrippingStream**(*underlying*)

A wrapper that strips ANSI/control sequences before writing.

This is typically used to wrap a file handle so that all log writes are cleaned of terminal formatting characters automatically.

Parameters
underlying (*TextIO*)

_stream

write(*data*)

Strip ANSI/control sequences from *data* and write to the underlying stream.

Parameters
data (*str*) – Raw string data to sanitize and write.
Returns
Number of characters written.
Return type
int

flush()

Flush the underlying stream.

Return type
None

test_automation.utils.networking_base

Abstract base for networking/session managers used by targets.

This is intentionally minimal so that existing FVP (TelnetSessionManager) and new FPGA (SSHSessionManager) can adopt it with very small changes.

Classes

BaseNetworkManager	Base interface for target networking/session managers.

Module Contents

class test_automation.utils.networking_base.**BaseNetworkManager**

Bases: abc.ABC

Base interface for target networking/session managers.

abstract connect()

Establish underlying connections/sessions.

Return type
None

abstract disconnect()

Tear down all underlying sessions, connections and file handles.

Return type
None

set_log_dir(*path*)

Optional hook to change the base directory used for logs.

Parameters
path (*str*)
Return type
None

abstract execute_simple_command(*command*, *options=None*, ***kwargs*)

Optional generic "run one shell command" hook.

Subclasses may override this to run a command and return its exit code. Provide structured settings via `options`; `**kwargs` is available for backward-compatible extras.

Parameters
- **command** (*str*) – Shell command to execute.
- **options** (*Optional[types.SimpleNamespace]*) – Optional settings namespace.
- **kwargs** – Legacy/extra keyword arguments accepted for backward compatibility.

Returns
Command exit code.
Raises
NotImplementedError – If not implemented by subclass.
Return type
int

test_automation.utils.utils

Functions

find_project_root([filename])	Locate the root directory by searching through directory that contains

Module Contents

test_automation.utils.utils.**find_project_root**(*filename='pyproject.toml'*)

Locate the root directory by searching through directory that contains `filename`.

Parameters
filename – Marker file to look for when determining the project root.

Returns
Absolute path to the directory containing the marker file.

test_automation.version

Functions

get_version()	Resolve the installed package version.

Module Contents

test_automation.version.**get_version**()

Resolve the installed package version.

Returns
Resolved version string, or "0.0.0.dev0" if unavailable.

Return type
str

tests

Submodules

tests.conftest

Attributes

logger

Functions

_compute_run_paths(config)	Create a unique run directory and return run metadata dict.
_load_target_plugin(config)	Dynamically import target plugin (e.g., for FVP) if selected.
_apply_debug_logging(config)	Enable CLI logging if --debug-logs was passed.
_prepare_bundle_paths(pytestconfig, bundle)	Ensure bundle paths (manager + driver) are set up; return run paths dict.
_power_on(bundle, ptype, plat)	Bring the platform up and wait until ready.
_cleanup_manager(mgr)	Best-effort session cleanup on the manager.
_power_off(bundle)	Attempt graceful power-off and session cleanup.
pytest_configure(config)	Configure pytest logging and conditionally load target plugins.
cfg(pytestconfig)	Load the YAML configuration once per test session.
_available_platform_names(cfg)	Return a list of platform 'name' values from cfg.platforms, or an empty
_apply_cfg2_overrides(platform_config)	Temporarily inject cfg2 specific configs before boot.
auto_platform_config_data(pytestconfig, cfg)	Resolve and return the selected platform dictionary from the YAML config.
platform_bundle(pytestconfig, cfg, ...)	Build the platform bundle once from the config.
platform_driver(platform_bundle)	Provide the platform driver object for tests.
session_manager(platform_bundle)	Provide the session/console manager for tests.
platform_name(platform_base_obj, cfg)	Normalize the selected platform name to a canonical short name.
platform_base_obj(session_manager, ...)	Construct the base platform helper object once per session.
fpga_device(request)	Session-scoped FPGA device fixture.
pytest_addoption(parser)	Register custom CLI options used by the test framework.
_get_cfg2_si_cluster1_prompt()	Return the expected prompt for si_cluster1 based on build type.

Module Contents

tests.conftest.**logger**

tests.conftest.**_compute_run_paths**(*config*)

Create a unique run directory and return run metadata dict.

Parameters

- **pytestconfig** – Pytest configuration object.
- **config** (*pytest.Config*)

Returns

Dictionary.

Return type

dict

tests.conftest.**_load_target_plugin**(*config*)

Dynamically import target plugin (e.g., for FVP) if selected.

Parameters

- **pytestconfig** – Pytest configuration object.
- **config** (*pytest.Config*)

Returns
None.

Return type
None

tests.conftest.**_apply_debug_logging**(*config*)

Enable CLI logging if –debug-logs was passed.

Parameters

- **pytestconfig** – Pytest configuration object.
- **config** (*pytest.Config*)

Returns
None.

Return type
None

tests.conftest.**_prepare_bundle_paths**(*pytestconfig*, *bundle*)

Ensure bundle paths (manager + driver) are set up; return run paths dict.

If pytest_configure already created run paths, reuse them; otherwise, compute a fallback here.

Parameters

- **pytestconfig** (*pytest.Config*)
- **bundle** (test_automation.targets.registry.DriverBundle)

Return type
dict

tests.conftest.**_power_on**(*bundle*, *ptype*, *plat*)

Bring the platform up and wait until ready.

Parameters

- **bundle** (test_automation.targets.registry.DriverBundle)
- **ptype** (*str*)
- **plat** (*dict*)

Return type
None

tests.conftest.**_cleanup_manager**(*mgr*)

Best-effort session cleanup on the manager.

Return type
None

tests.conftest.**_power_off**(*bundle*)

Attempt graceful power-off and session cleanup.

Parameters
bundle (test_automation.targets.registry.DriverBundle)

Return type
None

tests.conftest.**pytest_configure**(*config*)

Configure pytest logging and conditionally load target plugins.

Parameters
config (*pytest.Config*) – Pytest configuration object.

Returns
None.

Return type
None

tests.conftest.**cfg**(*pytestconfig*)

Load the YAML configuration once per test session.

Parameters
pytestconfig (*pytest.Config*) – Pytest configuration object.

Returns
Parsed `Config` instance.

Return type
test_automation.configs.config.Config

tests.conftest.**_available_platform_names**(*cfg*)

Return a list of platform 'name' values from cfg.platforms, or an empty list if cfg.platforms is not present.

Parameters
cfg (test_automation.configs.config.Config) – Parsed configuration object.

Returns
List of platform names defined in the configuration.

Return type
list

tests.conftest.**_apply_cfg2_overrides**(*platform_config*)

Temporarily inject cfg2 specific configs before boot.

Parameters
platform_config (*dict*)

Return type
None

tests.conftest.**auto_platform_config_data**(*pytestconfig*, *cfg*)

Resolve and return the selected platform dictionary from the YAML config.

Parameters

- **pytestconfig** (*pytest.Config*) – Pytest configuration object.
- **cfg** (test_automation.configs.config.Config) – Parsed configuration object.

Returns
Platform configuration dictionary.

Return type
dict

tests.conftest.**platform_bundle**(*pytestconfig*, *cfg*, *auto_platform_config_data*)

Build the platform bundle once from the config.

Parameters

- **pytestconfig** – Pytest configuration object.
- **cfg** (test_automation.configs.config.Config) – Parsed configuration object.
- **auto_platform_config_data** – Platform configuration dictionary.

Returns
Yields an initialized DriverBundle.

tests.conftest.**platform_driver**(*platform_bundle*)

Provide the platform driver object for tests.

Parameters
platform_bundle – Initialized driver bundle.

Returns
Driver instance from the bundle.

tests.conftest.**session_manager**(*platform_bundle*)

Provide the session/console manager for tests.

Parameters
platform_bundle – Initialized driver bundle.

Returns
Session manager object.

tests.conftest.**platform_name**(*platform_base_obj*, *cfg*)

Normalize the selected platform name to a canonical short name.

Parameters

- **platform_base_obj** – Initialized AutoTestPlatformBase.
- **cfg** (test_automation.configs.config.Config) – Parsed configuration object.

Returns
Canonical name like 'aspne'.

Return type
str

tests.conftest.**platform_base_obj**(*session_manager*, *auto_platform_config_data*, *pytestconfig*)

Construct the base platform helper object once per session.

Parameters

- **session_manager** – Session/console manager instance.
- **auto_platform_config_data** – Platform configuration dictionary.
- **pytestconfig** – Pytest configuration object.

Returns
Initialized AutoTestPlatformBase instance.

Return type
test_automation.utils.auto_platform_base.AutoTestPlatformBase

tests.conftest.**fpga_device**(*request*)

Session-scoped FPGA device fixture.

- Initializes and starts the FPGA platform
- Waits for ready state
- Starts UART keepalive
- Ensures graceful shutdown and log collection

tests.conftest.**pytest_addoption**(*parser*)

Register custom CLI options used by the test framework.

Parameters
parser – Pytest argument parser.

Returns
None.

Return type
None

tests.conftest.**_get_cfg2_si_cluster1_prompt**()

Return the expected prompt for si_cluster1 based on build type.

Return type
str | None

tests.utils

Submodules

tests.utils.common_utils

Classes

CommonUtils	Common utilities for test automation.

Module Contents

class tests.utils.common_utils.**CommonUtils**

Common utilities for test automation.

static get_logger(*name*)

Return a logger configured with the common format.
Parameters
name (*str*) – Name of the logger (usually __name__)
Returns
Configured logger instance
Return type
logging.Logger

static require_supported_platform(*platform_name*, *platform_base_obj*, *supported_platforms*, *suite_name*)

Skip the test if the platform is not in the supported list.

Parameters

- **platform_name** (*str*) – Name of the platform being tested
- **platform_base_obj** – Base object for the platform
- **supported_platforms** (*Iterable[str]*) – Iterable of supported platform names
- **suite_name** (*str*) – Name of the test suite for logging

Return type

None

tests.utils.cpu_utils

Classes

CpuUtils	Utility class for CPU-related operations.

Module Contents

class tests.utils.cpu_utils.**CpuUtils**

Utility class for CPU-related operations.

static get_clusters_and_cores(*cpu_count*, *cores_per_cluster*)

Return a mapping of cluster IDs to lists of core IDs.

Parameters

- **cpu_count** (*int*) – Total number of CPUs.
- **cores_per_cluster** (*int*) – Number of cores per cluster.

Returns

Mapping `cluster_id -> [core_id, ...]`.

Return type

Dict[int, List[int]]

static get_cpu_index(*cluster_id*, *core_id*, *cores_per_cluster*)

Compute a global CPU index from cluster and core IDs.

Parameters

- **cluster_id** (*int*) – Cluster identifier.
- **core_id** (*int*) – Core identifier within the cluster.
- **cores_per_cluster** (*int*) – Number of cores per cluster.

Returns

Global CPU index.

Return type

int

_list_state_dirs(*mgr*, *port*, *cpu*, *cpu_sysfs*)

List cpuidle state directories for a given CPU.

Parameters

- **cpu** (*int*) – CPU index to list states for.
- **cpu_sysfs** (*str*) – Base sysfs path for CPU information.

Returns

Mapping of state names to their sysfs paths.

Return type

Dict[str, str]

_load_states_per_cpu(*platform_base_obj*, *names*, *cpu_sysfs*)

Load cpuidle states for each CPU.

Parameters

- **platform_base_obj** – Object containing configuration and manager.
- **names** – List of state names to load.
- **cpu_sysfs** – Base sysfs path for CPU information.

Returns

Mapping of CPU indices to their cpuidle states.

Return type

Dict[int, Dict[str, str]]

_check_default_status_for_state(*mgr*, *port*, *cpu*, *state_id*, *states*)

Check the default status for a given cpuidle state on a specific CPU.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **cpu** (*int*) – CPU index to check.
- **state_id** (*str*) – Identifier of the cpuidle state to check.
- **states** (*Dict[str, str]*) – Mapping of state names to their sysfs paths for the CPU.

_validate_required_files(*mgr*, *port*, **paths*)

Validate that required cpuidle files exist.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **paths** (*str*) – List of file paths to validate.

Returns

None

Return type

None

_verify_usage_increases(*mgr*, *port*, *usage_path*, *cpu*, *state_id*, *time_short=60*)

Verify that the usage count increases over a short period.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **usage_path** (*str*) – Path to the usage file for the state.
- **cpu** (*int*) – CPU index to check.
- **state_id** (*str*) – Identifier of the cpuidle state to check.
- **time_short** (*int*) – Time to wait between checks in seconds.

Returns

None

_disable_state(*mgr*, *port*, *disable_path*, *cpu*, *state_id*, *original_disable_values*)

Disable a specific cpuidle state for a given CPU.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **disable_path** (*str*) – Path to the disable file for the state.
- **cpu** (*int*) – CPU index to disable the state on.
- **state_id** (*str*) – Identifier of the cpuidle state to disable.
- **original_disable_values** (*Dict[int, Dict[str, str]]*) – Dictionary to store original disable values for restoration.

_verify_usage_stays_same_when_disabled(*mgr*, *port*, *usage_path*, *cpu*, *state_id*, *time_short=60*)

Verify that the usage count does not change when the state is disabled.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **usage_path** (*str*) – Path to the usage file for the state.
- **cpu** (*int*) – CPU index to check.
- **state_id** (*str*) – Identifier of the cpuidle state to check.
- **time_short** (*int*) – Time to wait between checks in seconds.

_verify_latency_residency_values(*mgr*, *port*, *base*, *props*, *cpu*, *state_id*)

Verify latency and residency values for a specific C-state on a CPU.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **base** (*str*) – Base sysfs path for the C-state.
- **props** (*Dict[str, int]*) – Expected properties containing 'latency' and 'residency'.
- **cpu** (*int*) – CPU index to check.
- **state_id** (*str*) – Identifier of the cpuidle state to check.

_verify_usage_time_advancement(*mgr*, *port*, *base*, *cpu*, *state_id*, *time_long=120*)

Verify that usage and time values advance over a longer period.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **base** (*str*) – Base sysfs path for the C-state.
- **cpu** (*int*) – CPU index to check.
- **state_id** (*str*) – Identifier of the cpuidle state to check.
- **time_long** (*int*) – Time to wait between checks in seconds.

available_governors(*mgr*, *port*, *cpuidle_dir*)

List available cpuidle governors.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **cpuidle_dir** – Base directory for cpuidle sysfs entries.

Returns

List of available cpuidle governors.

Return type

List[str]

current_governor(*mgr*, *port*, *cpuidle_dir*, *ro=True*)

Get the current cpuidle governor.

Parameters

- **mgr** – Manager object for executing commands.
- **port** – Port identifier for command execution.
- **cpuidle_dir** – Base directory for cpuidle sysfs entries.
- **ro** (*bool*) – Whether to read from the read-only current_governor_ro file.

Returns

Name of the current cpuidle governor.

Return type

str

set_cpu(*platform_base_obj*, *cpu_num*, *flag*)

Set a CPU core online/offline and verify the new state.

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **cpu_num** (*int*) – CPU index to modify.
- **flag** (*str*) – Desired CPU online state as string.

Returns

True when the value matches flag otherwise False

Return type

bool

enable_cpu(*platform_base_obj*, *cpu_num*)

Enable a specific CPU core.

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase)
- **cpu_num** (*int*)

Return type

bool

disable_cpu(*platform_base_obj*, *cpu_num*)

Disable a specific CPU core.

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase)
- **cpu_num** (*int*)

Return type

bool

validate_cpu_count_from_devicetree(*platform_base_obj*, *expected_num_cpus*)

Validate CPU count in device tree against expected value.

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **expected_num_cpus** (*int*) – Expected number of CPU nodes in device tree.

Raises

AssertionError – If CPU count does not match expected value.

Return type

None

assert_all_cores_online(*platform_base_obj*, *expected_num_cpus*)

Check that all expected CPU cores are currently online.

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **expected_num_cpus** (*int*) – Expected number of online processors.

Raises

AssertionError – If online CPU count differs from expected

Return type

None

stop_individual_core(*platform_base_obj*, *expected_num_cpus*)

Verify per-core CPU hotplug by toggling each core off then on.

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **expected_num_cpus** (*int*) – Expected number of online processors.

Raises

AssertionError – If online CPU count differs from expected

Return type

None

reenable_all_cpus(*platform_base_obj*, *expected_num_cpus*)

Attempt to re-enable all CPU cores and assert recovery

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **expected_num_cpus** (*int*) – number of CPU cores expected to be online.

Raises

AssertionError – If one or more CPU cores cannot be re-enabled.

Return type

None

check_cannot_disable_all_cores(*platform_base_obj*, *expected_num_cpus*)

Verify the system refuses disabling the final online CPU core.

Parameters

- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **expected_num_cpus** (*int*) – Total number of CPU cores in the system.

Raises

AssertionError – last core disable unexpectedly succeeds.

Return type

None

tests.utils.crypto_extension_utils

Classes

CryptographicExtensionUtils	Utility functions for Arm Cryptographic Extension tests.

Module Contents

class tests.utils.crypto_extension_utils.**CryptographicExtensionUtils**

Utility functions for Arm Cryptographic Extension tests. Provides helper methods to:

- Generate self-signed certificates
- Start and stop SSL servers
- Perform HTTPS downloads with timing

_generate_certificate(*mgr*, *port*, *days*)

Generate a self-signed certificate using OpenSSL

Parameters

- **mgr** – Manager object to execute commands
- **port** – Console port to run commands on
- **days** – Number of days the certificate is valid

Returns
None
Return type
None

`_start_ssl_server`(*mgr*, *port*)

Start SSL server serving 100MB random data file
Parameters
- **mgr** – Manager object to execute commands
- **port** – Console port to run commands on

Returns
None
Return type
None

`_cleanup_server`(*mgr*, *port*)

Cleanup the server process by bringing it to foreground and terminating
Parameters
- **mgr** – Manager object to execute commands
- **port** – Console port to run commands on

Returns
None
Return type
None

`_cleanup_files`(*mgr*, *port*)

Cleanup generated certificate files
Parameters
- **mgr** – Manager object to execute commands
- **port** – Console port to run commands on

Returns
None
Return type
None

`_extract_time_fields`(*output*)

Extract timing information from the 'time' command output. Returns a dictionary with 'real', 'user', and 'sys' timing values.
Parameters
output (*str*) – The output string from the 'time' command
Returns
A dictionary with keys 'real', 'user', 'sys' and their corresponding timing values in seconds
Return type
dict

`_download`(*mgr*, *port*, *enable_crypto*, *timeout=60*)

Download data from the SSL server using openssl s_client with specified cipher and return timing information.
Parameters
- **mgr** – Manager object to execute commands
- **port** – Console port to run commands on
- **enable_crypto** (*bool*) – Flag to enable or disable cryptographic operations
- **timeout** (*int*) – Timeout for the download command

Returns

A dictionary with keys 'real', 'user', 'sys' and their corresponding timing values in seconds

Return type
dict

tests.utils.fvp_utils

Attributes

logger

Classes

FvpUtils

Module Contents

tests.utils.fvp_utils.**logger**

class tests.utils.fvp_utils.**FvpUtils**

cpu_utils

check_rng(*platform_base_obj*, *hw_random*, *dev*)

Validate RNG sysfs entry contains the expected device identifier.

Parameters
- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **hw_random** (*str*) – Path to RNG sysfs node to inspect.
- **dev** (*str*) – Expected RNG device name.

Raises
RuntimeError – when command execution fails on the target.

Return type
None

check_devices(*platform_base_obj*, *cls*, *min_count*, *search_drivers*)

Validate device class population and expected driver bindings.

Parameters
- **platform_base_obj** (test_automation.utils.auto_platform_base.AutoTestPlatformBase) – Platform fixture with console access.
- **cls** (*str*) – Sysfs device class name.
- **min_count** (*int*) – Minimum number of class devices expected.
- **search_drivers** (*list[str]*) – acceptable driver names for the class devices.

Raises
AssertionError – If device count is below minimum or no expected driver is found.

Return type
None

tests.utils.fwu_utils

Attributes

logger	

Functions

check_block_device(platform_base_obj, dev)	Verify that a block device exists on target.
check_mount(platform_base_obj, target, source, error_msg)	Verify that a mount exists using findmnt linux command
check_file(platform_base_obj, path, error_msg)	Verify that a file exists on target.
prepare_capsule_environment(platform_base	Mount boot, verify devices, check capsule, and copy file
check_system_status(platform_base_obj[, timeout])	Verify that systemd reaches `running` state after FWU flow.

Module Contents

tests.utils.fwu_utils.**logger**

tests.utils.fwu_utils.**check_block_device**(*platform_base_obj*, *dev*)

Verify that a block device exists on target.

Parameters

- **platform_base_obj** – Platform fixture containing manager
- **dev** (*str*) – Device path

Returns
None

Return type
None

tests.utils.fwu_utils.**check_mount**(*platform_base_obj*, *target*, *source*, *error_msg*)

Verify that a mount exists using findmnt linux command

Parameters

- **platform_base_obj** – Platform fixture containing manager
- **target** (*str*) – target to mount the device
- **source** (*str*) – Expected device
- **error_msg** (*str*) – Msg in case of failure

Returns
None

Return type
None

tests.utils.fwu_utils.**check_file**(*platform_base_obj*, *path*, *error_msg*)

Verify that a file exists on target.

Parameters

- **platform_base_obj** – Platform fixture containing manager
- **path** (*str*) – File path
- **error_msg** (*str*) – Msg in case of failure

Returns
None

Return type
None

tests.utils.fwu_utils.**prepare_capsule_environment**(*platform_base_obj*)

Mount boot, verify devices, check capsule, and copy file

tests.utils.fwu_utils.**check_system_status**(*platform_base_obj*, *timeout=200*)

Verify that systemd reaches `running` state after FWU flow.

If running state is not reached, this helper drains pending console output, collects failed systemd units, and fails the test with diagnostics.

Parameters

- **platform_base_obj** – Platform fixture containing manager
- **timeout** (*int*) – Reserved timeout argument

Returns
None

Return type
None

tests.utils.scp_cli_utils

Classes

ScpCliUtils	SCP Debugger CLI utility class.
ScpTestUtils	Reusable SCP test helpers

Module Contents

class tests.utils.scp_cli_utils.**ScpCliUtils**

SCP Debugger CLI utility class.

Provides helpers to interact with the SCP CLI:

- Enter or exit CLI
- Capture logs between test markers

DEFAULT_TIMEOUT = 120

enter_scp_cli(*session*, *pattern*, *prompt*, *timeout*)

Send Ctrl+E and wait for the SCP CLI prompt to appear.

Parameters

- **session** (*pexpect.spawn*) – pexpect session connected to the console
- **pattern** (*str*) – Regex pattern to identify successful entry into CLI
- **prompt** (*str*) – Regex pattern to identify the CLI prompt after entry
- **timeout** (*int*) – Timeout waiting for patterns

Returns

Captured output from the CLI entry process

Raises

AssertionError – If expected patterns are not found within the timeout

Return type

str

exit_scp_cli(*session*, *pattern*, *timeout*)

Send Ctrl+D, ensure the exit banner is printed, and return it.

Parameters

- **session** (*pexpect.spawn*) – pexpect session connected to the console
- **pattern** (*str*) – Regex pattern to identify successful exit from CLI
- **timeout** (*int*) – Timeout for waiting for the exit pattern

Returns

Captured output from the CLI exit process

Raises

AssertionError – If the expected exit pattern is not found within the timeout

Return type

str

capture_integration_logs(*manager*, *session*, *test_name*, *timeout*)

Capture logs emitted between the Start/End markers of a test.

Parameters

- **manager** (*Any*) – The console manager to use for pattern matching
- **session** (*pexpect.spawn*) – pexpect session connected to the console
- **test_name** (*str*) – The name of the test, used to identify log markers
- **timeout** (*int*) – Timeout for waiting for log markers

Returns

Captured logs between the Start and End markers

Raises

AssertionError – If the expected log markers are not found within the timeout

Return type

str

_ensure_str(*data*)

Ensure that the given data is returned as a string.

If the input is of type `bytes`, it is then decoded using UTF-8 with errors ignored. For all other types, the value is converted using `str()`. :param data: Value to normalize into a string representation.

Parameters

data (*object*)

Return type

str

class tests.utils.scp_cli_utils.**ScpTestUtils**

Reusable SCP test helpers

DEFAULT_TIMEOUT = 120

TEST_RESULT_RE =
'(?P<file>.+?):(?P<line>\\d+):(?P<name>[^:]+):(?P<status>PASS|FAIL|IGNORE)'

_build_test_patterns(*test_name*)

Build regular expression patterns for integration test.

The provided test name is escaped to ensure safe inclusion in a regular expression. The returned patterns match log lines of the form:

```
[INTEGRATION_TEST] Start: <test_name>
[INTEGRATION_TEST] End: <test_name>
```

Parameters
test_name (*str*) – Name of the integration test.
Returns
Tuple containing the start and end regex patterns.
Return type
tuple[str, str]

_wait_for_pattern(*session*, *pattern*, *timeout*, *manager*)

Wait for a specific pattern to appear in the console session.

If a `manager` object providing an `_expect_pattern` method is supplied, it is used for pattern matching. Otherwise, the method falls back to direct `pexpect` matching. :param session: Active pexpect session connected to the console. :param pattern: Regular expression pattern to wait for. :param timeout: Maximum time in seconds to wait for the pattern. :param manager: Optional console manager implementing `_expect_pattern`. :raises AssertionError: If the expected pattern is not matched within given timeout when using the manager. :raises pexpect.TIMEOUT: If direct `pexpect` matching times out. :raises pexpect.EOF: If the session ends unexpectedly.

Parameters
- **session** (*pexpect.spawn*)
- **pattern** (*str*)
- **timeout** (*int*)
- **manager** (*Any*)

Return type
None

_normalize_output(*data*)

Normalize raw console output into a clean string.

This helper ensures that data returned from pexpect or other subprocess interactions is consistently represented as a string. If the input is of type *bytes*, it is decoded using UTF-8 with errors ignored. The result is then converted to a string (if not already) and stripped of leading and trailing whitespace.

Parameters
data (*Any*) – Raw console output to normalize.
Returns
Cleaned string representation of the input.
Return type
str

capture_logs(*session*, *test_name*, *timeout=None*, *manager=None*)

Capture integration test logs between Start/End markers.

This method waits for the integration test start and end markers corresponding to the given `test_name` and extracts all console output emitted between those markers.

The method supports both direct `pexpect` pattern matching and manager-assisted matching (when a console manager providing `_expect_pattern` is supplied). :param session: Active pexpect session connected to the console. :param test_name: Name of the integration test. :param timeout: Optional timeout in seconds. If not provided, `DEFAULT_TIMEOUT` is used. :param manager: Optional console manager providing `_expect_pattern` for matching. :returns: Captured console output between test start and end markers. :raises AssertionError: If the expected start or end markers are not detected within the timeout.

Parameters

- **session** (*pexpect.spawn*)
- **test_name** (*str*)
- **timeout** (*Optional[int]*)
- **manager** (*Any*)

Return type

str

_compute_width(*runs*)

Compute the display width required for the RUN column in the summary.

The width is determined by the longest run name in the provided list, A minimum width of 12 character is enforced to keep table formatting readable. :param runs: List of per-run summary dictionaries containing at least a `"name"` key. :returns: Calculated column width for the RUN field.

Parameters

runs (*List[Dict[str, Any]]*)

Return type

int

_compute_totals(*runs*)

Compute aggregated totals across multiple test runs.

This method sums the total number of executed, passed, failed, and ignored tests from the provided run summaries and determines an overall status. :param runs: List of per-run summary dictionaries containing the keys `"total"`, `"passed"`, `"failures"`, `"ignored"`, and optionally `"ok"`. :returns: Dictionary containing aggregated totals and overall status. Keys include `"total"`, `"passed"`, `"failures"`, `"ignored"`, and `"status"`.

Parameters

runs (*List[Dict[str, Any]]*)

Return type

Dict[str, Any]

_build_header(*title*, *meta*)

Build the formatted header section for the combined summary output.

The header includes: - A top separator line - The provided title - Optional metadata fields (platform, port, suite) - A separator line below the metadata :param title: Title displayed at the top of the summary. :param meta: Dictionary containing optional metadata fields. :returns: List of formatted header lines.

Parameters

- **title** (*str*)
- **meta** (*Dict[str, Any]*)

Return type

List[str]

_build_table(*runs*, *totals*, *width_name*)

Build the formatted summary table section.

This method generates a fixed-width, human-readable table showing: - Per-run statistics (total, passed, failed, ignored, result) - An aggregated "OVERALL" summary row

The table layout is aligned using the provided column width for the RUN column to ensure consistent formatting. :param runs: List of per-run summary dictionaries. Each dictionary must contain the keys `name`, `total`, `passed`, `failures`, `ignored`, and `status`. :param totals: Dictionary containing aggregated totals and overall status values. :param width_name: Width used to left-align the `RUN` column. :returns: List of formatted table lines.

Parameters
- **runs** (*List[Dict[str, Any]]*)
- **totals** (*Dict[str, Any]*)
- **width_name** (*int*)

Return type
List[str]

_build_footer(*totals*, *_width_name*)

Build the footer section of the combined summary output.

The footer displays the overall aggregated result, including total, passed, failed, and ignored counts, along with the computed overall status. It is visually separated from the table using delimiter lines. :param totals: Dictionary containing aggregated totals and overall status values. Expected keys include `total`, `passed`, `failures`, `ignored`, and `status`. :param _width_name: Unused width parameter included for interface consistency with other builder methods. :returns: List of formatted footer lines.

Parameters
- **totals** (*Dict[str, Any]*)
- **_width_name** (*int*)

Return type
List[str]

_build_appendix(*runs*, *overall_status*)

Build an optional appendix section listing failed or ignored tests.

The appendix is included only when:
- The overall status is not "PASS", or
- At least one run contains failed or ignored test entries.

For each run with issues, the appendix lists the specific failed and/or ignored test names to provide additional detail.

Parameters
- **runs** (*List[Dict[str, Any]]*) – List of per-run summary dictionaries. Each dictionary may contain `failed_tests` and `ignored_tests` keys.
- **overall_status** (*str*) – The computed overall test status (typically `"PASS"` or `"FAIL"`).

Returns
List of formatted appendix lines. Returns an empty list if the appendix should be omitted.

Return type
List[str]

_should_skip_appendix(*runs*, *overall_status*)

Determine whether the appendix section should be omitted.

Parameters
- **runs** (*List[Dict[str, Any]]*) – List of per-run summary dictionaries. Each dictionary may contain `failed_tests` and `ignored_tests` keys.
- **overall_status** (*str*) – The computed overall test status.

Returns
`True` if the appendix should be omitted, else `False`

Return type
bool

`_format_run_details`(*run*)

Format detailed failure/ignore information for a single run.

This method generates a list of lines describing failed and/or ignored tests for the given run. If no such tests exist, an empty list is returned. :param run: Dictionary containing per-run summary information. Expected keys include `name`, `failed_tests` and `ignored_tests`. :returns: List of formatted strings describing failed and/or ignored tests. Returns an empty list if none exist.

Parameters
run (*Dict[str, Any]*)
Return type
List[str]

`format_combined_summary`(*runs*, *title='SCP Integration Test Summary'*, *meta=None*)

Generate a formatted combined summary for multiple test runs.

This method aggregates per-run results, computes overall totals, and produces a structured, human-readable summary string

Parameters
- **runs** (*List[Dict[str, Any]]*) – List of per-run summary dictionaries. Each dictionary must contain keys such as `name`, `total`, `passed`, `failures`, `ignored`, and `status`.
- **title** (*str*) – Title displayed at the top of the summary. Defaults to `"SCP Integration Test Summary"`.
- **meta** (*Optional[Dict[str, Any]]*) – Optional metadata dictionary containing fields such as `platform`, `port`, or `suite`.

Returns
A newline-separated formatted summary string.
Return type
str

`_parse_overall_summary`(*text*, *test_name*)

Parse the overall test summary line from captured output.

Parameters
- **text** (*str*) – Full captured console output containing the summary line.
- **test_name** (*str*) – The test being parsed. Used for error reporting.

Returns
A tuple containing (total_tests, failures, ignored, passed).
Raises
- **AssertionError** – If `SUMMARY_RE` is not defined on the class.
- **AssertionError** – If the expected summary line is not found in the provided text.

Return type
tuple[int, int, int, int]

`_parse_test_results`(*text*)

Extract individual test result entries from captured output.

Parameters
text (*str*) – Full captured console output containing per-test results.
Returns
A list of dictionaries with keys `"name"` and `"status"` representing individual test outcomes.
Return type
List[Dict[str, str]]

_validate_results(*test_name*, *ok_present*, *failures*, *ignored*, *failed_tests*, *ignored_tests*)

Validate parsed test results and raise an error if validation fails.

Parameters

- **test_name** (*str*) – Name of the integration test being validated.
- **ok_present** (*bool*) – Indicates whether the OK marker was detected.
- **failures** (*int*) – Number of failed tests reported in the summary.
- **ignored** (*int*) – Number of ignored tests reported in the summary.
- **failed_tests** (*list*) – List of individual failed test names.
- **ignored_tests** (*list*) – List of individual ignored test names.

Raises

AssertionError – If the OK marker is missing or if any tests failed or were ignored.

Return type

None

_collect_problem_tests(*results*)

Collect failed and ignored test names from parsed test results.

This method iterates over parsed per-test result entries and separates tests marked as FAIL and IGNORE into two lists. :param results: Parsed per-test result entries, typically returned by *_parse_test_results()*. :returns: A tuple containing (failed_tests, ignored_tests).

summarize_results(*text*, *test_name*, *, *raise_on_fail=True*)

Generate a structured summary from raw integration test output.

Parameters

- **text** (*str*) – Full captured console output of the integration test.
- **test_name** (*str*) – Name of the test being summarized.
- **raise_on_fail** (*bool*) – Whether to raise an exception if validation fails. Defaults to True.

Returns

Dictionary containing structured summary information, including totals, status, and problematic test names.

Raises

AssertionError – If validation fails and raise_on_fail is True.

Return type

Dict[str, Any]

tests.utils.session_utils

Attributes

logger

Functions

expect_pattern_live(session, pattern, timeout, msg[, ...])	Wait for a specific pattern to appear in a live pexpect session.
fvp_power_cycle_and_reconnect(platform_ba ...)	Power cycle the FVP instance and restore console connectivity.
login_and_wait_for_shell(session, login_prompt, ...)	Complete login sequence and wait for root shell prompt.

Module Contents

tests.utils.session_utils.**logger**

tests.utils.session_utils.**expect_pattern_live**(*session*, *pattern*, *timeout*, *msg*, *level='info'*)

Wait for a specific pattern to appear in a live pexpect session.

Parameters

- **session** (*pexpect.spawn*) – Active pexpect session instance.
- **pattern** (*str*) – Pattern to wait for in the session output.
- **timeout** (*int*) – Maximum time (in seconds) to wait.
- **msg** (*str*) – Log message to emit if a timeout occurs.
- **level** (*str*) – Logging level to use.

Returns
True if the pattern is matched within the timeout

Return type
bool

tests.utils.session_utils.**fvp_power_cycle_and_reconnect**(*platform_base_obj*, *platform_bundle*, *boot_wait_seconds=0*)

Power cycle the FVP instance and restore console connectivity.

Parameters

- **platform_base_obj** – Initialized AutoTestPlatformBase.
- **platform_bundle** – Initialized driver bundle.
- **boot_wait_seconds** (*int*) – Optional delay in seconds

Returns
None

tests.utils.session_utils.**login_and_wait_for_shell**(*session*, *login_prompt*, *shell_prompt*)

Complete login sequence and wait for root shell prompt.

Parameters

- **login_prompt** (*str*) – login prompt pattern
- **shell_prompt** (*str*) – shell prompt pattern
- **session** (*pexpect.spawn*)

Returns
None

Return type
None

CHAPTER

TWO

RELEASE NOTES

Important

This page contains the common release notes for `sw-ref-stack`. For Arm Zena CSS release notes, see: https://arm-zena-css.docs.arm.com/en/latest/releasenotes.html.

2.1 v2.2

2.1.1 Common

New features

- Introduced linux-yocto-rt (PREEMPT_RT kernel) on RD-Aspen virtualization architecture for Xen domains DomU1 and DomU2.
- Enabled rasdaemon for RAS logging.

Changed

- Upgraded Xen version to 4.21.

2.2 v2.1

2.2.1 CSS-Aspen

New features

- Introduced GIC FMU driver.
- Added notification of FMU faults to the SSU.
- Introduced MHU FMU support.
- Introduced Software Built-In Self-Test Controller (SBISTC) fault handling.
- Introduced Platform Fault Detection Interface Monitoring.
- Introduced Arm Cryptographic Extension Demo.
- Introduced Transient Fault Protection for Application Processor.
- Introduced GIC Multiview on the Safety Island.
- Enabled Safety Island Cluster 1 to boot Zephyr with a “Hello World” application for CFG2.

- The sw-ref-stack supports two architectures, Baremetal and Virtualized. This release introduces the Virtualization architecture using Xen hypervisor.
- Introduced power and performance control feature for Application Processor.
- Introduced Mission-based Power Profile Demo for baremetal build.
- Introduced linux-yocto-rt (PREEMPT_RT kernel) on RD-Aspen baremetal demos.

Changed

- Refactored RAS synchronization between SI0 and the AP.
- Fixed Devicetree warnings within ACS suite.
- Fixed incorrect interrupt numbers in Devicetree.
- Removed the unused RSE non-secure image.
- Fixed RSE volatile memories size to 256KB.
- Update the RSE flash and AP flash images to be GPT partitioned.
- Configure the Safety Island ATU from the RSE instead of SI CL0.
- Upgraded Yocto from Styhead to Walnascar.
- Introduced Linux sniff test for unattended Debian distro installations.
- Debian sniff test results are accounted for in ACS tests results.

Resolved issues

- Fixed SVE related warning in Trusted Firmware-A RD-Aspen platform Makefile.
- Fixed RSS to SCP error message in RSE.
- Fixed false negative error in PFDI test case.
- Fixed the reboot hang when the number of AP CPUs is more than 4.
- Fixed link time error in TF-M's Release build type.
- Fixed 13 previously skipped psa-crypto-test cases
- Fixed ERXCTLR States for RAS configuration on all Application Processor cores.

The versions of the main components used in the Reference Software Stack:

Component	Version	Source
Arm Zena Compute Subsystem (CSS) FVP (FVP_RD_Aspen)	11.30.8	FVP Cfg1 download (arm64 host) FVP Cfg2 download (arm64 host) FVP Cfg1 download (x86 host) FVP Cfg2 download (x86 host)
RSE (Trusted Firmware-M)	9de5d116d02c07d490dc9c653e830 (based on main branch post v2.2.1)	Trusted Firmware-M repository
SCP-firmware	07181be79ae968be1479f0c714325a (based on main branch post v2.16.0)	SCP-Firmware repository
Trusted Firmware-A	168d78c376b7d39a40320df6852f13 (based on main branch post v2.13.0)	Trusted Firmware-A repository
OP-TEE	4936f055618d2a6a57ad6be12d5571 (based on master branch post v4.7.0)	OP-TEE repository
Trusted Services	337d474124a20a3d735dd22e5ccb4 (based on integration branch post v1.2.0)	Trusted Services repository
U-Boot	2025.10	U-Boot repository
Xen	4.20	Xen repository
Linux Kernel	6.12.30	Linux repository and Linux preempt-rt repository
Zephyr	4.1.0	Zephyr repository

Third-party Yocto layers used to build the Reference Software Stack:

```
URL: https://git.yoctoproject.org/meta-arm
layers: meta-arm, meta-arm-toolchain
branch: walnascar
revision: 21894cc2ea3197e6bfc1a56d889f757a09dc8b31

URL: https://gitlab.arm.com/cassini/meta-cassini
layers: meta-cassini-distro
branch: walnascar
revision: 4dad481980fb8a700cce8402bece7cf1bebbdee3

URL: https://github.com/kraj/meta-clang
layers: meta-clang
branch: walnascar
revision: 003cba92e982bdd565a6889f28799f8bba14957e

URL: https://gitlab.com/soafee/ewaol/meta-ewaol
layers: meta-ewaol
branch: walnascar
revision: 4ba5f48c4e10ad2a0271bb2287a66688e6c2fa15

URL: https://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-python, meta-perl
branch: walnascar
revision: 80ab58cc404959ae2f0e8b2e68935b3bfd8e8cfe

URL: https://github.com/pengutronix/meta-ptx
```

(continues on next page)

(continued from previous page)

```
layers: meta-ptx
branch: walnascar
revision: 23e46e92946ca0a1b1da4cf3ad212169d46b0af8

URL: https://github.com/Wind-River/meta-secure-core
layers: meta-secure-core-common, meta-efi-secure-boot, meta-signing-key
branch: walnascar
revision: 243281acbb4d3839b80b795030a7f4900e254735

URL: https://git.yoctoproject.org/meta-security
layers: meta-parsec
branch: walnascar
revision: 1f7eeb8e84811fa79b98f236ade42dc52d44cfc6

URL: https://git.yoctoproject.org/meta-virtualization
layers: meta-virtualization
branch: walnascar
revision: 898239e810acbb7db93299f20deec8afe434f11b

URL: https://git.yoctoproject.org/meta-zephyr
layers: meta-zephyr-core
branch: walnascar
revision: 3617fcdfd0f232dcaff4a153e667c26445b2077c

URL: https://git.yoctoproject.org/poky
layers: meta, meta-poky
branch: walnascar
tag: yocto-5.2.3
revision: a704e5171ce4f87e27408934b593e5a186ac1960
```

Limitations

- There are 4 unsupported test-cases out of 64 in PSA Crypto API test suite. Failed test cases are skipped.
- RSA is not supported by the current TF-M CryptoCell driver.
- CSS-Aspen FVP doesn't include a TrustZone Address Space Controller (TZC). Trusted Firmware-A doesn't program TZC to set up security configurations for DRAM or peripherals.
- System Memory Management Unit (SMMU) node is excluded from Primary Compute device tree.
- PCIe configuration is excluded.
- The flash device of TF-M Protected Storage (PS) in CSS-Aspen does not support the Replay-Protected Monotonic Counter (RPMC) feature. Instead PS Non-Volatile Counters (NV Counters) are implemented with a limited size in RSE OTP memory. Exceeding 512 writes to PS will cause the PS NV Counters to overflow and may trigger a system panic. Supporting an increase in the number of writes will require an increase in the NV Counter size.
- RSE flash is implemented as part of the wider system rather than within the CSS. The Internal Trusted Storage (ITS) is located in this external flash memory and therefore requires confidentiality, integrity protection, and replay protection against attackers with physical access to the device. These protections are typically achieved through a combination of software-based encryption and authentication, along with hardware features such as flash devices that include replay protection mechanisms or by writing replay protection values through the PSA Internal Trusted Storage (ITS) API.

Known issues

- Platform Fault Detection Interface (PFDI) Architecture Compliance Suite (ACS) An intermittent timeout may occur when running the PFDI Architecture Compliance Suite (ACS) on the Primary Compute. When this occurs, one or more subsequent test cases might be skipped. The issue affects ACS version 3.1.0, which is used in this release. Re-running the affected test suite usually completes all test cases successfully.

2.2.2 RD-Kronos

Changed

Following the latest Arm Automotive Solutions v2.1 release, Arm will no longer support Kronos, which was available in the previous v2.0 release. Please consider migrating to the latest Arm Zena Compute Subsystem (CSS). Details are available here.

2.3 v2.0

2.3.1 CSS-Aspen

New features

- CSS-Aspen secure boot-flow implemented for the RSE, Safety Island Cluster 0 and Primary Compute.
- Added support for DSU-120AE in TF-A.
- Added support for GIC-720AE in TF-A.
- Added support for RoS PLL configuration in SCP-firmware.
- Added CPU RAS error handling support in TF-A and SCP-Firmware.
- Added configurable Primary Compute CPUs topology feature.
- Added Platform Fault Detection Interface (PFDI).
- Implemented security features on Primary Compute.
  - Enabled Trusted Board Boot in TF-A.
  - Enabled SE-Proxy and SMM Gateway in Trusted Services, to provide secure services to Primary Compute Normal World.
  - Enabled UEFI secure boot in U-Boot.
- Added SystemReady Devicetree ACS 3.0.1 support including Base Boot Security Requirements Self-Certification Test (BBSR SCT) and Firmware Test Suite (BBSR FWTS).
- Added unattended and attended distro installation support for:
  - Debian v12.8.0
  - Fedora v39.1.5
  - openSUSE v15.5

Limitations

- CSS-Aspen FVP doesn't include a TrustZone Address Space Controller (TZC). Trusted Firmware-A doesn't program TZC to set up security configurations for DRAM or peripherals.
- System Memory Management Unit (SMMU) node is excluded from Primary Compute device tree.
- PCIe configuration is excluded.

- There are 17 unsupported test-cases out of 64 in PSA Crypto API test suite. Failed test cases are skipped to unblock CI pipeline.
- The flash device of TF-M Protected Storage (PS) in CSS-Aspen does not support the Replay-Protected Monotonic Counter (RPMC) feature. Instead PS Non-Volatile Counters (NV Counters) are implemented with a limited size in RSE OTP memory. Exceeding 512 writes to PS will cause the PS NV Counters to overflow and may trigger a system panic. Supporting an increase in the number of writes will require an increase in the NV Counter size.

Known issues

- The number of available Primary Compute CPUs is currently limited to 4 due to a reboot-related issue. After initiating a reboot from Linux, the Out-of-Reset Platform Fault Detection Interface (PFDI) fails to re-enable all CPU cores.

2.3.2 Kronos

New features

- Introduced a new build called *Arm Automotive Solutions Demo*, which demonstrates the use cases of the Arm Automotive Solutions Software Reference Stack, including:
  - Critical Application Monitoring Demo
  - Safety Island Actuation Demo
  - Safety Island Communication (utilizing HIPC)
  - Parsec-enabled TLS Demo
  - Primary Compute PSA Secure Storage and Crypto APIs Architecture Test Suite
  - Safety Island PSA Secure Storage APIs Architecture Test Suite
  - Safety Island PSA Crypto APIs Architecture Test Suite
  - Fault Management Demo
  - Secure Firmware Update
- Added Debian v12.8.0 distribution unattended installation option.

Changed

- Updated the build host system requirements to Ubuntu 22.04.
- Updated RD-1 AE FVP to 11.29.27 version.

Limitations

- Same as *v1.1.1 Limitations*.

Resolved and known issues

Resolved issues

- Backported PPU AE feature addition in SCP-firmware to align with the Safety Island Cortex-R82AE modeling fix in the RD-1 AE FVP.
- Added GIC-720AE IIDR in TF-A to align with the corrected IIDR in the RD-1 AE FVP.

Known issues

- Same as *v1.1.1 Known Issues*.

2.4 v1.1.1

2.4.1 New features

No new features were introduced.

2.4.2 Changed

- Change diagram showing RSE-oriented boot flow to show that RSE BL1_2 is executed from SRAM.
- Change diagram showing RSE-oriented boot flow to show that BL2 releases SCP RAMFW from reset.
- Change diagram showing Yocto layer dependencies to show that meta-efi-secure-boot depends on meta-perl.
- Added command to refresh firmware image before re-running entire ACS test suite.

Bug fixes as listed in *Resolved issues*.

2.4.3 Limitations

- The platform lacks hardware support for partitioning DRAM into secure and non-secure regions. Consequently, a non-secure endpoint running on the AP can access the DRAM region allocated for AP BL32, compromising its security.
- Same as *v1.1 Limitations*.

2.4.4 Resolved and known issues

Resolved issues

- Added Google Analytics extension for Read The Docs to replace removed integrated Google Analytics support.
- Fixed incorrect `if()` conditional statements in RSE drivers.
- Backported fixes for SCP vulnerabilities:
  - CVE-2024-9413.
  - CVE-2024-11863.
  - CVE-2024-11864.
- Fixed MHUv3 communication documentation mistake in Secure Services section.
- Fixed GIC GICR register region size and PSCI cpu_on function ID in Kronos device tree.

Known issues

- Same as *v1.1 Known Issues*.

2.5 v1.1

2.5.1 New features

Implementation of UEFI secure boot.

2.5.2 Changed

- Use the EWAOL Yocto distribution instead of Cassini.
- Extended SystemReady IR ACS with the Security Interface Extension (SIE) Self-Certification Test (SCT).
- Assembled the firmware images using genimage from the meta-ptx Yocto layer instead of wks/wic images.
- Updated support from openSUSE 15.4 to 15.5.
- Updated support from Debian 11.7 to 12.4.
- Added Fedora 39.1.5 distribution to comply with the SystemReady IR v2.1 requirements.
- Added Fedora 39.1.5 distribution unattended installation option.
- Added openSUSE 15.5 distribution unattended installation option.
- Added compiler tuning for Cortex-R82 to the Zephyr toolchain.
- Upgraded from Yocto nanbield to scarthgap.
- Introduced the Yocto layer meta-arm-safety-island.
- Removed LCP from the boot flow.
- Aligned the number of supported MHUv3 channels with the RSE specification.
- Enabled TF-A Trusted Board Boot (TBB).
- Enabled PSA Internal Trusted Storage API on Primary Compute.
- Added an AP_REFCLK non-secure Generic Timer node in Kronos device tree.
- Updated identified non-alignments on RD-Kronos for Devicetree missing schemas.
- Introduced Safety Island GIC FMU device for Safety Island Cluster 1 and automated tests.
- Renamed Runtime Security Subsystem (RSS) to Runtime Security Engine (RSE) to be aligned with TF-M naming.
- Fixed the System FMU ERRIIDR register value in the Fault Management driver.
- Fixed the GIC-720AE IVIEWRn register offsets in TF-M.
- Using bindings of Linux Kernel from 6.3.7 to 6.10 for SystemReady IR Devicetree validation.
- Supported EFI System Partition (ESP) checks in Arm Systemready IR ACS test.
- Updated Safety Island Actuation Demo from v2.0 to v2.1.
- Added Secure Firmware Update support on Virtualization architecture.
- Enabled capsule authentication for Secure Firmware Update.
- Automated SystemReady IR capsule update test.
- Made the Dom0 RAM size configurable.
- Exposed the following kas build parameters:
  - `CASSINI_ROOTFS_EXTRA_SPACE`
  - `BAREMETAL_IMAGE_MEM_SIZE`
  - `DOM0_MEMORY_SIZE`
  - `DOMU1_MEMORY_SIZE`
  - `DOMU2_MEMORY_SIZE`

- Updated Critical Application Monitoring Demo from v1.0 to v1.1.

The versions of the main components used in the Reference Software Stack:

Component	Version	Source
Arm Reference Design-1 AE FVP (FVP_RD_1_AE)	11.27.20	FVP download (arm64 host) FVP download (x86 host)
RSE (Trusted Firmware-M)	53aa78efef274b9e46e63b429078ae1863609728 (based on main branch post v1.8.1)	Trusted Firmware-M repository
SCP-firmware	cc4c9e017348d92054f74026ee1beb081403c168 (based on main branch post v2.13.0)	SCP-Firmware repository
Trusted Firmware-A	168d78c376b7d39a40320df6852f13b633a4ccee (based on main branch post v2.13.0)	Trusted Firmware-A repository
OP-TEE	3.22.0	OP-TEE repository
Trusted Services	602be607198ea784bc5ab1c0c9d3ac4e2c67f1d9 (based on main branch, post v1.0.0)	Trusted Services repository
U-Boot	2023.07.02	U-Boot repository
Xen	4.18	Xen repository
Linux Kernel	6.6.35	Linux repository and Linux preempt-rt repository
Zephyr	3.5.0	Zephyr repository
Safety Island Actuation Demo	v2.1	Actuation repository
Mbed TLS	1ec69067fa1351427f904362c1221b31538c8b57 (based on 3.5.0)	Mbed TLS repository
Critical Application Monitoring	v1.1	Critical Application Monitoring repository

Third-party Yocto layers used to build the Reference Software Stack:

```
URL: https://git.yoctoproject.org/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-systemready, meta-arm-toolchain
branch: scarthgap
revision: 38bce82e42ea093333a53c4a10e51d1b26cbc989

URL: https://gitlab.arm.com/cassini/meta-cassini
layers: meta-cassini-distro, meta-cassini-tests
branch: scarthgap
tag: v2.0.0
revision: bef1d728c6db464ff89828afae5b51e648058f35

URL: https://github.com/kraj/meta-clang
layers: meta-clang
branch: scarthgap
revision: 0acff283249842eb1f617b20c2ed4ebf9f8e3557

URL: https://gitlab.com/soafee/ewaol/meta-ewaol
layers: meta-ewaol
branch: scarthgap
tag: ewaol-2.0.0
revision: c28142e72691202ba55a954f0faaed4375615b68

URL: https://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-python, meta-perl
```

(continues on next page)

(continued from previous page)

```
branch: scarthgap
revision: 78a14731cf0cf38a19ff8bd0e9255b319afaf3a7

URL: https://github.com/pengutronix/meta-ptx
layers: meta-ptx
branch: scarthgap
revision: 547b079bf309ebe1576aa5ae0d58564feb245a42

URL: https://github.com/Wind-River/meta-secure-core
layers: meta-secure-core-common, meta-efi-secure-boot, meta-signing-key
branch: scarthgap
revision: f3f928d097917b8a131044fe718440eb7f7e381b

URL: https://git.yoctoproject.org/meta-security
layers: meta-parsec
branch: scarthgap
revision: 11ea91192d43d7c2b0b95a93aa63ca7e73e38034

URL: https://git.yoctoproject.org/meta-virtualization
layers: meta-virtualization
branch: scarthgap
revision: 37c06acf58f9020bccfc61954eeefe160642d5f3

URL: https://git.yoctoproject.org/meta-zephyr
layers: meta-zephyr-core
branch: scarthgap
revision: 763c72fc3088fc09ccfde6edfcdad43811d16616

URL: https://git.yoctoproject.org/poky
layers: meta, meta-poky
branch: scarthgap
revision: ca27724b44031fe11b631ee50eb1e20f7a60009d
```

2.5.3 Limitations

- Same as *v1.0 Limitations* with the following exception:
  - Now, the Reference Software Stack also supports the Internal Trusted Storage (ITS) API on the Primary Compute.

2.5.4 Resolved and known issues

Resolved issues

- Added runtime checks of Update Capsule flags in U-Boot, which fixed SystemReady IR ACS SCT Update Capsule test failure.
- Fixed a bug in TF-M where the RSE communication request from AP was not handled by RSE.

Known issues

- For Heterogeneous Inter-Processor Communication (HIPC), during ping between Clusters, a transient issue is observed where ICMP replies take longer time to reach the originating Cluster.
- The CAM automated validation might rarely fail with the error: "Received timestamp is in the future" in the Safety Island console. This is caused by PTP sync loss between the Primary Compute and Safety Island in the FVP model.
- The Virtualization Architecture might rarely fail to boot a DomU, leaving it hanging before reaching its shell. This may be caused by an RCU stalling issue. The last expected line printed by the DomU is (potentially followed by an RCU backtrace):

  ```
  Freeing initrd memory: 117108K
  ```

  When running the Automated Validation the output looks like:

  ```
  pexpect.exceptions.TIMEOUT: Timeout exceeded.
  [...]
  RESULTS - test_10_linuxlogin.LinuxLoginTest.test_linux_login: ERROR
  ```

  To overcome the problem, restart the command that launched the FVP (either directly or through the Automated Validation).
- Same as *v1.0 Known Issues*.

2.6 v1.0

2.6.1 New features

Implementation of the use cases.

The versions of the main components used in the Reference Software Stack:

Component	Version	Source
Kronos Reference Design FVP (FVP_RD_Kronos)	11.25.15	FVP download (arm64 host) FVP download (x86 host)
RSS (Trusted Firmware-M)	53aa78efef274b9e46e63b429078ae1863609728 (based on master branch post v1.8.1)	Trusted Firmware-M repository
SCP-firmware	cc4c9e017348d92054f74026ee1beb081403c168 (based on master branch post v2.13.0)	SCP-Firmware repository
Trusted Firmware-A	2.8.0	Trusted Firmware-A repository
OP-TEE	3.22.0	OP-TEE repository
Trusted Services	08b3d39471f4914186bd23793dc920e83b0e3197 (based on main branch, pre v1.0.0)	Trusted Services repository
U-Boot	2023.07.02	U-Boot repository
Xen	4.18	Xen repository
Linux Kernel	6.1.73	Linux repository and Linux preempt-rt repository
Zephyr	3.5.0	Zephyr repository
Safety Island Actuation Demo	v2.0	Actuation repository
Mbed TLS	1ec69067fa1351427f904362c1221b31538c8b57 (based on 3.5.0)	Mbed TLS repository
Critical Application Monitoring	v1.0	Critical Application Monitoring repository

Third-party Yocto layers used to build the Reference Software Stack:

```
URL: https://git.yoctoproject.org/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-systemready, meta-arm-toolchain
branch: kronos-nanbield
revision: 5e4851a884985b952b33f6f88a8724fbbe5300ec

URL: https://gitlab.arm.com/cassini/meta-cassini
layers: meta-cassini-distro
branch: nanbield
revision: v1.1.0

URL: https://github.com/kraj/meta-clang
layers: meta-clang
branch: nanbield
revision: 5170ec9cdfe215fcef146fa9142521bfad1d7d6c

URL: https://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-python
branch: nanbield
revision: da9063bdfbe130f424ba487f167da68e0ce90e7d

URL: https://git.yoctoproject.org/meta-security
layers: meta-parsec
branch: nanbield
revision: 5938fa58396968cc6412b398d403e37da5b27fce

URL: https://git.yoctoproject.org/meta-virtualization
layers: meta-virtualization
```

(continues on next page)

(continued from previous page)

```
branch: nanbield
revision: ac125d881f34ff356390e19e02964f8980d4ec38

URL: https://git.yoctoproject.org/meta-zephyr
layers: meta-zephyr-core
branch: nanbield
revision: fa76b75bd65da63abcc2d65dd5d4eb24296f2f65

URL: https://git.yoctoproject.org/poky
layers: meta, meta-poky
branch: nanbield
revision: 1a5c00f00c14cee3ba5d39c8c8db7a9738469eab
```

2.6.2 Changed

Initial version.

2.6.3 Limitations

- In the HIPC, the iperf parameter "-l/–length" should be less than 1473 (IP and UDP overhead) in the case of Zephyr running as a UDP server since it does not support IP fragmentation.
- PSA Secure Storage API defines two interfaces for storages: Internal Trusted Storage (ITS) API and Protected Storage (PS) API. For now the Reference Software Stack supports the ITS API on Safety Island only.
- PSA Protected Storage Optional APIs `psa_ps_create` and `psa_ps_extended` are not supported by Arm Automotive Solutions as they are not implemented in the Protected Storage Service provided by Trusted Firmware-M.
- PSA Secure Storage APIs Architecture Test Suite only runs on Cluster 2 in the Safety Island due to the following limitations:
  - Trusted Firmware-M supports a single partition only. This causes tests running simultaneously on different entities to interfere with each other due to accessing the same assets, resulting in failures.
  - Trusted Firmware-M has no support against Denial of Service attacks, where a test running on one entity might take up all the storage on the RSS resulting in denial of service for tests running on other entities.

2.6.4 Resolved and known issues

Known issues

- The automated validation might fail due to the encoding issues in the logs. This has been observed on an AWS aarch64 Graviton 2 build host. In the test logs, the error message that appears is a typical timeout error.

  The console log appears normal, but some characters are either corrupted or replaced with 00, x00 or ^@ characters. This issue is likely caused by encoding mismatches or inconsistencies in the logging process, and it could occur in any of the test suites. When this issue occurs, something similar to the following would be observed in the logs:

  ```
  52 28 bytes from 192.168.1.2 to 192.168.1.1: icmp_seq=7 ttl=64 time=0.00 ^@s^M
  or
  fault set_critical f\00u@2a570000 0x10000600 0
  or
  System shutdown complet\x00
  ```

  If this occurs, trigger the "Automated Validation" again to resolve it.

- Automated validation may fail at times due to CPU frequency and throttling issues. If this occurs, trigger the "Automated Validation" again to resolve it.
- Refer to Critical Application Monitoring Known Issues for CAM-related known issues.

CHAPTER

THREE

LICENSE

This project follows the REUSE specification for declaring copyright and licensing.

The repository contains files under **multiple licenses**.

Each file in the repository is individually licensed. The applicable license is identified by the file-level SPDX headers. In case of any doubt, the file-level SPDX headers are authoritative.

All applicable licenses are available in the LICENSES directory.

3.1 Apache-2.0

Files licensed under the Apache License 2.0 are subject to the terms of that license.

Files under this license use the `SPDX-License-Identifier`:

```
Apache-2.0
```

The full license text is available in LICENSES/Apache-2.0.txt.

3.2 BSD-2-Clause

Files licensed under the BSD 2-Clause License are subject to the terms of that license.

Files under this license use the `SPDX-License-Identifier`:

```
BSD-2-Clause
```

The full license text is available in LICENSES/BSD-2-Clause.txt.

3.3 GPL-2.0-only

Files licensed under the GNU General Public License, version 2 only, are subject to the terms of that license.

Files under this license use the `SPDX-License-Identifier`:

```
GPL-2.0-only
```

The full license text is available in LICENSES/GPL-2.0-only.txt.

3.4 GPL-2.0-or-later

Files licensed under the GNU General Public License, version 2 or later, are subject to the terms of that license.

Files under this license use the `SPDX-License-Identifier`:

```
GPL-2.0-or-later
```

The full license text is available in LICENSES/GPL-2.0-or-later.txt.

3.5 LGPL-2.1-only

Files licensed under the GNU Lesser General Public License, version 2.1, are subject to the terms of that license.

Files under this license use the `SPDX-License-Identifier`:

```
LGPL-2.1-only
```

The full license text is available in LICENSES/LGPL-2.1-only.txt.

3.6 Linux-syscall-note

Files licensed under the Linux-syscall-note License are subject to the terms of that license.

Files under this license use the `SPDX-License-Identifier`:

```
Linux-syscall-note
```

The full license text is available in LICENSES/Linux-syscall-note.txt.

3.7 MIT

Files licensed under the MIT License are subject to the terms of that license.

Files under this license use the `SPDX-License-Identifier`:

```
MIT
```

The full license text is available in LICENSES/MIT.txt.

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

