ML on MCU Documentation

Release 0.3.0

TUM Department of Electrical and Computer Engineering - Chair o

Dec 17, 2023

CONTENTS:

CHAPTER

ONE

ML ON MCU

This project contains research code related to the deployment of inference or learning applications on tiny micro-controllers.

- Free software: Apache License, Version 2.0
- Python Package: https://pypi.org/project/mlonmcu/
- Documentation: https://mlonmcu.readthedocs.io or https://tum-ei-eda.github.io/mlonmcu/

1.1 Features

- Highly configurable python package
- Automatic resolution and installation of dependencies
- Supporting a large combination of frameworks/backends/targets/features
- Build-in parallel processing of large number of benchmarks
- Isolated enironments (not interfering with other installations)
- Command Line and Python Development Interfaces
- Docker images to get started quickly
- Extensive documentation on usage and code details
- CI/CD integration and high PyTest coverage

1.2 Getting started

1.2.1 Prerequisites

Ubuntu/Debian

First, a set of APT packages needs to be installed:

```
# Python related
sudo apt install python3-pip python3-venv

# MLonMCU related
sudo apt install libboost-all-dev graphviz doxygen libtinfo-dev zlib1g-dev texinfo unzip␣
↪device-tree-compiler tree g++

# Optional (depending on configuration)
sudo apt install ninja-build
```

Also make sure that your default Python is at least v3.7. If the `python` command is not available in your shell or points Python v2.7 check out `python-is-python3`.

Warning: It seems like the ETISS tool fails to compile if if find a version of LLVM 11 on your system which does not include Clang 11. The best workaround for now is to (if possible) remove those tools from your system: `sudo apt remove llvm-11* clang-11*` (See issue #1)

Make sure to use a fresh virtual Python environment in the following steps.

Install Release from PyPI

Warning: As the PyPI package is not always up to date, it is currently recommented to use a self-build version of the package (as explained in the next section)

To use the PIP package, run the following: `pip install mlonmcu` (Add `--user` if you are not using a virtual environment)

Build Package manually

First, install all relevant dependencies:

```
python -m venv .venv  # Feel free to choose a different directory or use a conda␣
↪environment

# Run this whenever your have updated the repository
source .venv/bin/activate

# Environment-specific dependencies are installed later

**Warning:** It is recommended to have at least version 3.20 of CMake installed for full␣
↪compatibility!

# Install ptional dependecies (only for development)
pip install -r requirements_dev.txt
pip install -r docs/requirements.txt

# Only if you want to use the provided python notebooks, as explained in  ./ipynb/README.
↪md
pip install -r ipynb/requirements.txt
```

Then you should be able to install the `mlonmcu` python package like this

```
# Optionally remove an older version first: pip uninstall mlonmcu

make install  # Alternative: python setup.py install
```

Docker (Any other OS)

See ./docker/README.md for more details.

This repository ships three different types of docker images based on Debian:

- A minimal one with preinstalled software dependencies and python packages

  Feel free to use this one if you do not want to install anything (except Docker) on your main sytem to work with mlonmcu

- A medium one which already has the `mlonmcu` python package installed

  Recommended and the easiest to use. (Especially when using `docker-compose` to mount shared directories etc.)

- A very large one with an already initialized and installed

  Mainly used for triggering automated benchmarks without spending too much time on downloading/compiling heavy dependencies over and over again.

1.2.2 Usage

Is is recommended to checkout the provided Demo Jupyter Notebook as it contains a end-to-end example which should help to understand the main concepts and methodology of the tool. The following paragraphs can be seen as a TL;DL version of the information in that Demo notebook.

While some tools and features of this project work out of the box, some of them require setting up an environment where additional dependencies are installed. This can be achived by creating a MLonMCU environment as follows:

```
mlonmcu init
```

Make sure to point the `MLONMCU_HOME` environment variable to the location of the previously initialied environment. (Alternative: use the `default` environment or `--home` argument on the command line)

Next, generate a `requirements_addition.txt` file inside the environment directory using `mlonmcu setup -g` which now be installed by running `pip install -r $MLONMCU_HOME/requirements_addition.txt` inside the virtual Python environment.

To use the created environment in a python program, a `MlonMcuContext` needs to be created as follows:

```
import mlonmcu.context

with mlonmcu.context.MlonMcuContext() as context:
    pass
```

1.3 Development

Make sure to first install the additonal set of development Python packages into your virtual environment:

```
pip install -r requirements_all.txt  # Install packages for every component (instead of␣
↪using mlonmcu setup -g)
pip install -r requirements_dev.txt  # Building distributions and running tests
pip install -r docs/requirements.txt  # For working with the documentation
```

Unit test and integration test are defined in the `tests/` directory and can be triggered using `make test` or `pytest tests/`

Coverage can be determined by running `make coverage`. The latest coverage report (HTML) for the default branch can also be found as an artifact of the CI/CD workflow.

Documentation is mainly generated automatically from doctrings (triggered via `make html`). It is also possible to include markdown files from the repo into the `.rst` files found in the *docs/* directory. There is a GitHub workflow which publishes the documentation for the default branch to our GitHub Pages.

Regarding coding style, it is recommended to run `black` before every commit. The default line length should be given in the `setup.cfg` file.

1.4 Credits

This is a research project proposed by the Chair of Design Automation of the Technical University of Munich.

1.4.1 Developers

- Rafael Stahl (TUM) [@rafzi]
  - Wrote initial version of the MLonMCU project
- Philipp van Kempen (TUM) [@PhilippvK]
  - Came up with MLonMCU Python package

1.4.2 Other

This package was created with Cookiecutter_ and the `audreyr/cookiecutter-pypackage_` project template. However most of the templates was manually changed to be in Markdown instead of reStructuredText.

- **Cookiecutter:** https://github.com/audreyr/cookiecutter
- **audreyr/cookiecutter-pypackage:** https://github.com/audreyr/cookiecutter-pypackage

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install ML on MCU, run this command in your terminal:

```
$ pip install mlonmcu
```

This is the preferred method to install ML on MCU, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for ML on MCU can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/tum-ei-eda/mlonmcu
```

Or download the tarball:

```
$ curl -OJL https://github.com/tum-ei-eda/mlonmcu/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

To use ML on MCU in a project:

```
import mlonmcu
```

CHAPTER

FOUR

IMPORTANT TERMS AND DESIGN DECISIONS (RFC)

MLonMCU offers a hand full of high level interfaces as well as a large number of internally used objects. You may use this document as a Glossary to understand the meaning of these some core concepts of the MLonMCU software infrastructure.

4.1 Motivation and Goals

MLonMCU is basically a reimplemented version of a TinyML benchmarking project which was used internally before for about one year.

The open source design was approached with the following set of goals in mind:

- Split up previously used all-time growing Python script into a hierarchical Python package
- Dependency management should be mostly invisible to the user without interfering with other software installed on the system
- In addition to revamping the existing command line interface, a Python API should be integrated to ease scripting and access to intermediate results.
- Increase scalability of large benchmarking tasks by inherently supporting parallelisms in multiple dimensions (Model x Backend x Features x Targets)
- Improve expandability by providing “Plugin” interfaces for various features.
- Ability to support further targets and architecture as well as real Hardware
- Improve Code Quality by adding unit and integration tests and extensive CI/CD applications.
- Provide a common interface to all supported backends by adding wrappers for their Command Line Interfaces
- Offer a large number of examples and extensive documentation to enable the TinyML community to get started with MLonMCU easily
- Ensure reproducibility of research results by improved logging and import options and isolated environments.

4.2 Fundamentals

4.2.1 Features and Configuration

Two types of options can be found in a large number of classes in the MLonMCU Package: `features : List[Feature]` and `config : Dict`. This design decision leads to unified command line interfaces and less framework/backend/target/frontend/feature specific code in higher levels of the codebase. A baseline requirement for all classes which implement those two concepts is the definition of the class variables `FEATURES`, `DEFAULTS` and `REQUIRED` .

Learn more about these features here.

4.2.2 Contexts and Environments

TODO

4.2.3 Session Management/Run Definition

4.2.4 Artifacts Handling

TODO

4.2.5 Abstraction at Various Levels

Inheritance is used at multiple levels in the MLonMCU project to introduce abstract interfaces for important objects.

Here ere the most relevant examples:

- **Backend:** A backend is a wrapper for a specific code generator
- **Framework:** The used framework is implicitly defined by the backend.
- **Target:** This contains definitions to interface with read hardware or a simulator.
- **Frontend:** Loading and converting models of various types and features is done by the Frontend classes.
- **Feature:** As features are a property of the aforementioned classes, they can have multiple base classes, e.g. `FrameworkFeature`, `TargetFeature`

There are two exceptions to this scheme:

MLIF class: The CMake code which is used in the MLonMCU flow could be replaced relatively easily as long the alternative is offering similar command line options or overwrites parts of the MLIF class definition. If this becomes the case, a abstract base class inherited by the new class Wes well as MLIF can be added. The only way why it does not yet exist is because I did not yet came up with a suitable name for this base class. If you have something better than `class Compile` in mind, please raise an issue to discuss your proposal.

Setup class: At the current point in time, it is quiet unrealistic, that the current dependency resolution mechanism would be replaced by an alternative tool. However there is at least one options which shall be evaluated in the future (See CK). We than might need to discuss how to rename the currently very generic Name `class Setup` of the original approach by a new name and what the base class should be called.

CHAPTER

FIVE

COMPONENTS

5.1 Models and Frontends

The types of models which can be processed with MLonMCU are given by the implemented frontends. The Following Table shows the currently supported ones:

Frontend	Formats
TFLite	`.tflite`

Here is also a list of frontends with will probably implemented in the furture:

- ONNX (`onnx`)
- TensorFlow SavedModel (`.pb`)

While you can use your own models we also provide support for the following model zoos which can be cloned from GitHub:

- Model Collection by EDA@TUM (`tum-ei-eda/mlonmcu-models`)
- ARM Model Zoo (`ARM-software/ML-zoo`) Work in Progress

5.2 Frameworks and Backends

For each framework supported by MLonMCU, a number of backends is implemented.

Framework	Backends
TenflowFlow Litefor Microcontrollers (`tflm`)	Default Interpreter (`tflmi`)Offline Compiler (`tflmc`) Work in Progress
TVM (`tvm`)	AoT Executor (`tvmaot`)Graph Executor (`tvmrt`)Custom Codegenerator (`tvmcg`)

5.3 Platforms and Targets

While support for some targets (especially simulator based ones) is directly build into MLonMCU, a platform is used for more complivcated targets (e.g. real hardware) to reuse existing Flows for compiling and flashing

Platform	Targets
Default	**RISC-V:**ETISS/Pulpino (`etiss_pulpino`)Spike (`spike`)VPSimPlus (`ovpsim`)**ARM:**Corstone300 FVP (`corstone300`)**x86:**Host `host_x86`)
ESP IDF (`espidf`)	**LX6:**ESP32 (`esp32`)**RISC-V:**ESP32-C3 (`esp32c3`)

To extend the support of read hardware targets, it would be great if this list would be extended by some of the following platforms in the furture:

- Arduino Excosystem
- PlatformIO
- ZephyrOS

5.4 Features

An extensive overview of the available features in TVM is given in the following table. The types of those features are denoted with a check mark in the respective column.

Feature	Setup	Frontend	Framework	Backend	Target	Platform/Compile	Other
Debug Arena Usage (`debug_arena`)							
Validate Output Data (`validate`)							
muRISCV-NN (`muriscvnn`)							
CMSIS-NN (`cmsisnn`)							
CMSIS-NN + TVM BYOC (`cmsisnnbyoc`)							
Fused TIling for TVM (`fusetile`)							
Custom TVM meory planner (`memplan`)							
Unified Static Memory Planner for TVM (`usmp`)							
V-Extension for RISC-V (`vext`)					(`spike`, `ovpsim`)		
Debug Build (`debug`)							
GDBServer (`gdbserver`)							
Debug ETISS VP (`etissdbg`)					(`etiss_pulpino`)		
Create Memory Trace (`trace`)					(`etiss_pulpino`)		
Unpacked API (`unpacked_api`)				(`tvmaot`)			
Autotune TVM Model (`autotune`)							
Use TVM Tuning Records (`autotuned`)							

5.5 Managed Dependencies

MLonMCU tries to either manage dependencies internally (hidden to the user) or rely on 3rd party platforms to install them.

The following list gives and overview of the set of dependencies which are currently managed:

- Toolchains - RISC-V GCC Linux Toolchain - Download and extract - ARM GCC Linux Toolchain - Download and extract - LLVM - Download and extract
- Targets/Simulators - ETISS - Clone Repository - Build ETISS - Install ETISS - Build `bare_etiss_processor` - Spike - Clone Repositories - Build Proxy Kernel - Build Simulator - Corstone-300 - Download and extract - Install FVP
- Frameworks/Backends - TFLM - Clone Repository - Download 3rd party dependencies - TFLite Micro Compiler - Clone Repository - Build - TVM - Clone Repository (including 3rd party dependencies) - Configure & Build - uTVM Staticrt Codegen - Clone Repository - Build

- Features - muRISCV-NN - Clone Repository - Build - CMSIS(-NN) - Clone Repository - Build

The following dependencies are intentionally NOT managed by MLonMCU:

- OVPSimPlus: The simulator is closed source and needs an individual license for usage (free)
- ESP-IDF: Make sure you provide a `espidf.path` with the required components installed

CHAPTER

SIX

COMPONENTS IN DETAIL

6.1 CMSIS-NN BYOC Feature

6.1.1 Usage

- The used extensions have to be manually selected by setting `cmsisnnbyoc.mcpu` on the command line

6.1.2 Compatibility

- The `cmsisnnbyoc` feature is not compatible with the `desired_layout` config for the TVM targets

6.2 muRISCV-NN BYOC Feature

6.2.1 Usage

- The used extensions have to be manually selected by setting `muriscvnnbyoc.mcpu` on the command line
- Example mapping: `cortex-m55 -> VEXT`, `cortex-m33 -> PEXT`, `cortex-m0 -> No extensions`

6.2.2 Compatibility

- The `muriscvnnbyoc` feature is not compatible with the `desired_layout` config for the TVM targets

6.3 Auto-Vectorize Feature

6.3.1 Usage

- Add `-f auto_vectorize` to the command line arguments

6.3.2 TODOs:

- [ ] Configure loop and basic block vectorizer individually.

6.3.3 Warning

- Auto vectorization is enables by default (if available) on the following optimization levels:
  - GCC: `-O2`
  - LLVM: `-O1`

6.3.4 Configuration

- `auto_vectorize.enable`: Allows to turn off ne auto-vectorization completely (Default: `true`)
- `auto_vectorize.verbose`: Print details about auto vectorization possibilities during compililation. Need to check the MLID stdout artifact or enable `mlif.print_outputs` to be effective (Default: `false`)

6.3.5 Compatibility

- Only RISC-V targets is supported at the momemt
- The supported MLIF toolchains are GCC and LLVM
- A VLEN larger equals 128 is required for this feature
- It seems like this currently needs a ELEN=64 and proper alignment (e.g. `tvmaot.alignment_bytes=8`) for the backend data. TFLMI seems to break with this.

CHAPTER

SEVEN

ENVIRONMENTS

7.1 Motivation

While the base set of features in MLonMCU should work out of the box, there are some reasons for not sticking to predefined default values. Instead of hardcoding values such as repository urls or file paths inside the codebase, they can be completely configured by the user to allow setting up custom environments with very low efforts required. Having multiple MLonMCU environments installed in parallel has further advantages as they are completely isolated from each other an therefore allow using different versions of components and a different set of features. In addition there is a possibility to turn of certain components completely to reduce the installation time. User configuration for the MLonMCU flow which would typically need to be passed via the command line can be instead defined in the environment file which going to be explained next.

7.2 `environment.yml` Explained

Each MLonMCU environment has a unique directory which can be chosen by the user where dependencies are installed and exports are written to. In this directory which can also be referred as MLonMCU-Home the environment configuration file `environment.yml` can be found. The basic structure of this YAML file can be summarized as follows:

```yaml
# The MLONMCU_HOME is filled in automatically when creating the environment
home: "{{ home_dir }}"
logging:
  enabled: false
  ...
# Default locations for certain directoriescan be changed here
# Non-absolute paths will always be threated relative to the MLONMCU_HOME
paths:
  deps: deps
  ...
# Here default clone_urls
repos:
  some_repo:
    url: "insert_repo_url_here"
    ref: optional_branch_tag_or_commit
  ...

# Here all supported frameworks with their specific features are defined
# Optionally disable unwanted or incomatible backends or features here
# The configured defaults are used if no backend was specified in the command line␣
↪options
```

(continues on next page)

(continued from previous page)

```
frameworks:
  default: some_framework
  some_framework:
    enabled: true
    backends:
      default: some_backend
      some_backend:
        enabled: true
        features:
          some_feature: true
          ...
      ...
    features:
      another_feature: true
      ...
  ...
# The enabled fronends are processed in the order defined here until a compatible one is␣
↪found for a given model type
frontends:
  some_frontend:
    enabled: true
    features:
      some_feature: false
      ...
  another_frontend:
    enabled: false
  ...
# List of supported targets in the environment
targets:
  default: some_target
  some_target:
    enabled: true
    features:
      some_featuee: true
  ...
# This is where further options such as specific versions of dependencies can be set in␣
↪the furture
vars:
  some_backend.some_var: 10
  foo: "bar"
```

While some parts of the file can theoretically be omitted, it is not recommended to do do. Also it has to be noted, that frameworks, backends, targets, frontends and features need to be explicitly enabled in the environment file to be available in the MLonMCU flow.

Hint: The `default` property which is available for some components supports wildcards, e.g. instead of providing a single backend name just put in "*" to use all enabled backends of the given framework by default.

7.3 Environment Templates

There is a set of environment file templates provided with the MLonMCU package which can be chosen from by the user e.g.

- `default`: Should work out of the box for everyone
- `minimal`: Stripped down version of MLonMCU with only a small set of dependencies (just the essentials)
- `dev`: Development version which will is not guaranteed to work all the time.
- `tumeda`: Version on MLonMCU depending on tool which are (not yet) available publicly.

After a template was chosen, the initial environment file is being generated which can be freely modified by the user afterwards.

7.4 Creating environments

7.4.1 Command line (recommended)

To get started with MLonMCU on the command line first an environment has to be created using the `mlonmcu init` command. As only some usage examples are shown in the following, make sure to check out `mlonmcu init --help` to learn more.

- Initialize a default environment at the default location (`~/.config/mlonmcu/environments/default` on most UNIX Systems): `mlonmcu init`
- Initialize an environment inside the current working directory: `mlonmcu init -H .`

The tool will ask some questions on the command line interactively.

7.4.2 Python API

At the moment please stick to the CLI tool!

7.5 Using environments

7.5.1 Command line

Most of the `mlonmcu` subcommands need a MLonMCU environment to operate on. In some cases it can be resolved automatically however it is recommended to pass it explicitly by the user in either of the following ways:

- Point the `MLONMCU_HOME` environment variable to the environment directory which should be used.
- Use the command line flags `-H` (`--home` or `--hint`) to provide either the path or (if available) the registered name of the environment.

If none of this was specified, MLonMCU will first look for a valid environment in the current working directory and else fall back to the default environment of the current user (if configured).

Example usage:

```
export MLONMCU_HOME=/tmp/home
mlonmcu models
```

or

```
mlonmcu models -H myenv
```

or

```
mlonmcu models -H ./env/
```

7.5.2 Python API

For the best experience a MLonMCU environment should always we wrapped with a `MlonMcuContext` as it provides useful utilities and a locking mechanism which can ensure that only one instance of the environment can be requested at a time.

The typical using using a Python `with` block looks as follows:

```
from mlonmcu.context.context import MlonMcuContext

with MlonMcuContext() as ctx:
    pass
```

Analogous to the command line flags an environment path or name should be provided to use a non-default environment location.

7.6 Environment registry

Optionally environment can be registered in the users home config directory with a given name which enabled referring to them without providing a file path. Use the `--register` and `--name` flags of the `mlonmcu init` command to do so. The command `mlonmcu env` provides useful utilities to list and modify existing entries in the registry file which is typically located at `~/.config/mlonmcu/environments.ini`.

CHAPTER

EIGHT

POSTPROCESSES

Postprocesses are a feature which was recently added to MLonMCU which is intended to help with automating common tasks for benchmarking and visualizations. These processes mainly operate on the Rows and columns of the report generated after a completed run or session. Their complexity can vary from very minimal utilities to powerful evaluation scripts.

8.1 Stages

Currently postprocesses can be applied at two different stages:

- after a `Run` which means you are only operating on a single row of a report
- or after a `Session` which has a variety of rows and columns

In the future it might be possible to also insert postprocesses at earlier stages.

8.2 Examples

- `AverageCyclesPostprocess`: average over a number of similar runs (useful for non-deterministic targets)
- `DetailedCyclesPostprocess`: determine the number of cycles required for the model initialization and invocation from two runs with a different `--num` value
- `FlattenConfig`: Turn each dictionary item in the `Config` column into a new Column which makes their values filterable
- `FlattenFeatures`: Convert the list of enabled features into one column per feature with boolean values in every row

8.3 Usage of Postprocesses

8.3.1 Implement custom postprocesses

To extend the list of predefined post-processes with a custom implementation a Python class has to be developed as follows:

```
TODO

from mlonmcu.postproces import Postprocess, ProcessStage
```

(continues on next page)

(continued from previous page)

```
class MyPostprocess(Postprocess):

    def __init__(self, features=None, config=None):
        super().__init__("foo", stage=ProcessStage.SESSION, features=features,␣
↪config=config)

    def process(self, data):
        pass
```

It is also possible to inherit another base-class such "AggregatePostprocess" to reuse some of their functionalities.

To use the newly implemented process, it needs to be registered. There are two approaches to do so:

- Call the registration function manually:

```
from mlonmcu.postprocess import register_postprocess

register_postprocess("foo", MyPostprocess)
```

- Use a decorator for the registration. However this option is only available when playing the new postprocess in `mlonmcu/postprocess/postprocesses.py` as the decorators are only processed in this file.

```
@register_postprocess("foo")
class MyPostprocess(Postprocess):
    ...
```

TODO: implement the registration process!

CHAPTER

NINE

LOGGING AND VERBOSITY

9.1 Command-line output

On the CLI-side of MLonMCU two flags are provided to customized the verbosity:

- `--verbose` or `-v` sets the used log level from `INFO` to `DEBUG` for more detailed outputs
- `--quiet` or `-q` sets the used log level from `INFO` to `WARNING` to have minimal information printed to the command line

Of course these flags can **not** be used together!

9.2 Debugging Errors and specific components

MLonMCUs logging output is designed to be very clean so user doe not have to deal with the things going on in the background. However in terms of an error a full stack trace of the exception which was raised is provided to ease debugging. Using the `--verbose` flag sets the loggers level to `DEBUG` which additionally print some useful information on the commands being executed etc. If required, most components feature a config such as `mlif.print_output` or `{backend_name}.print_output` which redirects all of its outputs to the command line.

9.3 Writing log files

Is is possible to additionally write the messages produced by the MLonMCU logger to a log file in a user-specified directory. This feature can be enabled in the `environment.yml` file using the following options:

```
logging:
  level: DEBUG
  to_file: true
  rotate: false
```

By default a directory called `logs` in the environment directory is used, but this can be overwritten by the user itself. Enabling the `rotate` option may be helpful as well as it makes it easier to find logs related to a certain date. The log level configured in the environment file does only affect the logging to the file and not the command line output.

9.4 Background

MLonMCUs logging is based on the `logging` package, however it comes with a set of functions which need to be executed to initialized the logger class. For this reason someone should always use the `get_logger` function from `mlonmcu.logging` instead the official one.

CHAPTER

TEN

DOCUMENTATION

10.1 mlonmcu

10.1.1 mlonmcu package

Subpackages

mlonmcu.cli package

Subpackages

mlonmcu.cli.helper package

Submodules

mlonmcu.cli.helper.filter module

mlonmcu.cli.helper.filter.**filter_arg**(*arg*)
TODO

mlonmcu.cli.helper.parse module

mlonmcu.cli.helper.parse.**extract_backend_names**(*args*, *context=None*)

mlonmcu.cli.helper.parse.**extract_config**(*args*)

mlonmcu.cli.helper.parse.**extract_config_and_feature_names**(*args*, *context=None*)

mlonmcu.cli.helper.parse.**extract_feature_names**(*args*)

mlonmcu.cli.helper.parse.**extract_frontend_names**(*args*, *context=None*)

mlonmcu.cli.helper.parse.**extract_platform_names**(*args*, *context=None*)

mlonmcu.cli.helper.parse.**extract_postprocess_names**(*args*, *context=None*)

mlonmcu.cli.helper.parse.**extract_target_names**(*args*, *context=None*)

mlonmcu.cli.helper.parse.**parse_var**(*s*)

Parse a key, value pair, separated by '=' That's the reverse of ShellArgs.

On the command line (argparse) a declaration will typically look like: foo=hello

or foo="hello world"

mlonmcu.cli.helper.parse.**parse_vars**(*items*)

Parse a series of key-value pairs and return a dictionary

Module contents

Submodules

mlonmcu.cli.build module

Command line subcommand for the build process.

mlonmcu.cli.build.**add_build_options**(*parser*)

mlonmcu.cli.build.**get_parser**(*subparsers*, *parent=None*)

"Define and return a subparser for the build subcommand.

mlonmcu.cli.build.**handle**(*args*, *ctx=None*, *require_target=False*)

mlonmcu.cli.cleanup module

Command line subcommand for cleaning up the current environment.

mlonmcu.cli.cleanup.**get_parser**(*subparsers*)

"Define and return a subparser for the cleanup subcommand.

mlonmcu.cli.cleanup.**handle**(*args*)

mlonmcu.cli.common module

mlonmcu.cli.common.**add_common_options**(*parser*)

mlonmcu.cli.common.**add_context_options**(*parser*, *with_home=True*)

mlonmcu.cli.common.**add_flow_options**(*parser*)

mlonmcu.cli.common.**add_model_options**(*parser*)

mlonmcu.cli.common.**handle_logging_flags**(*args*)

mlonmcu.cli.common.**kickoff_runs**(*args*, *until*, *context*)

mlonmcu.cli.compile module

Command line subcommand for the run process.

mlonmcu.cli.compile.**add_compile_options**(*parser*)

mlonmcu.cli.compile.**check_args**(*context*, *args*)

mlonmcu.cli.compile.**get_parser**(*subparsers*)

"Define and return a subparser for the compile subcommand.

mlonmcu.cli.compile.**handle**(*args*, *ctx=None*)

mlonmcu.cli.env module

Command line subcommand for manaing environments.

class mlonmcu.cli.env.**EnvironmentHint**(*name*, *path*, *created_at=None*)

Bases: object

mlonmcu.cli.env.**get_parser**(*subparsers*)

"Define and return a subparser for the env subcommand.

mlonmcu.cli.env.**handle**(*args*)

mlonmcu.cli.env.**lookup_user_environments**(*file*)

mlonmcu.cli.export module

Command line subcommand for exporting session and runs.

mlonmcu.cli.export.**add_export_options**(*parser*)

mlonmcu.cli.export.**get_parser**(*subparsers*)

"Define and return a subparser for the cleanup subcommand.

mlonmcu.cli.export.**handle**(*args*)

mlonmcu.cli.flow module

Command line subcommand for invoking the mlonmcu flow.

mlonmcu.cli.flow.**get_parser**(*subparsers*, *parent=None*)

"Define and return a subparser for the flow subcommand.

mlonmcu.cli.flow.**handle**(*args*)

Callback function which will be called to process the flow subcommand

mlonmcu.cli.flow.**handle_list_targets**(*args*)

mlonmcu.cli.init module

Command line subcommand for initializing a mlonmcu environment.

mlonmcu.cli.init.**add_init_options**(*parser*)

mlonmcu.cli.init.**get_parser**(*subparsers*)
"Define and return a subparser for the init subcommand.

mlonmcu.cli.init.**handle**(*args*)
Callback function which will be called to process the init subcommand

mlonmcu.cli.load module

Command line subcommand for the load stage.

mlonmcu.cli.load.**add_load_options**(*parser*)

mlonmcu.cli.load.**get_parser**(*subparsers*)
"Define and return a subparser for the load subcommand.

mlonmcu.cli.load.**handle**(*args*, *ctx=None*)

mlonmcu.cli.main module

Console script for mlonmcu.

mlonmcu.cli.main.**handle_docker**(*args*)

mlonmcu.cli.main.**main**(*args=None*)
Console script for mlonmcu.

mlonmcu.cli.models module

Command line subcommand for managing models.

mlonmcu.cli.models.**add_models_options**(*parser*)

mlonmcu.cli.models.**get_parser**(*subparsers*)
"Define and return a subparser for the models subcommand.

mlonmcu.cli.models.**handle**(*args*)

mlonmcu.cli.run module

Command line subcommand for the run process.

mlonmcu.cli.run.**add_run_options**(*parser*)

mlonmcu.cli.run.**check_args**(*context*, *args*)

mlonmcu.cli.run.**get_parser**(*subparsers*)
"Define and return a subparser for the run subcommand.

mlonmcu.cli.run.**handle**(*args*)

mlonmcu.cli.setup module

Command line subcommand for installing a mlonmcu environment.

mlonmcu.cli.setup.**add_setup_options**(*parser*)

mlonmcu.cli.setup.**get_parser**(*subparsers*)

"Define and return a subparser for the setup subcommand.

mlonmcu.cli.setup.**handle**(*args*)

mlonmcu.cli.tune module

Command line subcommand for the tune stage.

mlonmcu.cli.tune.**get_parser**(*subparsers*, *parent=None*)

"Define and return a subparser for the tune subcommand.

mlonmcu.cli.tune.**handle**(*args*, *ctx=None*)

Module contents

Submodule handling the command line interface for mlonmcu.

mlonmcu.context package

Submodules

mlonmcu.context.context module

Definition if the contextmanager for mlonmcu environments.

class mlonmcu.context.context.**MlonMcuContext**(*name: str = None*, *path: str = None*, *deps_lock: str = 'write'*)

Bases: object

Contextmanager for mlonmcu environments.

Attributes

environment [Environment] The MLonMCU Environment where paths, repos, features,... are configured.

deps_lock [str ("read" or "write" default "write")] Read means that the program does not write to the ./deps folder in the env folder.

sessions [list] List of sessions for the current environment.

session_idx [list] A counter for determining the next session index.

cache [TaskCache] The cache where paths of installed dependencies can be looked up.

cleanup()

Clean up the context before leaving the context by closing all active sessions

cleanup_sessions(*keep=10*, *interactive=True*)

Utility to cleanup old sessions from the disk.

create_session(*label=''*, *config=None*)

export(*dest*, *session_ids=None*, *run_ids=None*, *interactive=True*)

get_session(*label=''*, *resume=False*, *config=None*) → *mlonmcu.session.session.Session*

Get an active session if available, else create a new one.

Returns

Session: An active session

get_sessions_runs_idx()

property is_clean

Return true if all sessions in the context are inactive

load_cache()

If available load the cache.ini file in the deps directory

load_extensions()

If available load the extensions.py scripts in the plugin directories

print_summary(*sessions=True*, *runs=False*, *labels=True*)

mlonmcu.context.context.**get_environment_by_name**(*name: str*) → *mlonmcu.environment.environment.Environment*

Utility to find an environment file using a supplied name.

Parameters

name [str] The name/alias if the environment.

Returns

Environment The environment (if the lookup was successful).

mlonmcu.context.context.**get_environment_by_path**(*path: Union[str, pathlib.Path]*) → *mlonmcu.environment.environment.Environment*

Utility to find an environment file using a supplied path.

Parameters

path [str/Path] The path of the environment (or its YAML file).

Returns

Environment: The environment (if the lookup was successful).

mlonmcu.context.context.**get_ids**(*directory: pathlib.Path*) → List[int]

Get a sorted list of ids for sessions/runs found in the given directory.

Parameters

directory [Path] Directory where the sessions/runs are stored.

Returns:

list List of integers representing the session numbers. Empty list if directory does not exist.

mlonmcu.context.context.**load_recent_sessions**(*env:* mlonmcu.environment.environment.Environment, *count: int = None*) → List[*mlonmcu.session.session.Session*]

Get a list of recent sessions for the environment.

Parameters

env [Environment] MLonMCU environment which should be used.

count [int] Maximum number of sessions to return. Collect all if None.

Returns

list: The resulting list of session objects.

mlonmcu.context.context.**lookup_environment**() → *mlonmcu.environment.environment.Environment*

Helper function to automatically find a suitable environment.

This function is used if neither a name nor a path of the environment was specified by the user. The lookup follows a predefined order: - Check current working directory - Check MLONMCU_HOME environment variable - Default environment for current user

Returns

environment [Path] The environment (if the lookup was successful).

mlonmcu.context.context.**resolve_environment_file**(*name: str = None*, *path: str = None*) → pathlib.Path

Utility to find the environment file by a optionally given name or path.

The lookup is performed in a predefined order: - If specified: name/path - Else: see lookup_environment()

Parameters

name [str] Hint for the environment name provided by the user.

path [str] Hint for the environment path provided by the user.

Returns

Path Path to the found environment.yml (if sucessful)

mlonmcu.context.context.**setup_logging**(*environment*)

Check logging settings for environment and initialize the logs directory.

Attributes

environment [Environment] The MLonMCU Environment where paths, repos, features,... are configured.

mlonmcu.context.read_write_filelock module

This file contains read lock and write lock classes based on filelock The locks are non-blocking.

exception mlonmcu.context.read_write_filelock.**RWLockTimeout**(*lock*)

Bases: TimeoutError

Raised when the lock could not be acquired.

lock

The Read or Write lock instance.

class mlonmcu.context.read_write_filelock.**ReadFileLock**(*filepath*)

Bases: object

acquire(*raise_exception=True*)

This function tried to acquire a ReadFileLock. The process is the following: 1. acquire filelock 2. read the lock occupation situation 3. check if the lock is already occupied by another write process 4.1. release filelock and raise exception(or return 0) if the lock is already occupied by another write process 4.2. otherwise write the updated lock occupation situation back, release filelock and return

Parameters: raise_exception (bool): whether an exception should be raised when failed (default: True)

Returns:

success (bool): whether succeeded or not. True means succeeded, False means failed (if the param raiseException is set to False). A RWLockTimeout exception will be raised if failed (if the param raiseException is set to True).

property is_locked

This property returns if a lock is occupied(locked) by other processes. the process is the following: 1. acquire filelock 2. read the lock occupation situation 3. check if the lock is already occupied(locked) by another write process

Returns: is_locked (bool): whether the lock is already occupied(locked) by another write process

release()

This function releases a ReadFileLock. the process is the following: 1. acquire filelock 2. read the lock occupation situation 3. delete the record of self.id (no exception will be raised if self.id is not found in the record) 4. write the updated lock occupation situation back, release filelock and return

class mlonmcu.context.read_write_filelock.**WriteFileLock**(*filepath*)

Bases: object

acquire(*raise_exception=True*)

This function tried to acquire a WriteFileLock. The process is the following: 1. acquire filelock 2. read the lock occupation situation 3. check if the lock is already occupied by another write process 4.1. release filelock and raise exception(or return 0) if the lock is already occupied by another write process 4.2. otherwise write the updated lock occupation situation back, release filelock and return

Parameters: raise_exception (bool): whether raises an exception when failed (default: True)

Returns:

success (bool): whether succeeded or not. True means succeeded, False means failed (if the param raiseException is set to False). A RWLockTimeout exception will be raised if failed (if the param raiseException is set to True).

property is_locked

This property returns if a lock is occupied(locked) by other processes. the process is the following: 1. acquire filelock 2. read the lock occupation situation 3. check if the lock is already occupied(locked) by another write process

Returns: is_locked (bool): whether the lock is already occupied(locked) by another write process

release()

This function releases a WriteFileLock. the process is the following: 1. acquire filelock 2. read the lock occupation situation 3. delete the record of self.id (no exception will be raised if self.id is not found in the record) 4. write the updated lock occupation situation back, release filelock and return

Module contents

class mlonmcu.context.**MlonMcuContext**(*name: str = None*, *path: str = None*, *deps_lock: str = 'write'*)

Bases: object

Contextmanager for mlonmcu environments.

Attributes

environment [Environment] The MLonMCU Environment where paths, repos, features,... are configured.

deps_lock [str ("read" or "write" default "write")] Read means that the program does not write to the ./deps folder in the env folder.

sessions [list] List of sessions for the current environment.

session_idx [list] A counter for determining the next session index.

cache [TaskCache] The cache where paths of installed dependencies can be looked up.

cleanup()

Clean up the context before leaving the context by closing all active sessions

cleanup_sessions(*keep=10*, *interactive=True*)

Utility to cleanup old sessions from the disk.

create_session(*label=''*, *config=None*)

export(*dest*, *session_ids=None*, *run_ids=None*, *interactive=True*)

get_session(*label=''*, *resume=False*, *config=None*) → *mlonmcu.session.session.Session*

Get an active session if available, else create a new one.

Returns

Session: An active session

get_sessions_runs_idx()

property is_clean

Return true if all sessions in the context are inactive

load_cache()

If available load the cache.ini file in the deps directory

load_extensions()

If available load the extensions.py scripts in the plugin directories

print_summary(*sessions=True*, *runs=False*, *labels=True*)

mlonmcu.environment package

Submodules

mlonmcu.environment.config module

class mlonmcu.environment.config.**BackendConfig**(*name*, *enabled=True*, *features={}*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**BackendFeatureConfig**(*name*, *backend*, *supported=True*)

Bases: *mlonmcu.environment.config.FeatureConfig*

class mlonmcu.environment.config.**BaseConfig**

Bases: object

class mlonmcu.environment.config.**DefaultsConfig**(*log_level=20*, *log_to_file=False*, *log_rotate=False*, *default_framework=None*, *default_backends={}*, *default_target=None*, *cleanup_auto=False*, *cleanup_keep=100*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**FeatureConfig**(*name*, *kind=FeatureKind.UNKNOWN*, *supported=True*)

Bases: object

class mlonmcu.environment.config.**FeatureKind**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: enum.Enum

BACKEND = 2

FRAMEWORK = 1

FRONTEND = 4

TARGET = 3

UNKNOWN = 0

class mlonmcu.environment.config.**FrameworkConfig**(*name*, *enabled=True*, *backends={}*, *features={}*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**FrameworkFeatureConfig**(*name*, *framework*, *supported=True*)

Bases: *mlonmcu.environment.config.FeatureConfig*

class mlonmcu.environment.config.**FrontendConfig**(*name*, *enabled=True*, *features={}*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**FrontendFeatureConfig**(*name*, *frontend*, *supported=True*)

Bases: *mlonmcu.environment.config.FeatureConfig*

class mlonmcu.environment.config.**PathConfig**(*path*, *base=None*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**PlatformConfig**(*name*, *enabled=True*, *features={}*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**PlatformFeatureConfig**(*name*, *platform*, *supported=True*)

Bases: *mlonmcu.environment.config.FeatureConfig*

class mlonmcu.environment.config.**RepoConfig**(*url*, *ref=None*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**TargetConfig**(*name*, *enabled=True*, *features={}*)

Bases: *mlonmcu.environment.config.BaseConfig*

class mlonmcu.environment.config.**TargetFeatureConfig**(*name*, *target*, *supported=True*)

Bases: *mlonmcu.environment.config.FeatureConfig*

mlonmcu.environment.config.**get_config_dir**()

mlonmcu.environment.config.**get_environments_dir**()

mlonmcu.environment.config.**get_environments_file**()

mlonmcu.environment.config.**get_plugins_dir**()

mlonmcu.environment.config.**init_config_dir**()

mlonmcu.environment.environment module

class mlonmcu.environment.environment.**DefaultEnvironment**

Bases: *mlonmcu.environment.environment.Environment*

class mlonmcu.environment.environment.**Environment**

Bases: object

classmethod from_file(*filename*)

get_default_backends(*framework*)

get_default_frameworks()

get_default_targets()

has_backend(*name*)

has_feature(*name*)

An alias for supports_feature.

has_framework(*name*)

has_frontend(*name*)

has_platform(*name*)

has_target(*name*)

has_toolchain(*name*)

property home

Home directory of mlonmcu environment.

lookup_backend_configs(*backend=None*, *framework=None*, *names_only=False*)

lookup_backend_feature_configs(*name=None*, *framework=None*, *backend=None*)

lookup_feature_configs(*name=None*, *kind=None*, *frontend=None*, *framework=None*, *backend=None*, *platform=None*, *target=None*)

lookup_framework_configs(*framework=None*, *names_only=False*)

lookup_framework_feature_configs(*name=None*, *framework=None*)

lookup_frontend_configs(*frontend=None*, *names_only=False*)

lookup_frontend_feature_configs(*name=None*, *frontend=None*)

lookup_path(*name*)

lookup_platform_configs(*platform=None*, *names_only=False*)

lookup_platform_feature_configs(*name=None*, *platform=None*)

lookup_target_configs(*target=None*, *names_only=False*)

lookup_target_feature_configs(*name=None*, *target=None*)

lookup_var(*name*, *default=None*)

supports_feature(*name*)

to_file(*filename*)

class mlonmcu.environment.environment.**UserEnvironment**(*home*, *merge=False*, *alias=None*, *defaults=None*, *paths=None*, *repos=None*, *frameworks=None*, *frontends=None*, *platforms=None*, *toolchains=None*, *targets=None*, *variables=None*, *default_flags=None*)

Bases: *mlonmcu.environment.environment.DefaultEnvironment*

mlonmcu.environment.init module

mlonmcu.environment.init.**clone_models_repo**(*dest*, *url='https://github.com/tum-ei-eda/mlonmcu-models.git'*)

mlonmcu.environment.init.**create_environment_directories**(*path*, *directories*)

mlonmcu.environment.init.**create_venv_directory**(*base*, *hidden=True*)

mlonmcu.environment.init.**initialize_environment**(*directory*, *name*, *interactive=True*, *create_venv=None*, *clone_models=None*, *allow_exists=None*, *register=None*, *template=None*, *config=None*)

mlonmcu.environment.list module

mlonmcu.environment.list.**get_alternative_name**(*name*, *names*)

mlonmcu.environment.list.**get_environment_names**()

mlonmcu.environment.list.**get_environments_map**()

mlonmcu.environment.list.**register_environment**(*name*, *path*, *overwrite=False*)

mlonmcu.environment.list.**validate_name**(*name*)

mlonmcu.environment.loader module

mlonmcu.environment.loader.**load_environment_from_file**(*filename*, *base*)
Utility to initialize a mlonmcu environment from a YAML file.

mlonmcu.environment.templates module

Definitions of mlonmcu config templates.

mlonmcu.environment.templates.**fill_environment_yaml**(*template_name*, *home_dir*, *config=None*)

mlonmcu.environment.templates.**fill_template**(*name*, *data={}*)

mlonmcu.environment.templates.**get_template_names**()

mlonmcu.environment.templates.**get_template_text**(*name*)

mlonmcu.environment.templates.**write_environment_yaml_from_template**(*path*, *template_name*, *home_dir*, *config=None*)

mlonmcu.environment.writer module

mlonmcu.environment.writer.**create_environment_dict**(*environment*)

mlonmcu.environment.writer.**write_environment_to_file**(*environment*, *filename*)
Utility to initialize a mlonmcu environment from a YAML file.

Module contents

mlonmcu.feature package

Submodules

mlonmcu.feature.feature module

MLonMCU Features API

class mlonmcu.feature.feature.**BackendFeature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Backend related feature

add_backend_config(*backend*, *config*)

feature_type = 4

get_backend_config(*backend*)

class mlonmcu.feature.feature.**Feature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Feature of unknown type

feature_type = 0

class mlonmcu.feature.feature.**FeatureBase**(*name*, *features=None*, *config=None*)

Bases: abc.ABC

Feature base class

DEFAULTS = {'enabled': True}

OPTIONAL = []

REQUIRED = []

property enabled

feature_type = None

remove_config_prefix(*config*)

scope = None

classmethod types()

Find out which types the features is based on.

class mlonmcu.feature.feature.**FrameworkFeature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Framework related feature

add_framework_config(*framework*, *config*)

feature_type = 3

get_framework_config(*framework*)

class mlonmcu.feature.feature.**FrontendFeature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Frontend related feature

add_frontend_config(*frontend*, *config*)

feature_type = 2

get_frontend_config(*frontend*)

update_formats(*frontend*, *input_formats*, *output_formats*)

class mlonmcu.feature.feature.**PlatformFeature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Platform/Compile related feature

add_platform_config(*platform*, *config*)

add_platform_defs(*platform*, *defs*)

feature_type = 6

get_platform_config(*platform*)

get_platform_defs(*platform*)

class mlonmcu.feature.feature.**RunFeature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Run related feature

add_run_config(*config*)

feature_type = 7

get_run_config()

class mlonmcu.feature.feature.**SetupFeature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Setup/Cache related feature

add_required_cache_flags(*required_flags*)

add_setup_config(*config*)

feature_type = 1

get_required_cache_flags()

get_setup_config()

class mlonmcu.feature.feature.**TargetFeature**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.feature.feature.FeatureBase*

Target related feature

add_target_callbacks(*target*, *pre_callbacks*, *post_callbacks*)

add_target_config(*target*, *config*)

feature_type = 5

get_target_callbacks(*target*)

get_target_config(*target*)

mlonmcu.feature.features module

Definition of MLonMCU features and the feature registry.

mlonmcu.feature.features.**filter_none**(*data*)

Helper function which drop dict items with a None value.

mlonmcu.feature.features.**get_available_feature_names**(*feature_type=None*)

Utility for getting feature names.

mlonmcu.feature.features.**get_available_features**(*feature_type=None*, *feature_name=None*)

Utility for looking up features.

mlonmcu.feature.features.**get_matching_features**(*features*, *feature_type*)

mlonmcu.feature.features.**register_feature**(*name*)

Decorator for adding a feature to the global registry.

mlonmcu.feature.type module

class mlonmcu.feature.type.**FeatureType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: enum.Enum

Enumeration for Feature types.

BACKEND = 4

FRAMEWORK = 3

FRONTEND = 2

OTHER = 0

PLATFORM = 6

RUN = 7

SETUP = 1

TARGET = 5

Module contents

mlonmcu.flow package

Subpackages

mlonmcu.flow.tflm package

Subpackages

mlonmcu.flow.tflm.backend package

Submodules

mlonmcu.flow.tflm.backend.backend module

class mlonmcu.flow.tflm.backend.backend.**TFLMBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.backend.Backend*

DEFAULTS = {}

REQUIRED = []

load_model(*model*, *input_shapes=None*, *output_shapes=None*, *input_types=None*, *output_types=None*)

name = None

registry = {'tflmc': <class 'mlonmcu.flow.tflm.backend.tflmc.TFLMCBackend'>, 'tflmi': <class 'mlonmcu.flow.tflm.backend.tflmi.TFLMIBackend'>}

mlonmcu.flow.tflm.backend.tflmc module

class mlonmcu.flow.tflm.backend.tflmc.**TFLMCBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.tflm.backend.backend.TFLMBackend*

DEFAULTS = {'custom_ops': [], 'debug_arena': False, 'print_outputs': False, 'registrations': {}}

FEATURES = ['debug_arena']

REQUIRED = ['tflmc.exe']

generate() → Tuple[dict, dict]

generate_header()

name = 'tflmc'

property print_outputs

mlonmcu.flow.tflm.backend.tflmi module

class mlonmcu.flow.tflm.backend.tflmi.**TFLMIBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.tflm.backend.backend.TFLMBackend*

DEFAULTS = {'arena_size': 1048576, 'custom_ops': [], 'debug_arena': False, 'legacy': False, 'ops': [], 'ops_resolver': 'mutable', 'registrations': {}}

REQUIRED = []

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

property legacy

name = 'tflmi'

class mlonmcu.flow.tflm.backend.tflmi.**TFLMICodegen**

Bases: object

generate_header(*prefix='model'*)

generate_wrapper(*model*, *prefix='model'*, *header=True*, *legacy=False*, *debug_arena=False*, *arena_size=None*, *ops=None*, *custom_ops=None*, *registrations=None*, *ops_resolver=None*)

makeCustomOpPrototypes(*custom_ops*)

make_op_registrations(*ops*, *custom_ops*)

mlonmcu.flow.tflm.backend.tflmi.**make_hex_array**(*data*)

Module contents

class mlonmcu.flow.tflm.backend.**TFLMBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.backend.Backend*

DEFAULTS = {}

REQUIRED = []

load_model(*model*, *input_shapes=None*, *output_shapes=None*, *input_types=None*, *output_types=None*)

name = None

registry = {'tflmc': <class 'mlonmcu.flow.tflm.backend.tflmc.TFLMCBackend'>, 'tflmi': <class 'mlonmcu.flow.tflm.backend.tflmi.TFLMIBackend'>}

class mlonmcu.flow.tflm.backend.**TFLMCBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.tflm.backend.backend.TFLMBackend*

DEFAULTS = {'custom_ops': [], 'debug_arena': False, 'print_outputs': False, 'registrations': {}}

FEATURES = ['debug_arena']

REQUIRED = ['tflmc.exe']

generate() → Tuple[dict, dict]

generate_header()

name = 'tflmc'

property print_outputs

class mlonmcu.flow.tflm.backend.**TFLMIBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.tflm.backend.backend.TFLMBackend*

DEFAULTS = {'arena_size': 1048576, 'custom_ops': [], 'debug_arena': False, 'legacy': False, 'ops': [], 'ops_resolver': 'mutable', 'registrations': {}}

REQUIRED = []

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

property legacy

name = 'tflmi'

Submodules

mlonmcu.flow.tflm.framework module

Definitions for TFLMFramework.

class mlonmcu.flow.tflm.framework.**TFLMFramework**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.framework.Framework*

TFLM Framework specialization.

DEFAULTS = {'optimized_kernel': None, 'optimized_kernel_inc_dirs': [], 'optimized_kernel_libs': []}

FEATURES = ['muriscvnn', 'cmsisnn']

REQUIRED = ['tf.src_dir']

backends = {'tflmc': <class 'mlonmcu.flow.tflm.backend.tflmc.TFLMCBackend'>, 'tflmi': <class 'mlonmcu.flow.tflm.backend.tflmi.TFLMIBackend'>}

get_platform_defs(*platform*)

name = 'tflm'

property optimized_kernel

property optimized_kernel_inc_dirs

property optimized_kernel_libs

property tf_src

Module contents

TFLite framework module.

class mlonmcu.flow.tflm.**TFLMBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.backend.Backend*

DEFAULTS = {}

REQUIRED = []

load_model(*model*, *input_shapes=None*, *output_shapes=None*, *input_types=None*, *output_types=None*)

name = None

registry = {'tflmc': <class 'mlonmcu.flow.tflm.backend.tflmc.TFLMCBackend'>, 'tflmi': <class 'mlonmcu.flow.tflm.backend.tflmi.TFLMIBackend'>}

class mlonmcu.flow.tflm.**TFLMCBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.tflm.backend.backend.TFLMBackend*

DEFAULTS = {'custom_ops': [], 'debug_arena': False, 'print_outputs': False, 'registrations': {}}

FEATURES = ['debug_arena']

REQUIRED = ['tflmc.exe']

generate() → Tuple[dict, dict]

generate_header()

name = 'tflmc'

property print_outputs

class mlonmcu.flow.tflm.**TFLMIBackend**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.tflm.backend.backend.TFLMBackend*

DEFAULTS = {'arena_size': 1048576, 'custom_ops': [], 'debug_arena': False, 'legacy': False, 'ops': [], 'ops_resolver': 'mutable', 'registrations': {}}

REQUIRED = []

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

property legacy

name = 'tflmi'

mlonmcu.flow.tvm package

Subpackages

mlonmcu.flow.tvm.backend package

Submodules

mlonmcu.flow.tvm.backend.backend module

class mlonmcu.flow.tvm.backend.backend.**TVMBackend**(*target='c'*, *executor=None*, *runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.backend.Backend*

DEFAULTS = {'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt']

OPTIONAL = ['tvm.build_dir', 'tvm.pythonpath', 'tvm.configs_dir', 'tvm.use_tlcpack']

REQUIRED = []

property desired_layout

property disabled_passes

property extra_target

property extra_target_mcpu

get_extra_target_details()

get_target_details()

get_tvmc_compile_args(*out*, *dump=None*)

invoke_tvmc(*command*, **args*, *cwd=None*)

invoke_tvmc_compile(*out*, *dump=None*, *cwd=None*)

load_model(*model*, *input_shapes=None*, *output_shapes=None*, *input_types=None*, *output_types=None*)

name = None

num_threads()

property opt_level

property pass_config

property print_outputs

registry = {}

property target_device

property target_mabi

property target_march

property target_mattr

property target_mcpu

property target_model

property target_mtriple

property tophub_url

property tuning_records

property tvm_build_dir

property tvm_configs_dir

property tvm_pythonpath

property tvmc_custom_script

property tvmc_extra_args

property use_tlcpack

property use_tuning_results

mlonmcu.flow.tvm.backend.model_info module

class mlonmcu.flow.tvm.backend.model_info.**ModelInfo**(*in_tensors*, *out_tensors*, *fix_names=False*)

Bases: object

property has_ins

property has_outs

class mlonmcu.flow.tvm.backend.model_info.**ONNXModelInfo**(*model_file*)

Bases: *mlonmcu.flow.tvm.backend.model_info.ModelInfo*

class mlonmcu.flow.tvm.backend.model_info.**PBModelInfo**(*model_file*)

Bases: *mlonmcu.flow.tvm.backend.model_info.ModelInfo*

class mlonmcu.flow.tvm.backend.model_info.**PaddleModelInfo**(*model_file*)

Bases: *mlonmcu.flow.tvm.backend.model_info.ModelInfo*

class mlonmcu.flow.tvm.backend.model_info.**RelayModelInfo**(*mod_text*, *fix_names=False*)

Bases: *mlonmcu.flow.tvm.backend.model_info.ModelInfo*

class mlonmcu.flow.tvm.backend.model_info.**RelayTensorInfo**(*name*, *shape*, *dtype*, *fix_names=False*)

Bases: *mlonmcu.flow.tvm.backend.model_info.TensorInfo*

class mlonmcu.flow.tvm.backend.model_info.**TensorInfo**(*name*, *shape*, *dtype*, *fix_names=False*)

Bases: object

property size

class mlonmcu.flow.tvm.backend.model_info.**TfLiteModelInfo**(*model*, *fix_names=False*)

Bases: *mlonmcu.flow.tvm.backend.model_info.ModelInfo*

class mlonmcu.flow.tvm.backend.model_info.**TfLiteTensorInfo**(*t*, *fix_names=False*)

Bases: *mlonmcu.flow.tvm.backend.model_info.TensorInfo*

mlonmcu.flow.tvm.backend.model_info.**get_fallback_model_info**(*model*, *input_shapes*, *output_shapes*, *input_types*, *output_types*, *backend_name='unknown'*)

mlonmcu.flow.tvm.backend.model_info.**get_model_format**(*model*)

mlonmcu.flow.tvm.backend.model_info.**get_model_info**(*model*, *backend_name='unknown'*)

mlonmcu.flow.tvm.backend.model_info.**get_onnx_model_info**(*model_file*)

mlonmcu.flow.tvm.backend.model_info.**get_paddle_model_info**(*model_file*)

mlonmcu.flow.tvm.backend.model_info.**get_pb_model_info**(*model_file*)

mlonmcu.flow.tvm.backend.model_info.**get_relay_model_info**(*mod_text*)

mlonmcu.flow.tvm.backend.model_info.**get_supported_formats**()

mlonmcu.flow.tvm.backend.model_info.**get_tfgraph_inout**(*graph*)

mlonmcu.flow.tvm.backend.model_info.**get_tflite_model_info**(*model_buf*)

mlonmcu.flow.tvm.backend.model_info.**parse_relay_main**(*line*)

mlonmcu.flow.tvm.backend.model_info.**shape_from_str**(*shape_str*)

mlonmcu.flow.tvm.backend.python_utils module

mlonmcu.flow.tvm.backend.python_utils.**prepare_python_environment**(*pythonpath*, *tvm_build_dir*, *tvm_configs_dir*, *tophub_url=None*, *num_threads=None*)

mlonmcu.flow.tvm.backend.tuner module

class mlonmcu.flow.tvm.backend.tuner.**TVMTuner**(*backend*, *config=None*)

Bases: `object`

DEFAULTS = {'append': None, 'early_stopping': None, 'enable': False, 'max_parallel': 1, 'mode': 'autotvm', 'num_workers': 1, 'print_outputs': False, 'results_file': None, 'tasks': None, 'timeout': 100, 'trials': 10, 'tuner': 'ga', 'use_rpc': False, 'visualize': False}

mlonmcu.flow.tvm.backend.tvmaot module

class mlonmcu.flow.tvm.backend.tvmaot.**TVMAOTBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'alignment_bytes': 4, 'arena_size': None, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'unpacked_api': False, 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena', 'unpacked_api', 'usmp']

property alignment_bytes

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

get_tvmc_compile_args(*out*, *dump=None*)

get_workspace_size_from_metadata(*metadata*)

name = 'tvmaot'

property unpacked_api

mlonmcu.flow.tvm.backend.tvmaotplus module

class mlonmcu.flow.tvm.backend.tvmaotplus.**TVMAOTPlusBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.tvmaot.TVMAOTBackend*

DEFAULTS = {'alignment_bytes': 4, 'arena_size': 0, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {'tir.usmp.algorithm': 'hill_climb', 'tir.usmp.enable': True}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'unpacked_api': True, 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena', 'unpacked_api', 'usmp']

name = 'tvmaotplus'

mlonmcu.flow.tvm.backend.tvmc_utils module

mlonmcu.flow.tvm.backend.tvmc_utils.**check_allowed**(*target*, *name*)

mlonmcu.flow.tvm.backend.tvmc_utils.**gen_target_details_args**(*target*, *target_details*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_bench_tvmc_args**(*print_time=False*, *profile=False*, *end_to_end=False*, *repeat=1*, *number=1*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_data_tvmc_args**(*mode=None*, *ins_file=None*, *outs_file=None*, *print_top=10*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_disabled_pass_tvmc_args**(*disabled_passes*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_input_shapes_tvmc_args**(*input_shapes*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_pass_config_tvmc_args**(*pass_config*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_rpc_tvmc_args**(*enabled*, *key*, *hostname*, *port*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_runtime_executor_tvmc_args**(*runtime*, *executor*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_target_tvmc_args**(*target='c'*, *extra_target=None*, *target_details={}*, *extra_target_details=None*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_tuning_records_tvmc_args**(*use_tuning_results*, *tuning_records_file*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_tvmaot_tvmc_args**(*alignment_bytes*, *unpacked_api*)

mlonmcu.flow.tvm.backend.tvmc_utils.**get_tvmrt_tvmc_args**(*runtime='crt'*)

mlonmcu.flow.tvm.backend.tvmcg module

class mlonmcu.flow.tvm.backend.tvmcg.**TVMCGBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.tvmrt.TVMRTBackend*

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt']

REQUIRED = ['utvmcg.exe']

generate() → Tuple[dict, dict]

get_max_workspace_size_from_metadata(*metadata*)

name = 'tvmcg'

mlonmcu.flow.tvm.backend.tvmllvm module

class mlonmcu.flow.tvm.backend.tvmllvm.**TVMLLVMBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt']

generate(*verbose=False*) → Tuple[dict, dict]

get_graph_and_params_from_mlf(*path*)

get_tvmc_compile_args(*out*, *dump=None*)

name = 'tvmllvm'

mlonmcu.flow.tvm.backend.tvmrt module

class mlonmcu.flow.tvm.backend.tvmrt.**TVMRTBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'arena_size': 1048576, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena']

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

get_graph_and_params_from_mlf(*path*)

get_tvmc_compile_args(*out*, *dump=None*)

name = 'tvmrt'

mlonmcu.flow.tvm.backend.wrapper module

TODO

mlonmcu.flow.tvm.backend.wrapper.**calc_pages**(*workspace_size*, *page_size=1024*)

mlonmcu.flow.tvm.backend.wrapper.**fill**(*template*, ***kwargs*)

mlonmcu.flow.tvm.backend.wrapper.**generate_aot_includes**(*allocator*)

mlonmcu.flow.tvm.backend.wrapper.**generate_common_includes**()

mlonmcu.flow.tvm.backend.wrapper.**generate_graph_includes**()

mlonmcu.flow.tvm.backend.wrapper.**generate_header**()

mlonmcu.flow.tvm.backend.wrapper.**generate_tvmaot_wrapper**(*model_info*, *workspace_size*, *mod_name*, *api='c'*, *debug_arena=False*)

mlonmcu.flow.tvm.backend.wrapper.**generate_tvmrt_wrapper**(*graph*, *params*, *model_info*, *workspace_size*, *debug_arena=False*)

mlonmcu.flow.tvm.backend.wrapper.**generate_wrapper_header**()

mlonmcu.flow.tvm.backend.wrapper.**getSizes**(*tensors*)

mlonmcu.flow.tvm.backend.wrapper.**write_tvmaot_wrapper**(*path*, *model_info*, *workspace_size*, *mod_name*, *api='c'*)

mlonmcu.flow.tvm.backend.wrapper.**write_tvmrt_wrapper**(*path*, *graph*, *params*, *model_info*, *workspace_size*)

Module contents

class mlonmcu.flow.tvm.backend.**TVMAOTBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'alignment_bytes': 4, 'arena_size': None, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'unpacked_api': False, 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena', 'unpacked_api', 'usmp']

property alignment_bytes

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

get_tvmc_compile_args(*out*, *dump=None*)

get_workspace_size_from_metadata(*metadata*)

name = 'tvmaot'

property unpacked_api

class mlonmcu.flow.tvm.backend.**TVMAOTPlusBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.tvmaot.TVMAOTBackend*

DEFAULTS = {'alignment_bytes': 4, 'arena_size': 0, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {'tir.usmp.algorithm': 'hill_climb', 'tir.usmp.enable': True}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'unpacked_api': True, 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena', 'unpacked_api', 'usmp']

name = 'tvmaotplus'

class mlonmcu.flow.tvm.backend.**TVMBackend**(*target='c'*, *executor=None*, *runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.backend.Backend*

DEFAULTS = {'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt']

OPTIONAL = ['tvm.build_dir', 'tvm.pythonpath', 'tvm.configs_dir', 'tvm.use_tlcpack']

REQUIRED = []

property desired_layout

property disabled_passes

property extra_target

property extra_target_mcpu

get_extra_target_details()

get_target_details()

get_tvmc_compile_args(*out*, *dump=None*)

invoke_tvmc(*command*, **args*, *cwd=None*)

invoke_tvmc_compile(*out*, *dump=None*, *cwd=None*)

load_model(*model*, *input_shapes=None*, *output_shapes=None*, *input_types=None*, *output_types=None*)

name = None

num_threads()

property opt_level

property pass_config

property print_outputs

registry = {}

property target_device

property target_mabi

property target_march

property target_mattr

property target_mcpu

property target_model

property target_mtriple

property tophub_url

property tuning_records

property tvm_build_dir

property tvm_configs_dir

property tvm_pythonpath

property tvmc_custom_script

property tvmc_extra_args

property use_tlcpack

property use_tuning_results

class mlonmcu.flow.tvm.backend.**TVMCGBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.tvmrt.TVMRTBackend*

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt']

REQUIRED = ['utvmcg.exe']

generate() → Tuple[dict, dict]

get_max_workspace_size_from_metadata(*metadata*)

name = 'tvmcg'

class mlonmcu.flow.tvm.backend.**TVMLLVMBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt']

generate(*verbose=False*) → Tuple[dict, dict]

get_graph_and_params_from_mlf(*path*)

get_tvmc_compile_args(*out*, *dump=None*)

name = 'tvmllvm'

class mlonmcu.flow.tvm.backend.**TVMRTBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'arena_size': 1048576, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena']

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

get_graph_and_params_from_mlf(*path*)

get_tvmc_compile_args(*out*, *dump=None*)

name = 'tvmrt'

Submodules

mlonmcu.flow.tvm.framework module

Definitions for TVMFramework.

class mlonmcu.flow.tvm.framework.**TVMFramework**(*features=None*, *config=None*)

Bases: *mlonmcu.flow.framework.Framework*

TVM Framework specialization.

DEFAULTS = {'crt_config_dir': '/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/mlonmcu/../resources/frameworks/tvm/crt_config', 'extra_incs': [], 'extra_libs': []}

FEATURES = ['cmsisnnbyoc', 'muriscvnnbyoc']

REQUIRED = ['tvm.src_dir']

property crt_config_dir

property extra_incs

property extra_libs

get_platform_defs(*platform*)

name = 'tvm'

property tvm_src

mlonmcu.flow.tvm.framework.**get_crt_config_dir**()

Module contents

TVM framework module.

class mlonmcu.flow.tvm.**TVMAOTBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'alignment_bytes': 4, 'arena_size': None, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'unpacked_api': False, 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena', 'unpacked_api', 'usmp']

property alignment_bytes

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

get_tvmc_compile_args(*out*, *dump=None*)

get_workspace_size_from_metadata(*metadata*)

name = 'tvmaot'

property unpacked_api

class mlonmcu.flow.tvm.**TVMAOTPlusBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.tvmaot.TVMAOTBackend*

DEFAULTS = {'alignment_bytes': 4, 'arena_size': 0, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {'tir.usmp.algorithm': 'hill_climb', 'tir.usmp.enable': True}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'unpacked_api': True, 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena', 'unpacked_api', 'usmp']

name = 'tvmaotplus'

class mlonmcu.flow.tvm.**TVMCGBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.tvmrt.TVMRTBackend*

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt']

REQUIRED = ['utvmcg.exe']

generate() → Tuple[dict, dict]

get_max_workspace_size_from_metadata(*metadata*)

name = 'tvmcg'

class mlonmcu.flow.tvm.**TVMRTBackend**(*runtime='crt'*, *fmt='mlf'*, *features=None*, *config=None*)

Bases: *mlonmcu.flow.tvm.backend.backend.TVMBackend*

DEFAULTS = {'arena_size': 1048576, 'debug_arena': False, 'desired_layout': None, 'disabled_passes': [], 'extra_pass_config': {}, 'extra_target': None, 'extra_target_mcpu': None, 'num_threads': 2, 'opt_level': 3, 'print_outputs': False, 'target_device': None, 'target_mabi': None, 'target_march': None, 'target_mattr': None, 'target_mcpu': None, 'target_model': None, 'target_mtriple': None, 'tophub_url': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_tuning_results': False}

FEATURES = ['autotuned', 'cmsisnnbyoc', 'muriscvnnbyoc', 'disable_legalize', 'moiopt', 'debug_arena']

property arena_size

property debug_arena

generate() → Tuple[dict, dict]

get_graph_and_params_from_mlf(*path*)

get_tvmc_compile_args(*out*, *dump=None*)

name = 'tvmrt'

Submodules

mlonmcu.flow.backend module

class mlonmcu.flow.backend.**Backend**(*framework=''*, *features=None*, *config=None*)

Bases: abc.ABC

DEFAULTS = {}

FEATURES = []

OPTIONAL = []

REQUIRED = []

add_platform_defs(*platform*, *defs*)

export_artifacts(*path*)

abstract generate() → Tuple[dict, dict]

generate_artifacts() → List[*mlonmcu.artifact.Artifact*]

get_platform_defs(*platform*)

property has_tuner

abstract load_model(*model*, *input_shapes=None*, *output_shapes=None*, *input_types=None*, *output_types=None*)

name = None

process_features(*features*)

set_tuning_records(*filepath*)

supports_model(*model*)

mlonmcu.flow.backend.**get_parser**(*backend_name*, *features*, *required*, *defaults*)

mlonmcu.flow.backend.**init_backend_features**(*names*, *config*)

mlonmcu.flow.backend.**main**(*backend*, *args=None*)

mlonmcu.flow.framework module

class mlonmcu.flow.framework.**Framework**(*features=None*, *config=None*, *backends={}*)

Bases: abc.ABC

DEFAULTS = {}

FEATURES = []

OPTIONAL = []

REQUIRED = ['tf.src_dir']

add_platform_defs(*platform*, *defs*)

get_platform_defs(*platform*)

name = None

process_features(*features*)

registry = {'tflm': <class 'mlonmcu.flow.tflm.framework.TFLMFramework'>, 'tvm': <class 'mlonmcu.flow.tvm.framework.TVMFramework'>}

remove_config_prefix(*config*)

Module contents

Flow module for frameworks and backend.

mlonmcu.flow.**get_available_backend_names**()
Return all available backend names.

mlonmcu.models package

Submodules

mlonmcu.models.convert_data module

mlonmcu.models.frontend module

class mlonmcu.models.frontend.**Frontend**(*name*, *input_formats=None*, *output_formats=None*, *features=None*, *config=None*)

Bases: abc.ABC

DEFAULTS = {'use_inout_data': False}

FEATURES = ['validate']

OPTIONAL = []

REQUIRED = []

export_artifacts(*path*)

generate(*model*) → Tuple[dict, dict]

generate_artifacts(*model*) → List[*mlonmcu.artifact.Artifact*]

process_features(*features*)

process_metadata(*model*, *cfg=None*)

abstract produce_artifacts(*model*)

supports_formats(*ins=None*, *outs=None*)
Returs true if the frontend can handle at least one combination of input and output formats.

property use_inout_data

class mlonmcu.models.frontend.**LayerGenFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.Frontend*

DEFAULTS = {'fmt': 'tflite', 'use_inout_data': False}

FEATURES = ['validate']

REQUIRED = ['layergen.exe']

property fmt

generate(*model*) → Tuple[dict, dict]

property layergen_exe

produce_artifacts(*model*)

class mlonmcu.models.frontend.**ONNXFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'use_inout_data': False}

FEATURES = ['validate']

REQUIRED = []

class mlonmcu.models.frontend.**PBFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'use_inout_data': False}

FEATURES = ['validate']

REQUIRED = []

class mlonmcu.models.frontend.**PackedFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.Frontend*

DEFAULTS = {'check': False, 'fake_pack': False, 'ignore_existing': True, 'use_inout_data': False, 'use_packed': True}

FEATURES = ['validate', 'packing', 'packed']

REQUIRED = ['packer.exe']

property check

property fake_pack

property ignore_existing

produce_artifacts(*model*)

property use_packed

class mlonmcu.models.frontend.**PaddleFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'use_inout_data': False}

FEATURES = ['validate']

REQUIRED = []

class mlonmcu.models.frontend.**RelayFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'relayviz_plotter': 'term', 'use_inout_data': False, 'visualize_graph': False}

FEATURES = ['validate', 'relayviz']

REQUIRED = ['tvm.build_dir', 'tvm.pythonpath']

produce_artifacts(*model*)

property relayviz_plotter

property tvm_build_dir

property tvm_pythonpath

property visualize_graph

class mlonmcu.models.frontend.**SimpleFrontend**(*name*, *fmt*, *features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.Frontend*

An abstract frontend with equivalent input and output formats.

produce_artifacts(*model*)

class mlonmcu.models.frontend.**TfLiteFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'pack_script': None, 'split_layers': False, 'use_inout_data': False, 'visualize_enable': False, 'visualize_script': None}

FEATURES = ['validate', 'visualize', 'split_layers']

REQUIRED = []

generate(*model*) → Tuple[dict, dict]

property pack_script

produce_artifacts(*model*)

property split_layers

property visualize_enable

property visualize_script

mlonmcu.models.group module

class mlonmcu.models.group.**ModelGroup**(*name*, *models*, *description=''*)

Bases: object

mlonmcu.models.lookup module

mlonmcu.models.lookup.**apply_modelgroups**(*models*, *context=None*)

mlonmcu.models.lookup.**find_metadata**(*directory*, *model_name=None*)

mlonmcu.models.lookup.**get_model_directories**(*context*)

mlonmcu.models.lookup.**list_modelgroups**(*directory*)

mlonmcu.models.lookup.**list_models**(*directory*, *depth=1*, *formats=None*, *config=None*)

mlonmcu.models.lookup.**lookup_models**(*names*, *frontends=None*, *context=None*)

mlonmcu.models.lookup.**lookup_models_and_groups**(*directories*, *formats*)

mlonmcu.models.lookup.**map_frontend_to_model**(*model*, *frontends*, *backend=None*)

mlonmcu.models.lookup.**print_groups**(*groups*, *all_models=[]*, *duplicates=[]*, *detailed=False*)

mlonmcu.models.lookup.**print_models**(*models*, *duplicates=[]*, *detailed=False*)

mlonmcu.models.lookup.**print_paths**(*directories*)

mlonmcu.models.lookup.**print_summary**(*context*, *detailed=False*)

mlonmcu.models.metadata module

mlonmcu.models.metadata.**parse_metadata**(*path*)

mlonmcu.models.model module

class mlonmcu.models.model.**Model**(*name*, *paths*, *config=None*, *alt=None*, *formats=ModelFormats.TFLITE*)

Bases: object

DEFAULTS = {'input_shapes': None, 'input_types': None, 'inputs_path': 'input', 'metadata_path': 'definition.yml', 'output_shapes': None, 'output_types': None, 'outputs_path': 'output', 'support_path': 'support'}

property input_shapes

property input_types

property inputs_path

property metadata_path

property output_shapes

property output_types

property outputs_path

property skip_check

property support_path

class mlonmcu.models.model.**ModelFormat**(*value*, *extensions*)

Bases: tuple

extensions

Alias for field number 1

value

Alias for field number 0

class mlonmcu.models.model.**ModelFormats**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: enum.Enum

IPYNB = ModelFormat(value=3, extensions=['ipynb'])

NONE = ModelFormat(value=0, extensions=[])

ONNX = ModelFormat(value=4, extensions=['onnx'])

PACKED = ModelFormat(value=2, extensions=['tflm'])

PADDLE = ModelFormat(value=7, extensions=['pdmodel'])

PB = ModelFormat(value=6, extensions=['pb'])

RELAY = ModelFormat(value=5, extensions=['relay'])

TEXT = ModelFormat(value=8, extensions=['txt'])

TFLITE = ModelFormat(value=1, extensions=['tflite'])

property extension

property extensions

classmethod from_extension(*ext*)

mlonmcu.models.model.**parse_metadata_from_path**(*path*)

mlonmcu.models.model.**parse_shape_string**(*inputs_string*)

Parse an input shape dictionary string to a usable dictionary.

Taken from: https://github.com/apache/tvm/blob/main/python/tvm/driver/tvmc/shape_parser.py

Parameters

inputs_string: str A string of the form "input_name:[dim1,dim2,...,dimn] input_name2:[dim1,dim2]" that indicates the desired shape for specific model inputs. Colons, forward slashes and dots within input_names are supported. Spaces are supported inside of dimension arrays.

Returns

———-

shape_dict: dict A dictionary mapping input names to their shape for use in relay frontend converters.

mlonmcu.models.model.**parse_type_string**(*inputs_string*)

mlonmcu.models.options module

class mlonmcu.models.options.**BackendModelOptions**(*backend*, *supported=True*, *options={}*)

Bases: object

class mlonmcu.models.options.**TFLMIModelOptions**(*backend*, *supported=True*, *arena_size=None*, *builtin_ops=None*, *custom_ops=None*)

Bases: *mlonmcu.models.options.BackendModelOptions*

class mlonmcu.models.options.**TVMRTModelOptions**(*backend*, *supported=True*, *arena_size=None*)

Bases: *mlonmcu.models.options.BackendModelOptions*

mlonmcu.models.options.**parse_model_options_for_backend**(*backend*, *options*)

mlonmcu.models.utils module

mlonmcu.models.utils.**fill_data_source**(*in_bufs*, *out_bufs*)

mlonmcu.models.utils.**get_data_source**(*input_paths*, *output_paths*)

mlonmcu.models.utils.**lookup_data_buffers**(*input_paths*, *output_paths*)

mlonmcu.models.utils.**make_hex_array**(*filename*, *mode='bin'*)

Module contents

class mlonmcu.models.**LayerGenFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.Frontend*

DEFAULTS = {'fmt': 'tflite', 'use_inout_data': False}

FEATURES = ['validate']

REQUIRED = ['layergen.exe']

property fmt

generate(*model*) → Tuple[dict, dict]

property layergen_exe

produce_artifacts(*model*)

class mlonmcu.models.**ONNXFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'use_inout_data': False}

FEATURES = ['validate']

REQUIRED = []

class mlonmcu.models.**PBFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'use_inout_data': False}

FEATURES = ['validate']

REQUIRED = []

class mlonmcu.models.**PackedFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.Frontend*

DEFAULTS = {'check': False, 'fake_pack': False, 'ignore_existing': True, 'use_inout_data': False, 'use_packed': True}

FEATURES = ['validate', 'packing', 'packed']

REQUIRED = ['packer.exe']

property check

property fake_pack

property ignore_existing

produce_artifacts(*model*)

property use_packed

class mlonmcu.models.**TfLiteFrontend**(*features=None*, *config=None*)

Bases: *mlonmcu.models.frontend.SimpleFrontend*

DEFAULTS = {'pack_script': None, 'split_layers': False, 'use_inout_data': False, 'visualize_enable': False, 'visualize_script': None}

FEATURES = ['validate', 'visualize', 'split_layers']

REQUIRED = []

generate(*model*) → Tuple[dict, dict]

property pack_script

produce_artifacts(*model*)

property split_layers

property visualize_enable

property visualize_script

mlonmcu.models.**print_summary**(*context*, *detailed=False*)

mlonmcu.platform package

Submodules

mlonmcu.platform.espidf module

MLonMCU ESP-IDF platform

class mlonmcu.platform.espidf.**EspIdfPlatform**(*features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.CompilePlatform*, *mlonmcu.platform.platform.TargetPlatform*

ESP-IDF Platform class.

DEFAULTS = {'baud': 115200, 'build_dir': None, 'debug': False, 'flash_only': False, 'num_threads': 2, 'port': None, 'print_outputs': False, 'project_dir': None, 'project_template': None, 'use_idf_monitor': True, 'wait_for_user': True}

FEATURES = ['debug', 'benchmark']

REQUIRED = ['espidf.install_dir', 'espidf.src_dir']

property baud

check()

close()

compile(*target*, *src=None*)

create_target(*name*)

property espidf_install_dir

property espidf_src_dir

flash(*elf*, *target*, *timeout=120*)

property flash_only

generate(*src*, *target*, *model=None*)

get_idf_cmake_args()

get_idf_serial_args(*monitor=False*)

get_supported_targets()

property idf_exe

init_directory(*path=None*, *context=None*)

invoke_idf_exe(**args*, ***kwargs*)

monitor(*target*, *timeout=60*)

property port

prepare(*target*, *src*)

property project_template

property use_idf_monitor

property wait_for_user

mlonmcu.platform.espidf_target module

mlonmcu.platform.lookup module

mlonmcu.platform.lookup.**get_platform_names**(*context*)

mlonmcu.platform.lookup.**get_platforms_backends**(*context*, *config=None*)

mlonmcu.platform.lookup.**get_platforms_targets**(*context*, *config=None*)

mlonmcu.platform.lookup.**print_backends**(*platform_backends*)

mlonmcu.platform.lookup.**print_platforms**(*platform_names*)

mlonmcu.platform.lookup.**print_summary**(*context*)

mlonmcu.platform.lookup.**print_targets**(*platform_targets*)

mlonmcu.platform.microtvm module

MLonMCU MicroTVM platform

class mlonmcu.platform.microtvm.**MicroTvmPlatform**(*features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.CompilePlatform*, *mlonmcu.platform.platform.TargetPlatform*, *mlonmcu.platform.platform.BuildPlatform*, *mlonmcu.platform.platform.TunePlatform*

TVM Platform class.

DEFAULTS = {'autotuning_append': None, 'autotuning_early_stopping': None, 'autotuning_enable': False, 'autotuning_max_parallel': 1, 'autotuning_mode': 'autotvm', 'autotuning_num_workers': 1, 'autotuning_print_outputs': False, 'autotuning_results_file': None, 'autotuning_tasks': None, 'autotuning_timeout': 100, 'autotuning_trials': 10, 'autotuning_tuner': 'ga', 'autotuning_use_rpc': False, 'autotuning_visualize': False, 'build_dir': None, 'debug': False, 'experimental_autotvm_visualize': False, 'experimental_tvmc_micro_tune': False, 'experimental_tvmc_print_time': False, 'experimental_tvmc_tune_tasks': False, 'fill_mode': 'random', 'ins_file': None, 'num_threads': 2, 'outs_file': None, 'print_outputs': False, 'print_top': False, 'profile': False, 'project_dir': None, 'project_options': {}, 'project_template': None, 'repeat': 1, 'rpc_hostname': None, 'rpc_key': None, 'rpc_port': None, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_rpc': False}

FEATURES = ['debug', 'autotune', 'tvm_rpc', 'tvm_profile']

REQUIRED = ['tvm.build_dir', 'tvm.pythonpath', 'tvm.configs_dir']

close()

collect_available_project_options(*command*, *path*, *mlf_path*, *template*, *micro=True*, *target=None*)

collect_available_run_project_options(*path*, *device*)

compile(*target*)

create_backend(*name*)

create_target(*name*)

property experimental_tvmc_micro_tune

property experimental_tvmc_print_time

property experimental_tvmc_tune_tasks

property experimental_tvmc_tune_visualize

property fill_mode

flash(*elf*, *target*, *timeout=120*)

generate(*src*, *target*, *model=None*) → Tuple[dict, dict]

get_micro_tune_args(*model*, *backend*, *out*)

get_supported_backends()

get_supported_targets()

get_template_args(*target*)

get_tvmc_micro_args(*command*, *path*, *mlf_path*, *template*, *list_options=False*, *tune_args=None*)

get_tvmc_run_args(*path*, *device*, *list_options=False*)

init_directory(*path=None*, *context=None*)

property ins_file

invoke_tvmc(*command*, **args*, *target=None*, *prefix=''*)

invoke_tvmc_micro(*command*, *path*, *mlf_path*, *template*, *target*, *extra_args=None*, *micro=True*, *tune_args=None*, *prefix=''*)

invoke_tvmc_run(*path*, *device*, *template*, *target*, *micro=True*)

property outs_file

prepare(*mlf*, *target*)

property print_top

property profile

property project_options

property project_template

property repeat

property rpc_hostname

property rpc_key

property rpc_port

run(*elf*, *target*, *timeout=120*)

property tune_tasks

property tvm_build_dir

property tvm_configs_dir

property tvm_pythonpath

property tvmc_custom_script

property tvmc_extra_args

property use_rpc

property visualize_tuning

mlonmcu.platform.microtvm_backend module

mlonmcu.platform.microtvm_target module

mlonmcu.platform.mlif module

MLonMCU MLIF platform

class mlonmcu.platform.mlif.**MlifPlatform**(*features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.CompilePlatform*, *mlonmcu.platform.platform.TargetPlatform*

Model Library Interface Platform class.

DEFAULTS = {'build_dir': None, 'debug': False, 'debug_symbols': False, 'fail_on_error': False, 'ignore_data': True, 'input_data_path': None, 'mem_only': False, 'model_support_dir': None, 'num_threads': 2, 'optimize': None, 'output_data_path': None, 'prebuild_lib_path': None, 'print_outputs': False, 'toolchain': 'gcc', 'verbose_makefile': False}

FEATURES = ['debug', 'validate', 'muriscvnn', 'cmsisnn', 'muriscvnnbyoc', 'cmsisnnbyoc', 'vext', 'pext', 'arm_mvei', 'arm_dsp', 'auto_vectorize', 'benchmark', 'xpulp']

OPTIONAL = ['llvm.install_dir']

REQUIRED = ['mlif.src_dir']

close()

compile(*target*, *src=None*, *model=None*, *data_file=None*)

configure(*target*, *src*, *_model*)

create_target(*name*)

property debug_symbols

property fail_on_error

gen_data_artifact()

generate(*src*, *target*, *model=None*) → Tuple[dict, dict]

get_common_cmake_args()

get_supported_targets()

property ignore_data

init_directory(*path=None*, *context=None*)

property input_data_path

property llvm_dir

property mem_only

property mlif_dir

property model_support_dir

property optimize

property output_data_path

property prebuild_lib_dir

prepare()

property toolchain

property validate_outputs

property verbose_makefile

mlonmcu.platform.mlif_target module

mlonmcu.platform.platform module

class mlonmcu.platform.platform.**BuildPlatform**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.Platform*

Abstract backend platform class.

DEFAULTS = {'print_outputs': False}

FEATURES = []

REQUIRED = []

export_artifacts(*path*)

property supports_build

class mlonmcu.platform.platform.**CompilePlatform**(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.Platform*

Abstract compile platform class.

DEFAULTS = {'build_dir': None, 'debug': False, 'num_threads': 2, 'print_outputs': False}

FEATURES = ['debug']

REQUIRED = []

property debug

abstract generate(*src*, *target*, *model=None*) → Tuple[dict, dict]

generate_artifacts(*src*, *target*, *model=None*) → List[*mlonmcu.artifact.Artifact*]

get_metrics(*elf*)

property num_threads

property supports_compile

class mlonmcu.platform.platform.Platform(*name*, *features=None*, *config=None*)

Bases: object

Abstract platform class.

DEFAULTS = {'print_outputs': False}

FEATURES = []

OPTIONAL = []

REQUIRED = []

get_supported_backends()

get_supported_targets()

init_directory(*path=None*, *context=None*)

property print_outputs

process_features(*features*)

property supports_build

property supports_compile

property supports_flash

property supports_monitor

property supports_tune

class mlonmcu.platform.platform.TargetPlatform(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.Platform*

Abstract target platform class.

DEFAULTS = {'print_outputs': False}

FEATURES = []

REQUIRED = []

create_target(*name*)

flash(*elf*, *target*, *timeout=120*)

monitor(*target*, *timeout=60*)

run(*elf*, *target*, *timeout=120*)

property supports_flash

property supports_monitor

class mlonmcu.platform.platform.TunePlatform(*name*, *features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.Platform*

Abstract backend platform class.

DEFAULTS = {'print_outputs': False}

FEATURES = []

REQUIRED = []

export_artifacts(*path*)

property supports_tune

tune_model(*model_path*, *backend*, *target*)

mlonmcu.platform.tvm module

MLonMCU TVM platform

class mlonmcu.platform.tvm.**TvmPlatform**(*features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.BuildPlatform*, *mlonmcu.platform.platform.TargetPlatform*, *mlonmcu.platform.platform.TunePlatform*

TVM Platform class.

DEFAULTS = {'aggregate': 'none', 'autotuning_append': None, 'autotuning_early_stopping': None, 'autotuning_enable': False, 'autotuning_max_parallel': 1, 'autotuning_mode': 'autotvm', 'autotuning_num_workers': 1, 'autotuning_print_outputs': False, 'autotuning_results_file': None, 'autotuning_tasks': None, 'autotuning_timeout': 100, 'autotuning_trials': 10, 'autotuning_tuner': 'ga', 'autotuning_use_rpc': False, 'autotuning_visualize': False, 'experimental_tvmc_tune_tasks': False, 'experimental_tvmc_tune_visualize': False, 'fill_mode': 'random', 'ins_file': None, 'number': 1, 'outs_file': None, 'print_outputs': False, 'print_top': False, 'profile': False, 'project_dir': None, 'project_template': None, 'repeat': 1, 'rpc_hostname': None, 'rpc_key': None, 'rpc_port': None, 'total_time': False, 'tvmc_custom_script': None, 'tvmc_extra_args': [], 'use_rpc': False}

FEATURES = ['benchmark', 'tvm_rpc', 'autotune', 'tvm_profile']

REQUIRED = ['tvm.build_dir', 'tvm.pythonpath', 'tvm.configs_dir']

property aggregate

close()

create_backend(*name*)

create_target(*name*)

property experimental_tvmc_tune_tasks

property experimental_tvmc_tune_visualize

property fill_mode

get_supported_backends()

get_supported_targets()

get_tune_args(*model*, *backend*, *out*)

get_tvmc_run_args(*path*, *device*)

init_directory(*path=None*, *context=None*)

property ins_file

invoke_tvmc(*command*, **args*)

invoke_tvmc_run(*path*, *device*)

property number

property outs_file

property print_top

property profile

property repeat

property rpc_hostname

property rpc_key

property rpc_port

run(*elf*, *target*, *timeout=120*)

property total_time

property tvm_build_dir

property tvm_configs_dir

property tvm_pythonpath

property tvmc_custom_script

property tvmc_extra_args

property use_rpc

mlonmcu.platform.tvm_backend module

mlonmcu.platform.tvm_target module

mlonmcu.platform.zephyr module

MLonMCU Zephyr platform

class mlonmcu.platform.zephyr.**ZephyrPlatform**(*features=None*, *config=None*)

Bases: *mlonmcu.platform.platform.CompilePlatform*, *mlonmcu.platform.platform.TargetPlatform*

Zephyr Platform class.

DEFAULTS = {'baud': 115200, 'build_dir': None, 'debug': False, 'flash_only': False, 'num_threads': 2, 'optimize': None, 'port': None, 'print_outputs': False, 'project_dir': None, 'project_template': None, 'wait_for_user': True}

FEATURES = ['debug', 'benchmark']

REQUIRED = ['zephyr.install_dir', 'zephyr.sdk_dir', 'zephyr.venv_dir']

property baud

property build_dir

close()

compile(*target*, *src=None*)

create_target(*name*)

flash(*elf*, *target*, *timeout=120*)

property flash_only

generate(*src*, *target*, *model=None*) → Tuple[dict, dict]

get_serial(*target*)

get_supported_targets()

get_west_cmake_args()

init_directory(*path=None*, *context=None*)

invoke_west(**args*, ***kwargs*)

monitor(*target*, *timeout=60*)

property optimize

property port

prepare(*target*, *src*)

property project_template

property wait_for_user

property zephyr_install_dir

property zephyr_sdk_dir

property zephyr_venv_dir

mlonmcu.platform.zephyr_target module

Module contents

MLonMCU platform submodule

class mlonmcu.platform.**Platform**(*name*, *features=None*, *config=None*)

Bases: object

Abstract platform class.

DEFAULTS = {'print_outputs': False}

FEATURES = []

OPTIONAL = []

REQUIRED = []

get_supported_backends()

get_supported_targets()

init_directory(*path=None*, *context=None*)

property print_outputs

process_features(*features*)

property supports_build

property supports_compile

property supports_flash

property supports_monitor

property supports_tune

mlonmcu.platform.**get_platforms**()

mlonmcu.platform.**register_platform**(*platform_name*, *p*, *override=False*)

mlonmcu.session package

Subpackages

mlonmcu.session.postprocess package

Submodules

mlonmcu.session.postprocess.postprocess module

Definitions of base classes for MLonMCU postprocesses.

class mlonmcu.session.postprocess.postprocess.**Postprocess**(*name*, *config=None*, *features=None*)

Bases: object

Abstract postprocess.

DEFAULTS = {}

FEATURES = []

OPTIONAL = []

REQUIRED = []

process_features(*features*)

Utility which handles postprocess_features.

class mlonmcu.session.postprocess.postprocess.**RunPostprocess**(*name*, *config=None*, *features=None*)

Bases: *mlonmcu.session.postprocess.postprocess.Postprocess*

Run postprocess which is applied to a single run.

post_run(*report*, *artifacts*)

Called at the end of a run.

class mlonmcu.session.postprocess.postprocess.**SessionPostprocess**(*name*, *config=None*, *features=None*)

Bases: *mlonmcu.session.postprocess.postprocess.Postprocess*

Session postprocess which is applied to multiple runs at the end of a session. (multi-row)

post_session(*report*)

Called at the end of a session.

mlonmcu.session.postprocess.postprocesses module

Collection of (example) postprocesses integrated in MLonMCU.

class mlonmcu.session.postprocess.postprocesses.**AnalyseInstructionsPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.RunPostprocess*

Counting specific types of instructions.

DEFAULTS = {'groups': True, 'sequences': True, 'top': 10}

property groups

Get groups property.

post_run(*report*, *artifacts*)

Called at the end of a run.

property sequences

get sequences property.

property top

get sequences property.

class mlonmcu.session.postprocess.postprocesses.**Artifact2ColumnPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.RunPostprocess*

Postprocess for converting artifacts to columns in the report.

DEFAULTS = {'file2colname': {}}

property file2colname

Get file2colname property.

post_run(*report*, *artifacts*)

Called at the end of a run.

class mlonmcu.session.postprocess.postprocesses.**Bytes2kBPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

Postprocess which can be used to scale the memory related columns from Bytes to KiloBytes.

post_session(*report*)

Called at the end of a session.

class mlonmcu.session.postprocess.postprocesses.**CompareRowsPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

TODO

DEFAULTS = {'baseline': 0, 'group_by': None, 'invert': False, 'percent': False, 'substract': False, 'to_compare': None}

property baseline

Get baseline property.

property group_by

Get group_by property.

property invert

Get invert property.

property percent

Get percent property.

post_session(*report*)

Called at the end of a session.

property substract

Get substract property.

property to_compare

Get to_compare property.

class mlonmcu.session.postprocess.postprocesses.**Config2ColumnsPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

Postprocess which can be used to transform (explode) the 'Config' Column in a dataframe for easier filtering.

DEFAULTS = {'limit': None}

property limit

post_session(*report*)

Called at the end of a session.

class mlonmcu.session.postprocess.postprocesses.**Features2ColumnsPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

Postprocess which can be used to transform (explode) the 'Features' Column in a dataframe for easier filtering.

post_session(*report*)

Called at the end of a session.

class mlonmcu.session.postprocess.postprocesses.**FilterColumnsPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

Postprocess which can be used to drop unwanted columns from a report.

DEFAULTS = {'drop': None, 'drop_const': False, 'drop_empty': False, 'drop_nan': False, 'keep': None}

property drop

Get drop property.

property drop_const

Get drop_const property.

property drop_empty

Get drop_empty property.

property drop_nan

Get drop_nan property.

property keep

Get keep property.

post_session(*report*)

Called at the end of a session.

class mlonmcu.session.postprocess.postprocesses.**PassConfig2ColumnsPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

Postprocess which can be used to transform (explode) the TVM pass_config into separate columns. requires prior Config2Columns pass.

post_session(*report*)

Called at the end of a session.

class mlonmcu.session.postprocess.postprocesses.**RenameColumnsPostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

Postprocess which can rename columns based on a provided mapping.

DEFAULTS = {'mapping': {}}

property mapping

post_session(*report*)

Called at the end of a session.

class mlonmcu.session.postprocess.postprocesses.**VisualizePostprocess**(*features=None*, *config=None*)

Bases: *mlonmcu.session.postprocess.postprocess.SessionPostprocess*

A very simple example on how to generate a plot of the results using a postprocess.

DEFAULTS = {'format': 'png'}

property format

Get format property.

post_session(*report*)

Called at the end of a session.

mlonmcu.session.postprocess.postprocesses.**match_rows**(*df*, *cols*)

Helper function to group similar rows in a dataframe.

Module contents

MLonMCU postprocess submodule

Submodules

mlonmcu.session.run module

Definition of a MLonMCU Run which represents a single benchmark instance for a given set of options.

class mlonmcu.session.run.**Run**(*idx=None*, *model=None*, *framework=None*, *frontends=None*, *backend=None*, *target=None*, *platforms=None*, *features=None*, *config=None*, *postprocesses=None*, *archived=False*, *session=None*, *comment=''*)

Bases: `object`

A run is single model/backend/framework/target combination with a given set of features and configs.

DEFAULTS = {'export_optional': False, 'stage_subdirs': False, 'target_to_backend': False, 'tune_enabled': False}

FEATURES = ['autotune', 'target_optimized']

OPTIONAL = []

REQUIRED = []

add_backend(*backend*)

Setter for the backend instance.

add_backend_by_name(*backend_name*, *context=None*)

Helper function to initialize and configure a backend by its name.

add_feature(*feature*)
Setter for a feature instance.

add_feature_by_name(*feature_name*, *append=True*, *context=None*)
Helper function to initialize and configure a feature by its name.

add_features(*features*, *append=False*)
Setter for the list of features.

add_features_by_name(*feature_names*, *append=False*, *context=None*)
Helper function to initialize and configure features by their names.

add_framework(*framework*)
Setter for the framework instance.

add_frontend(*frontend*)
Setter for the frontend instance.

add_frontend_by_name(*frontend_name*, *context=None*)
Helper function to initialize and configure a frontend by its name.

add_frontends(*frontends*)
Setter for the list of frontends.

add_frontends_by_name(*frontend_names*, *context=None*)
Helper function to initialize and configure frontends by their names.

add_model(*model*)
Setter for the model instance.

add_model_by_name(*model_name*, *context=None*)
Helper function to initialize and configure a model by its name.

add_platform(*platform*)
Setter for the platform instance.

add_platform_by_name(*platform_name*, *context=None*)
Helper function to initialize and configure a platform by its name.

add_platforms(*platforms*)
Setter for the list of platforms.

add_platforms_by_name(*platform_names*, *context=None*)
Helper function to initialize and configure platforms by their names.

add_postprocess(*postprocess*, *append=False*)
Setter for a postprocess instance.

add_postprocess_by_name(*postprocess_name*, *append=True*, *context=None*)
Helper function to initialize and configure a postprocesses by its name.

add_postprocesses(*postprocesses*, *append=False*)
Setter for the list of postprocesses.

add_postprocesses_by_name(*postprocess_names*, *append=False*, *context=None*)
Helper function to initialize and configure postprocesses by their names.

add_target(*target*)

Setter for the target instance.

add_target_by_name(*target_name*, *context=None*)

Helper function to initialize and configure a target by its name.

property artifacts

build()

Process the run using the choosen backend.

property build_platform

Get platform for build stage.

compile()

Compile the target software for the run.

property compile_platform

Get platform for compile stage.

copy()

Create a new run based on this instance.

export(*path=None*, *optional=False*)

Write a run configuration to a disk.

property export_optional

Get export_optional property.

export_stage(*stage*, *optional=False*)

Export stage artifacts of this run to its directory.

classmethod from_file(*path*)

Restore a run object which was written to the disk.

property frontend

get_all_configs(*omit_paths=False*, *omit_defaults=False*, *omit_globals=False*)

Return dict with component-specific and global configuration for this run.

get_all_feature_names(*only_used=True*)

Return list of feature names for this run.

get_all_postprocess_names()

Return list of postprocess names for this run.

get_all_sub_artifacts(*sub*, *stage=None*)

get_frontend_name()

Return frontend name(s) for this run.

get_platform_name()

Return platform name(s) for this run.

get_report()

Returns teh complete report of this run.

has_stage(*stage*)

Returns true if the given stage is available for this run.

init_component(*component_cls*, *context=None*)

Helper function to create and configure a MLonMCU component instance for this run.

init_directory()

Initialize the temporary directory for this run.

property last_stage

Determines the next not yet completed stage. Returns RunStage.DONE if already completed.

load()

Load the model using the given frontend.

lock()

Aquire a mutex to lock the current run.

property next_stage

Determines the next not yet completed stage. Returns RunStage.DONE if already completed.

pick_model_frontend(*model_hints*, *backend=None*)

postprocess()

Postprocess the 'run'.

property prefix

Get prefix property.

process(*until=RunStage.RUN*, *skip=None*, *export=False*)

Process the run until a given stage.

process_features(*features*)

Utility which handles postprocess_features.

run()

Run the 'run' using the defined target.

property stage_subdirs

property target_platform

Get platform for run stage.

property target_to_backend

Get target_to_backend property.

toDict()

Utility not implemented yet. (TODO: remove?)

tune()

Tune the run using the choosen backend (if supported).

property tune_enabled

Get tune_enabled property.

property tune_platform

Get platform for tune stage.

unlock()

Release a mutex to unlock the current run.

write_run_file()

Create a run.txt file which contains information used to reconstruct the run based on its properties at a later point in time.

class mlonmcu.session.run.**RunStage**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: enum.IntEnum

Type describing the stages a run can have.

BUILD = 3

COMPILE = 4

DONE = 7

LOAD = 1

NOP = 0

POSTPROCESS = 6

RUN = 5

TUNE = 2

mlonmcu.session.session module

Definition of a MLonMCU Run which represents a set of benchmarks in a session.

class mlonmcu.session.session.**Session**(*label=''*, *idx=None*, *archived=False*, *dir=None*, *config=None*)

Bases: object

A session which wraps around multiple runs in a context.

DEFAULTS = {'report_fmt': 'csv'}

property active

Get active property.

close(*err=None*)

Close this run.

create_run(**args*, ***kwargs*)

Factory method to create a run and add it to this session.

discard()

Discard a run and remove its directory.

enumerate_runs()

Update run indices.

get_reports()

Returns a full report which includes all runs in this session.

open()

Open this run.

property prefix

get prefix property.

process_runs(*until=RunStage.DONE*, *per_stage=False*, *print_report=False*, *num_workers=1*, *progress=False*, *export=False*, *context=None*)

Process a runs in this session until a given stage.

property report_fmt

get report_fmt property.

request_run_idx()

Return next free run index.

class mlonmcu.session.session.**SessionStatus**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: enum.Enum

Status type for a session.

CLOSED = 2

CREATED = 0

ERROR = 3

OPEN = 1

Module contents

mlonmcu.setup package

Submodules

mlonmcu.setup.cache module

Definition of Taks Cache

class mlonmcu.setup.cache.**TaskCache**

Bases: object

Task cache used to store dependency paths for the current and furture sessions.

This can be interpreted as a "modded" dictionary which takes a key + some flags.

find_best_match(*name: str*, *flags=[]*) → Any

Utility whih tries to resolve the cache entry with the beste match.

Parameters

name [str] The cache-key.

flags [list] Optional flags used for the lookup.

read_from_file(*filename*, *reset=True*)

write_to_file(*filename*)

mlonmcu.setup.cache.**convert_key**(*name*)

mlonmcu.setup.gen_requirements module

MLonMCU Python requirements.txt generator.

This script generates a set of requirements.txt files (stored in *./requirements*) that describe MLonMCU's Python dependencies.

Pieces

MLonMCU can be roughly broken into these named pieces along the lines of Python dependencies:

- "core": A core piece, which is intended to be buildable with very few external dependencies. Users can use Relay, compile models, and run autotuning with this part.
- Extra features (i.e. TVM). These enhance MLonMCU's functionality, but aren't required for basic operation.

What this tool does

From these pieces, this tool builds:

- requirements/<name>.txt - Python dependencies for each named piece above, <*name*> is the same as the quoted piece name.
- requirements/all.txt - Consolidated Python dependencies for all pieces, excluding dev below.
- requirements/dev.txt - Python dependencies needed to develop MLONMCU, such as lint and test tools.

The data representing each piece is contained in the two maps below.

exception mlonmcu.setup.gen_requirements.**ValidationError**(*config: str*, *problems: List[str]*)

Bases: Exception

Raised when a validation error occurs.

static format_problems(*config: str*, *problems: List[str]*) → str

Format a list of problems with a global config variable into human-readable output.

Parameters

config [str] Name of the global configuration variable of concern. Prepended to the output.

problems: list[str] A list of strings, each one a distinct problem with that config variable.

Returns

str A human-readable string suitable for console, listing the problems as bullet points.

mlonmcu.setup.gen_requirements.**join_and_write_requirements**(*args: argparse.Namespace*)

mlonmcu.setup.gen_requirements.**join_requirements**() → Dict[str, Tuple[str, List[str]]]

Validate, then join REQUIRMENTS_BY_PIECE against CONSTRAINTS and return the result.

Returns

An OrderedDict containing REQUIREMENTS_BY_PIECE, except any dependency mentioned in CONSTRAIN

is replaced by a setuptools-compatible constraint.

mlonmcu.setup.gen_requirements.**main**()

mlonmcu.setup.gen_requirements.**parse_args**() → argparse.Namespace

mlonmcu.setup.gen_requirements.**parse_semver**(*package: str*, *constraint: str*, *problems: List[str]*) → Tuple[List[str], int, int]

Parse a semantic versioning constraint of the form "^X.[.Y[.Z[...]]]"

Parameters

package [str] Name of the package specifying this constraint, for reporting problems.

constraint [str] The semver constraint. Must start with "^"

problems [List[str]] A list of strings describing problems that have occurred validating the configuration. Problems encountered while validating constraint are appended to this list.

Returns

tuple[list[str], int, int] A 3-tuple. The first element is a list containing an entry for each component in the semver string (components separated by "."). The second element is the index of the component in the list which must not change to meet the semver constraint. The third element is an integer, the numeric value of the changing component (this can be non-trivial when the patch is the changing part but pre-, post-release, or build metadta.

See "Caret requirements" at https://python-poetry.org/docs/versions/.

mlonmcu.setup.gen_requirements.**semver_to_requirements**(*dep: str*, *constraint: str*, *joined_deps: List[str]*)

Convert a SemVer-style constraint to a setuptools-compatible constraint.

Parameters

dep [str] Name of the PyPI package to depend on.

constraint [str] The SemVer constraint, of the form "^<semver constraint>"

joined_deps [list[str]] A list of strings, each a setuptools-compatible constraint which could be written to a line in requirements.txt. The converted constraint is appended to this list.

mlonmcu.setup.gen_requirements.**validate_constraints**() → List[str]

Validate CONSTRAINTS, returning a list of problems found.

Returns

list[str] A list of strings, each one describing a distinct problem found in CONSTRAINTS.

mlonmcu.setup.gen_requirements.**validate_or_raise**()

mlonmcu.setup.gen_requirements.**validate_requirements_by_piece**() → List[str]

Validate REQUIREMENTS_BY_PIECE, returning a list of problems.

Returns

list[str] A list of strings, each one describing a distinct problem with REQUIREMENTS_BY_PIECE.

mlonmcu.setup.setup module

class mlonmcu.setup.setup.**Setup**(*features=None*, *config=None*, *context=None*, *tasks_factory=None*)

Bases: object

MLonMCU dependency management interface.

DEFAULTS = {'num_threads': None, 'print_outputs': False}

FEATURES = []

OPTIONAL = []

REQUIRED = []

clean_cache(*interactive=True*)

clean_dependencies(*interactive=True*)

generate_requirements()

get_dependency_order()

install_dependencies(*progress=False*, *write_cache=True*, *write_env=True*, *rebuild=False*)

invoke_single_task(*name*, *progress=False*, *write_cache=True*, *write_env=True*, *rebuild=False*)

process_features(*features*)

setup_progress_bar(*enabled*)

property verbose

visualize(*path*, *ordered=False*)

write_cache_file()

write_env_file()

mlonmcu.setup.task module

Definitions of a task registry used to automatically install dependencies.

class mlonmcu.setup.task.**TaskFactory**

Bases: object

Class which is used to register all available tasks and their annotations.

Attributes

registry [dict] Mapping of task names and their actual function

dependencies [dict] Mapping of task dependencies

providers [dict] Mapping of which task provides which artifacts

types [dict] Mapping of task types

validates [dict] Mapping of validation functions for the tasks

changed [list] List of tasks?artifacts which have changed recently

needs(*keys*, *force=True*)

Decorator which registers the artifacts a task needs to be processed.

optional(*keys*)

Decorator for optional task requirements.

param(*flag*, *options*)

Decorator which registers available task parameters.

provides(*keys*)

Decorator which registers what a task provides.

register(*category=TaskType.MISC*)

Decorator which actually registers a task in the registry.

reset_changes()

Reset all pending changes.

validate(*func*)

Decorator which registers validation functions for a task.

class mlonmcu.setup.task.**TaskGraph**(*names: List[str]*, *dependencies: dict*, *providers: dict*)

Bases: object

Task graph object.

Attributes

names [list] list of task names in the graph

dependencies [dict] Dependencies between task artifacts

providers [dict] Providers for all the artifacts

Examples

——-

TODO

export_dot(*path*)

Visualize the task dependency graph.

get_graph() → Tuple[list, list]

Get nodes and edges of the task graph.

Returns

nodes [list] List of edges

edges [list] List of edge tuples.

get_order() → list

Get execution order of tasks via topological sorting.

class mlonmcu.setup.task.**TaskType**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: enum.Enum

Enumeration for the task type.

BACKEND = 2

FEATURE = 7

FRAMEWORK = 1

FRONTEND = 5

MISC = 0

OPT = 6

PLATFORM = 8

TARGET = 4

TOOLCHAIN = 3

mlonmcu.setup.task.**get_combs**(*data*) → List[dict]

Utility which returns combinations of the input data.

Parameters

data [dict] Input dictionary

Returns

combs [list] All combinations of the input data.

Examples

```
>>> get_combs({"foo": [False, True], "bar": [5, 10]})
[{"foo": False, "bar": 5}, {"foo": False, "bar": 10}, {"foo": True, "bar": 5}, {"foo
↪": True, "bar": 10}]
```

mlonmcu.setup.tasks module

mlonmcu.setup.tasks.**get_task_factory**()

mlonmcu.setup.utils module

mlonmcu.setup.utils.**apply**(*repo_dir: pathlib.Path*, *patch_file: pathlib.Path*)

Helper function for applying a patch to a repository.

Parameters

repo_dir [Path] Clone directory of repository.

patch_file [Path] Path to patch file.

mlonmcu.setup.utils.**clone**(*url: str*, *dest: Union[str, bytes, os.PathLike]*, *branch: str = ''*, *submodules: list = []*, *recursive: bool = False*, *refresh: bool = False*)

Helper function for cloning a repository.

Parameters

url [str] Clone URL of the repository.

dest [Path] Destination directory path.

branch [str] Optional branch name or commit reference/tag.

submodules [list of strings] Only affects when recursive is true. Submodules to be updated. If empty, all submodules will be updated.

recursive [bool] If the clone should be done recursively.

refesh [bool] Enables switching the url/branch if the repo already exists

mlonmcu.setup.utils.**cmake**(*src*, **args*, *debug=False*, *use_ninja=False*, *cwd=None*, ***kwargs*)

mlonmcu.setup.utils.**copy**(*src*, *dest*)

mlonmcu.setup.utils.**download**(*url*, *dest*, *progress=False*)

mlonmcu.setup.utils.**download_and_extract**(*url*, *archive*, *dest*, *progress=False*)

mlonmcu.setup.utils.**exec**(**args*, ***kwargs*)

Execute a process with the given args and using the given kwards as Popen arguments.

Parameters

args The command to be executed.

mlonmcu.setup.utils.**exec_getout**(**args*, *live: bool = False*, *print_output: bool = True*, *handle_exit=None*, *prefix=''*, ***kwargs*) → str

Execute a process with the given args and using the given kwards as Popen arguments and return the output.

Parameters

args The command to be executed.

live [bool] If the stdout should be updated in real time.

print_output [bool] Print the output at the end on non-live mode.

Returns

output The text printed to the command line.

mlonmcu.setup.utils.**extract**(*archive*, *dest*, *progress=False*)

mlonmcu.setup.utils.**is_populated**(*path*)

mlonmcu.setup.utils.**make**(**args*, *threads=2*, *use_ninja=False*, *cwd=None*, *verbose=False*, ***kwargs*)

mlonmcu.setup.utils.**makeDirName**(*base: str*, **args*, *flags: list = None*) → str

Creates a directory name based on configuration values.

Using snake_case style.

Parameters

base [str] Prefix of the filename to be generated.

args List of tuples of the form: [(True, "foo"), (False, "bar")]

flags [list] Optional list of additional flags to be added.

Returns

dirname [str] The generated directory name

Examples

```
>>> makeDirName("base", (True, "foo"), (False, "bar"), flags=["flag"])
"base_foo_flag"
```

mlonmcu.setup.utils.**makeFlags**(**args*)

Resolve tuple-like arguments to a list of string.

Parameters

args List of tuples of the form: [(True, "foo"), (False, "bar")]

Returns

dirname [str] The generated directory name

Examples

```
>>> makeFlags((True, "foo"), (False, "bar"))
["foo"]
```

mlonmcu.setup.utils.**mkdirs**(*path: Union[str, bytes, os.PathLike]*)

Wrapper for os.makedirs which handels the special case where the path already exits.

mlonmcu.setup.utils.**move**(*src*, *dest*)

mlonmcu.setup.utils.**patch**(*path*, *cwd=None*)

mlonmcu.setup.utils.**python**(**args*, ***kwargs*)

Run a python script with the current interpreter.

mlonmcu.setup.utils.**remove**(*path*)

Module contents

mlonmcu.target package

Subpackages

mlonmcu.target.arm package

Submodules

mlonmcu.target.arm.corstone300 module

MLonMCU Corstone300 Target definitions

class mlonmcu.target.arm.corstone300.**Corstone300Target**(*name='corstone300'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.target.Target*

Target using an ARM FVP (fixed virtual platform) based on a Cortex M55 with EthosU support

DEFAULTS = {'enable_dsp': False, 'enable_ethosu': False, 'enable_fpu': True, 'enable_mvei': False, 'ethosu_num_macs': 256, 'extra_args': '', 'model': 'cortex-m55', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0}

FEATURES = ['ethosu', 'arm_mvei', 'arm_dsp']

REQUIRED = ['corstone300.exe', 'cmsisnn.dir', 'arm_gcc.install_dir']

property cmsisnn_dir

property enable_dsp

property enable_ethosu

property enable_fpu

property enable_mvei

property ethosu_num_macs

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extra_args

property fvp_exe

property gcc_prefix

get_arch()

get_backend_config(*backend*)

get_default_fvp_args()

get_ethosu_fvp_args()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

property model

parse_stdout(*out*, *handle_exit=None*)

property timeout_sec

mlonmcu.target.arm.util module

MLonMCU ARM Cortex-M utilities

mlonmcu.target.arm.util.**resolve_cpu_features**(*model*, *enable_fp=None*, *enable_fp_dp=None*, *enable_dsp=None*, *enable_mve=None*)

Module contents

class mlonmcu.target.arm.**Corstone300Target**(*name='corstone300'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.target.Target*

Target using an ARM FVP (fixed virtual platform) based on a Cortex M55 with EthosU support

DEFAULTS = {'enable_dsp': False, 'enable_ethosu': False, 'enable_fpu': True, 'enable_mvei': False, 'ethosu_num_macs': 256, 'extra_args': '', 'model': 'cortex-m55', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0}

FEATURES = ['ethosu', 'arm_mvei', 'arm_dsp']

REQUIRED = ['corstone300.exe', 'cmsisnn.dir', 'arm_gcc.install_dir']

property cmsisnn_dir

property enable_dsp

property enable_ethosu

property enable_fpu

property enable_mvei

property ethosu_num_macs

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extra_args

property fvp_exe

property gcc_prefix

get_arch()

get_backend_config(*backend*)

get_default_fvp_args()

get_ethosu_fvp_args()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

property model

parse_stdout(*out*, *handle_exit=None*)

property timeout_sec

mlonmcu.target.riscv package

Submodules

mlonmcu.target.riscv.etiss_pulpino module

MLonMCU ETISS/Pulpino Target definitions

class mlonmcu.target.riscv.etiss_pulpino.**EtissPulpinoTarget**(*name='etiss_pulpino'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a Pulpino-like VP running in the ETISS simulator

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'cpu_arch': None, 'cycle_time_ps': 31250, 'debug_etiss': False, 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'jit': None, 'pext_spec': 0.96, 'plugins': [], 'print_outputs': False, 'ram_size': 67108864, 'ram_start': 8388608, 'repeat': None, 'rom_size': 8388608, 'rom_start': 0, 'timeout_sec': 0, 'trace_memory': False, 'verbose': False, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'gdbserver', 'etissdbg', 'trace', 'log_instrs', 'pext', 'vext']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'etiss.src_dir', 'etiss.install_dir', 'etissvp.script']

property attr

property cpu_arch

property cycle_time_ps

property debug_etiss

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

property etiss_dir

property etiss_script

property etiss_src_dir

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_target_system()

property jit

parse_stdout(*out*, *handle_exit=None*)

property pext_spec

property plugins

property ram_size

property ram_start

property rom_size

property rom_start

property trace_memory

property verbose

property vext_spec

property vlen

write_ini(*path*)

mlonmcu.target.riscv.ovpsim module

MLonMCU OVPSim Target definitions

class mlonmcu.target.riscv.ovpsim.**OVPSimTarget**(*name='ovpsim'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using an ARM FVP (fixed virtual platform) based on a Cortex M55 with EthosU support

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'bitmanip_spec': 0.94, 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': True, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'pext_spec': 0.96, 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'variant': None, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext', 'pext', 'gdbserver', 'log_instrs', 'trace']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'ovpsim.exe']

property attr

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_backend_config(*backend*)

get_default_ovpsim_args()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

property ovpsim_exe

parse_stdout(*out*)

property pext_spec

property variant

property vext_spec

property vlen

mlonmcu.target.riscv.ovpsim.**replace_unsupported**(*exts*)

mlonmcu.target.riscv.riscv module

MLonMCU RISC-V Target definitions

class mlonmcu.target.riscv.riscv.**RISCVTarget**(*name: str*, *features: List[*mlonmcu.feature.feature.Feature*] = None*, *config: dict = None*)

Bases: *mlonmcu.target.target.Target*

Common base class for RISCV-like targets. Please do not use this as a target itself!

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'xlen': 32}

FEATURES = ['benchmark']

OPTIONAL = ['llvm.install_dir']

PUPL_GCC_TOOLCHAIN_REQUIRED = ['pulp_gcc.install_dir', 'pulp_gcc.name']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant']

property abi

property arch

property attr

property extensions

property extra_args

property fpu

property gcc_variant

get_arch()

get_backend_config(*backend*)

get_platform_defs(*platform*)

get_target_system()

property has_fpu

property pulp_gcc_basename

property pulp_gcc_prefix

property riscv_gcc_basename

property riscv_gcc_prefix

property timeout_sec

property xlen

mlonmcu.target.riscv.riscv_qemu module

MLonMCU RISC-V QEMU Target definitions

class mlonmcu.target.riscv.riscv_qemu.**RiscvQemuTarget**(*name='riscv_qemu'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a spike machine in the QEMU simulator

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'elen': 32, 'embedded_vext': False, 'enable_vext': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'riscv32_qemu.exe']

property attr

property elen

property embedded_vext

property enable_vext

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

get_cpu_str()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_qemu_args(*program*)

get_target_system()

parse_stdout(*out*, *handle_exit=None*)

property riscv32_qemu_exe

property vext_spec

property vlen

mlonmcu.target.riscv.spike module

MLonMCU Spike Target definitions

class mlonmcu.target.riscv.spike.**SpikeTarget**(*name='spike'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using the riscv-isa-sim (Spike) RISC-V simulator.

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'pext_spec': 0.92, 'print_outputs': False, 'repeat': None, 'spikepk_extra_args': [], 'timeout_sec': 0, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext', 'pext', 'cachesim', 'log_instrs']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'spike.exe', 'spike.pk']

property attr

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

parse_stdout(*out*)

property pext_spec

property spike_exe

property spike_pk

property spikepk_extra_args

property vext_spec

property vlen

mlonmcu.target.riscv.spike.**filter_unsupported**(*arch*)

mlonmcu.target.riscv.util module

MLonMCU RISC-V utilities

mlonmcu.target.riscv.util.**join_extensions**(*exts*)

mlonmcu.target.riscv.util.**sort_extensions_canonical**(*extensions*, *lower=False*, *unpack=False*)

Utility to get the canonical architecture name string.

mlonmcu.target.riscv.util.**update_extensions**(*exts*, *pext=None*, *pext_spec=None*, *vext=None*, *elen=None*, *embedded=None*, *fpu=None*, *variant=None*)

mlonmcu.target.riscv.util.**update_extensions_pulp**(*exts*, *xpulp_version*)

Module contents

class mlonmcu.target.riscv.**AraTarget**(*name='ara'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a Pulpino-like VP running in the GVSOC simulator

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'elen': 64, 'embedded_vext': False, 'enable_vext': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'nr_lanes': 4, 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'vext_spec': 1.0, 'vlen': 4096, 'xlen': 64}

FEATURES = ['benchmark', 'log_instrs', 'vext']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'ara.src_dir', 'verilator.install_dir']

property ara_apps_dir

property ara_hardware_dir

property elen

property embedded_vext

property enable_vext

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute an executable with given arguments

property extensions

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_target_system()

property nr_lanes

parse_stdout(*out*)

prepare_simulator(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

property verilator_install_dir

property vext_spec

property vlen

class mlonmcu.target.riscv.EtissPulpinoTarget(*name='etiss_pulpino'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a Pulpino-like VP running in the ETISS simulator

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'cpu_arch': None, 'cycle_time_ps': 31250, 'debug_etiss': False, 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'jit': None, 'pext_spec': 0.96, 'plugins': [], 'print_outputs': False, 'ram_size': 67108864, 'ram_start': 8388608, 'repeat': None, 'rom_size': 8388608, 'rom_start': 0, 'timeout_sec': 0, 'trace_memory': False, 'verbose': False, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'gdbserver', 'etissdbg', 'trace', 'log_instrs', 'pext', 'vext']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'etiss.src_dir', 'etiss.install_dir', 'etissvp.script']

property attr

property cpu_arch

property cycle_time_ps

property debug_etiss

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

property etiss_dir

property etiss_script

property etiss_src_dir

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_target_system()

property jit

parse_stdout(*out*, *handle_exit=None*)

property pext_spec

property plugins

property ram_size

property ram_start

property rom_size

property rom_start

property trace_memory

property verbose

property vext_spec

property vlen

write_ini(*path*)

class mlonmcu.target.riscv.**GvsocPulpTarget**(*name='gvsoc_pulp'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a Pulpino-like VP running in the GVSOC simulator

DEFAULTS = {'abi': 'ilp32', 'arch': None, 'attr': '', 'extensions': ['i', 'm', 'c'], 'extra_args': '', 'fpu': None, 'model': 'pulp', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'xlen': 32, 'xpulp_version': None}

FEATURES = ['benchmark', 'log_instrs', 'xpulp']

REQUIRED = ['pulp_gcc.install_dir', 'pulp_gcc.name', 'gvsoc.exe', 'pulp_freertos.support_dir', 'pulp_freertos.config_dir', 'pulp_freertos.install_dir']

property abi

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute an executable with given arguments

property extensions

get_backend_config(*backend*)

get_basic_gvsoc_simulating_arg(*program*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_target_system()

property gvsoc_folder

gvsoc_preparation_env()

property gvsoc_script

property model

parse_stdout(*out*)

property pulp_freertos_config_dir

property pulp_freertos_install_dir

property pulp_freertos_support_dir

property xpulp_version

class mlonmcu.target.riscv.OVPSimTarget(*name='ovpsim'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using an ARM FVP (fixed virtual platform) based on a Cortex M55 with EthosU support

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'bitmanip_spec': 0.94, 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': True, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'pext_spec': 0.96, 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'variant': None, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext', 'pext', 'gdbserver', 'log_instrs', 'trace']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'ovpsim.exe']

property attr

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_backend_config(*backend*)

get_default_ovpsim_args()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

property ovpsim_exe

parse_stdout(*out*)

property pext_spec

property variant

property vext_spec

property vlen

class mlonmcu.target.riscv.RiscvQemuTarget(*name='riscv_qemu'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a spike machine in the QEMU simulator

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'elen': 32, 'embedded_vext': False, 'enable_vext': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'riscv32_qemu.exe']

property attr

property elen

property embedded_vext

property enable_vext

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

get_cpu_str()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_qemu_args(*program*)

get_target_system()

parse_stdout(*out*, *handle_exit=None*)

property riscv32_qemu_exe

property vext_spec

property vlen

class mlonmcu.target.riscv.**SpikeTarget**(*name='spike'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using the riscv-isa-sim (Spike) RISC-V simulator.

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'pext_spec': 0.92, 'print_outputs': False, 'repeat': None, 'spikepk_extra_args': [], 'timeout_sec': 0, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext', 'pext', 'cachesim', 'log_instrs']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'spike.exe', 'spike.pk']

property attr

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

parse_stdout(*out*)

property pext_spec

property spike_exe

property spike_pk

property spikepk_extra_args

property vext_spec

property vlen

Submodules

mlonmcu.target.common module

Helper functions used by MLonMCU targets

mlonmcu.target.common.**add_common_options**(*parser: argparse.ArgumentParser*, *target*)

Add a set of common options to a command line parser.

Parameters

parser [argparse.ArgumentParser] The command line parser

mlonmcu.target.common.**cli**(*target*, *args: List[str] = None*)

Utility to handle the command line api for targets.

Parameters

target [Target] The target to be used.

args [list] Interface to pass arguments to the command line parser from test functions.

mlonmcu.target.common.**execute**(**args: typing.List[str], ignore_output: bool = False, live: bool = False, print_func: typing.Callable = <built-in function print>, handle_exit=None, err_func: typing.Callable = <bound method Logger.error of <Logger mlonmcu (INFO)>>, **kwargs*) → str

Wrapper for running a program in a subprocess.

Parameters

args [list] The actual command.

ignore_output [bool] Do not get the stdout and stderr or the subprocess.

live [bool] Print the output line by line instead of only at the end.

print_func [Callable] Function which should be used to print sysout messages.

err_func [Callable] Function which should be used to print errors.

kwargs: dict Arbitrary keyword arguments passed through to the subprocess.

Returns

out [str] The command line output of the command

mlonmcu.target.common.**init_target_features**(*names*, *config*)

mlonmcu.target.elf module

ELF File Tool

mlonmcu.target.elf.**get_results**(*elfFile*)

Converts and returns collected data.

mlonmcu.target.elf.**logger = <Logger mlonmcu (INFO)>**

Script to gather metrics on static ROM and RAM usage.

Heavility inspired by get_metrics.py found in the ETISS repository

mlonmcu.target.elf.**main**()

Main entry point for command line usage.

mlonmcu.target.elf.**parseElf**(*inFile*)

Extract static memory usage details from ELF file by mapping each segment.

mlonmcu.target.elf.**parse_cmdline**()

Cmdline interface definition.

mlonmcu.target.elf.**printSz**(*sz*, *unknown_msg=''*)

Helper function for printing file sizes.

mlonmcu.target.elf.**print_results**(*results*)

Displaying a fancy overview.

mlonmcu.target.elf.**write_csv**(*filename*, *results*)

Utility for writing a CSV file.

mlonmcu.target.host_x86 module

MLonMCU Host/x86 Target definitions

class mlonmcu.target.host_x86.**HostX86Target**(*name='host_x86'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.target.Target*

Target using the x86 host system

Mainly interesting to easy testing and debugging because benchmarking is not possible.

DEFAULTS = {'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'print_outputs': False, 'repeat': None}

FEATURES = ['benchmark', 'gdbserver']

exec(*program*, **args*, ***kwargs*)

Use target to execute a executable with given arguments

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_arch()

mlonmcu.target.metrics module

class mlonmcu.target.metrics.Metrics

Bases: object

add(*name*, *value*, *optional=False*, *overwrite=False*, *prepend=False*)

static from_csv(*text*)

get(*name*)

get_data(*include_optional=False*, *identify_optional=False*)

has(*name*)

to_csv(*include_optional=False*)

mlonmcu.target.target module

MLonMCU Target definitions

class mlonmcu.target.target.Target(*name: str*, *features: List[*mlonmcu.feature.feature.Feature*] = None*, *config: dict = None*)

Bases: object

Base target class

Attributes

name [str] Default name of the target

features [list] List of target features which should be enabled

config [dict] User config defined via key-value pairs

inspect_program [str] Program which can be used to inspect executables (i.e. readelf)

inspect_program_args [list] List of additional arguments to the inspect_program

env [os._Environ] Optinal map of environment variables

DEFAULTS = {'print_outputs': False, 'repeat': None}

FEATURES = ['benchmark']

OPTIONAL = []

REQUIRED = []

add_platform_defs(*platform*, *defs*)

exec(*program: pathlib.Path*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

export_artifacts(*path*)

generate(*elf*) → Tuple[dict, dict]

generate_artifacts(*elf*)

get_arch()

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_target_system()

inspect(*program: pathlib.Path*, **args*, ***kwargs*)

Use target to inspect a executable

property print_outputs

process_features(*features*)

property repeat

Module contents

MLonMCU target submodule

class mlonmcu.target.**Corstone300Target**(*name='corstone300'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.target.Target*

Target using an ARM FVP (fixed virtual platform) based on a Cortex M55 with EthosU support

DEFAULTS = {'enable_dsp': False, 'enable_ethosu': False, 'enable_fpu': True, 'enable_mvei': False, 'ethosu_num_macs': 256, 'extra_args': '', 'model': 'cortex-m55', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0}

FEATURES = ['ethosu', 'arm_mvei', 'arm_dsp']

REQUIRED = ['corstone300.exe', 'cmsisnn.dir', 'arm_gcc.install_dir']

property cmsisnn_dir

property enable_dsp

property enable_ethosu

property enable_fpu

property enable_mvei

property ethosu_num_macs

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extra_args

property fvp_exe

property gcc_prefix

get_arch()

get_backend_config(*backend*)

get_default_fvp_args()

get_ethosu_fvp_args()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

property model

parse_stdout(*out*, *handle_exit=None*)

property timeout_sec

class mlonmcu.target.EtissPulpinoTarget(*name='etiss_pulpino'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a Pulpino-like VP running in the ETISS simulator

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'cpu_arch': None, 'cycle_time_ps': 31250, 'debug_etiss': False, 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'jit': None, 'pext_spec': 0.96, 'plugins': [], 'print_outputs': False, 'ram_size': 67108864, 'ram_start': 8388608, 'repeat': None, 'rom_size': 8388608, 'rom_start': 0, 'timeout_sec': 0, 'trace_memory': False, 'verbose': False, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'gdbserver', 'etissdbg', 'trace', 'log_instrs', 'pext', 'vext']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'etiss.src_dir', 'etiss.install_dir', 'etissvp.script']

property attr

property cpu_arch

property cycle_time_ps

property debug_etiss

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

property etiss_dir

property etiss_script

property etiss_src_dir

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_target_system()

property jit

parse_stdout(*out*, *handle_exit=None*)

property pext_spec

property plugins

property ram_size

property ram_start

property rom_size

property rom_start

property trace_memory

property verbose

property vext_spec

property vlen

write_ini(*path*)

class mlonmcu.target.**HostX86Target**(*name='host_x86'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.target.Target*

Target using the x86 host system

Mainly interesting to easy testing and debugging because benchmarking is not possible.

DEFAULTS = {'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'print_outputs': False, 'repeat': None}

FEATURES = ['benchmark', 'gdbserver']

exec(*program*, **args*, ***kwargs*)

Use target to execute a executable with given arguments

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_arch()

class mlonmcu.target.OVPSimTarget(*name='ovpsim'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using an ARM FVP (fixed virtual platform) based on a Cortex M55 with EthosU support

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'bitmanip_spec': 0.94, 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': True, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'gdbserver_attach': False, 'gdbserver_enable': False, 'gdbserver_port': 2222, 'pext_spec': 0.96, 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'variant': None, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext', 'pext', 'gdbserver', 'log_instrs', 'trace']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'ovpsim.exe']

property attr

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

property gdbserver_attach

property gdbserver_enable

property gdbserver_port

get_backend_config(*backend*)

get_default_ovpsim_args()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

property ovpsim_exe

parse_stdout(*out*)

property pext_spec

property variant

property vext_spec

property vlen

class mlonmcu.target.RiscvQemuTarget(*name='riscv_qemu'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using a spike machine in the QEMU simulator

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'elen': 32, 'embedded_vext': False, 'enable_vext': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'print_outputs': False, 'repeat': None, 'timeout_sec': 0, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'riscv32_qemu.exe']

property attr

property elen

property embedded_vext

property enable_vext

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

get_cpu_str()

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_qemu_args(*program*)

get_target_system()

parse_stdout(*out*, *handle_exit=None*)

property riscv32_qemu_exe

property vext_spec

property vlen

class mlonmcu.target.SpikeTarget(*name='spike'*, *features=None*, *config=None*)

Bases: *mlonmcu.target.riscv.riscv.RISCVTarget*

Target using the riscv-isa-sim (Spike) RISC-V simulator.

DEFAULTS = {'abi': None, 'arch': None, 'attr': '', 'elen': 32, 'embedded_vext': False, 'enable_pext': False, 'enable_vext': False, 'end_to_end_cycles': False, 'extensions': ['i', 'm', 'a', 'c'], 'extra_args': '', 'fpu': 'double', 'pext_spec': 0.92, 'print_outputs': False, 'repeat': None, 'spikepk_extra_args': [], 'timeout_sec': 0, 'vext_spec': 1.0, 'vlen': 0, 'xlen': 32}

FEATURES = ['benchmark', 'vext', 'pext', 'cachesim', 'log_instrs']

REQUIRED = ['riscv_gcc.install_dir', 'riscv_gcc.name', 'riscv_gcc.variant', 'spike.exe', 'spike.pk']

property attr

property elen

property embedded_vext

property enable_pext

property enable_vext

property end_to_end_cycles

exec(*program*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

property extensions

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

parse_stdout(*out*)

property pext_spec

property spike_exe

property spike_pk

property spikepk_extra_args

property vext_spec

property vlen

class mlonmcu.target.**Target**(*name: str*, *features: List[*mlonmcu.feature.feature.Feature*] = None*, *config: dict = None*)

Bases: object

Base target class

Attributes

name [str] Default name of the target

features [list] List of target features which should be enabled

config [dict] User config defined via key-value pairs

inspect_program [str] Program which can be used to inspect executables (i.e. readelf)

inspect_program_args [list] List of additional arguments to the inspect_program

env [os._Environ] Optinal map of environment variables

DEFAULTS = {'print_outputs': False, 'repeat': None}

FEATURES = ['benchmark']

OPTIONAL = []

REQUIRED = []

add_platform_defs(*platform*, *defs*)

exec(*program: pathlib.Path*, **args*, *cwd='/home/docs/checkouts/readthedocs.org/user_builds/mlonmcu/checkouts/stable/docs'*, ***kwargs*)

Use target to execute a executable with given arguments

export_artifacts(*path*)

generate(*elf*) → Tuple[dict, dict]

generate_artifacts(*elf*)

get_arch()

get_backend_config(*backend*)

get_metrics(*elf*, *directory*, **args*, *handle_exit=None*)

get_platform_defs(*platform*)

get_target_system()

inspect(*program: pathlib.Path*, **args*, ***kwargs*)

Use target to inspect a executable

property print_outputs

process_features(*features*)

property repeat

mlonmcu.target.**get_targets**()

mlonmcu.target.**register_target**(*target_name*, *t*, *override=False*)

Submodules

mlonmcu.artifact module

Artifacts defintions internally used to refer to intermediate results.

class mlonmcu.artifact.**Artifact**(*name*, *content=None*, *path=None*, *data=None*, *raw=None*, *fmt=ArtifactFormat.UNKNOWN*, *flags=None*, *archive=False*, *optional=False*)

Bases: object

Artifact type.

export(*dest*, *extract=False*)

Export the artifact to a given path (file or directory) and update its path.

property exported

Returns true if the artifact was writtem to disk.

print_summary()

Utility to print information about an artifact to the cmdline.

validate()

Checker for artifact attributes for the given format.

class mlonmcu.artifact.**ArtifactFormat**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: enum.Enum

Enumeration of artifact types.

BIN = 11

DATA = 6

IMAGE = 5

JSON = 9

MLF = 3

MODEL = 4

NUMPY = 7

PARAMS = 8

PATH = 10

RAW = 11

SHARED_OBJECT = 12

SOURCE = 1

TEXT = 2

UNKNOWN = 0

mlonmcu.artifact.**lookup_artifacts**(*artifacts*, *name=None*, *fmt=None*, *flags=None*, *first_only=False*)

Utility to get a matching artifact for a given set of properties.

mlonmcu.config module

Collection of utilities to manage MLonMCU configs.

mlonmcu.config.**filter_config**(*config*, *prefix*, *defaults*, *optionals*, *required_keys*)

Filter the global config for a given component prefix.

Returns

cfg [dict] The filteres configuration.

Raises

AssertionError: If a required key is missing.

mlonmcu.config.**remove_config_prefix**(*config*, *prefix*, *skip=None*)

Iterate over keys in dict and remove given prefix.

Returns

ret [dict] The transformed configuration.

mlonmcu.config.**resolve_required_config**(*required_keys*, *optional=None*, *features=None*, *config=None*, *cache=None*, *hints=None*, *default_flags=None*)

Utility which iterates over a set of given config keys and resolves their values using the passed config and/or cache.

Parameters

required_keys [List[str]]

features [List[Feature]]

config [dict]

cache [TaskCache] Optional task cache parsed from the *cache.ini* file in the *deps* directory.

hints [List[str]] List of additional flags which can be provided as a hint to lookup a cache config.

default_flags [dict] User-provided mapping of cache flags for some cache entries.

Returns

result [dict]

mlonmcu.config.**str2bool**(*value*, *allow_none=False*)

mlonmcu.config.**str2dict**(*value*, *allow_none=False*)

mlonmcu.config.**str2list**(*value*, *allow_none=False*)

mlonmcu.logging module

Loging utilities for MLonMCU

mlonmcu.logging.**get_formatter**(*minimal=False*)

Returns a log formatter for one on two predefined formats.

mlonmcu.logging.**get_logger**()

Helper function which return the main mlonmcu logger while ensuring that is is properly initialized.

mlonmcu.logging.**set_log_file**(*path*, *level=10*, *rotate=False*)

Enable logging to a file.

mlonmcu.logging.**set_log_level**(*level*)

Set command line log level at runtime.

mlonmcu.mlonmcu module

Main module.

mlonmcu.plugins module

Utilities for MLonMCUs extension mechanism.

mlonmcu.plugins.**process_extensions**(*file*)

mlonmcu.report module

Definitions of the Report class used by MLonMCU sessions and runs.

class mlonmcu.report.**Report**

Bases: object

Report class wrapped around multiple pandas dataframes.

add(*reports*)

Helper function to append a line to an existing report.

property df

Combine the three internal dataframes to a large one and return in.

export(*path*)

Export the report to a file.

set(*pre=None*, *main=None*, *post=None*)

Setter for the dataframe.

set_main(*data*)

Setter for the center part of the dataframe.

set_post(*data*)

Setter for the right third of the dataframe.

set_pre(*data*)

Setter for the left third of the dataframe.

mlonmcu.utils module

mlonmcu.utils.**ask_user**(*text*, *default: bool*, *yes_keys=['y', 'j']*, *no_keys=['n']*, *interactive=True*)

mlonmcu.utils.**get_base_prefix_compat**()

Get base/real prefix, or sys.prefix if there is none.

mlonmcu.utils.**in_virtualenv**()

Detects if the current python interpreter is from a virtual environment.

mlonmcu.utils.**is_power_of_two**(*n*)

mlonmcu.version module

Version module for mlonmcu.

Module contents

Top-level package for ML on MCU.

CHAPTER

ELEVEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

11.1 Types of Contributions

11.1.1 Report Bugs

Report bugs at https://github.com/tum-ei-eda/mlonmcu/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

11.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

11.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

11.1.4 Write Documentation

ML on MCU could always use more documentation, whether as part of the official ML on MCU docs, in docstrings, or even on the web in blog posts, articles, and such.

11.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tum-ei-eda/mlonmcu/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

11.2 Get Started!

Ready to contribute? Here's how to set up `mlonmcu` for local development.

1. Fork the `mlonmcu` repo on GitHub.
2. Clone your fork locally

```
$ git clone git@github.com:your_name_here/mlonmcu.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mlonmcu
$ cd mlonmcu/
$ python setup.py develop
```

4. Create a branch for local development

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox

```
# TODO
$ flake8 mlonmcu tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

11.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/tum-ei-eda/mlonmcu/pull_requests and make sure that the tests pass for all supported Python versions.

11.4 Tips

To run a subset of tests

```
```
# TODO
$ python -m unittest tests.test_mlonmcu
```
```

11.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass. # TODO

CHAPTER

TWELVE

CREDITS

12.1 Development Lead

- TUM Department of Electrical and Computer Engineering - Chair of Electronic Design Automation mailto:philipp.van-kempen@tum.de

12.2 Contributors

None yet. Why not be the first?

CHAPTER

THIRTEEN

HISTORY

13.1 0.1.0 (2021-11-12)

- First closed-source release.

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z