Open-CMSIS-Pack
Technical Review Meeting

CMSIS Team

23. Aug 2022

arm

Introducing Layers

Current Examples that use Layers

Provide Layers from Board Support Packs (or other Packs)

Examples that uses TF-M with Boot Loader

Issues with the RTE Folder when using Layers

Csolution Examples

https://github.com/Open-CMSIS-Pack/csolution-examples

AWS_MQTT_MutualAuth_Demo

Target-Types

- IP-Stack: uses TCP/IP Stack connected via CMSIS-Driver Ethernet
- WiFi: uses TCP/IP CMSIS-Driver WiFi
- AVH: uses IoT-Socket (Vsocket on Arm Virtual Hardware)

All have a shim-layer to AWS “iot-secure-sockets”

Each of the Target-Types is also mapped to hardware, right now:

- IP-Stack: compatible with CMSIS-Driver Ethernet
- WiFi: compatible with CMSIS-Driver WiFi
- AVH: compatible with IoT-Socket

```
layers:
  - layer: ./Board/IMXRT1050-EVKB/Board.clayer.yml
    for-type:
      - +IP-Stack
  - layer: ./Board/B-U585I-IOT02A/Board.clayer.yml
    for-type:
      - +WiFi
  - layer: ./Board/AVH_MPS3_Corstone-300/Board.clayer.yml
    for-type:
      - +AVH
```

How can layers come from Board Support Packs (BSP)

- Reference Code Examples can pick-up the right layer BSPs
- Multiple pre-configured layers can be provided to support typical use cases
- Selection could depend on open component requirements or required interfaces

Potential YML solutions:

```
layers:
    - layer-template: board-IoT          # works with layer
      interfaces:
        - consumes:
          - api: CMSIS Driver:USART   // can be automatic
          - Heap: 65536
```

- Would require user interaction to copy the right layer and modify cproject.yml
- IDE (or alternative tool) could support this interaction
- Gives more control to the user; sharing multiple layers between projects
- With RTE-Folder proposal (slide 5) it allows multiple layers (i.e. for processor 1 and processor 2 in case of ASP) to share a common set of configuration files.

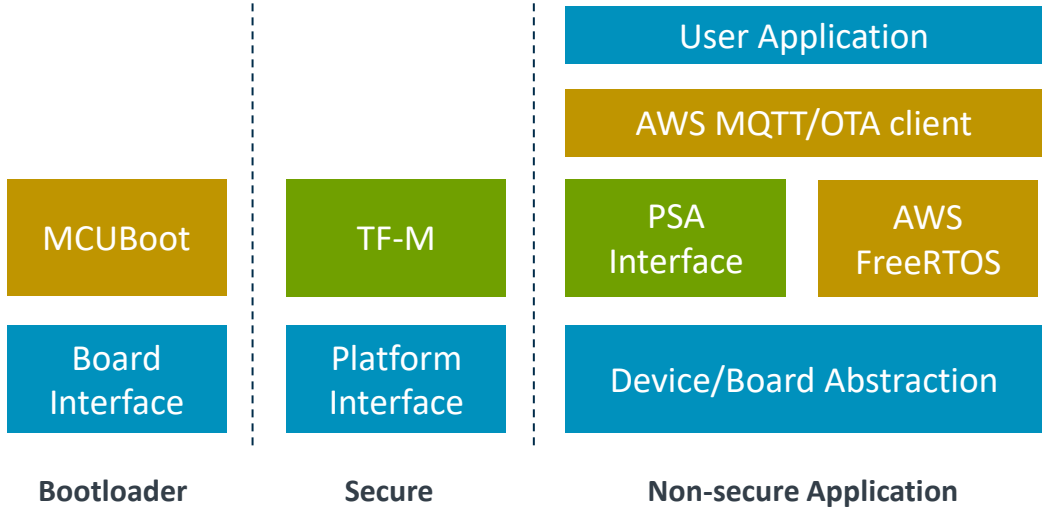
Alternative (more automatic but perhaps in-transparent) solution is shown on next slide.

arm

Example Project: Cloud Connector using MCUBoot and TF-M

Security and Firmware Update for TrustZone enabled Devices

Architecture



Software Packs for Device/Board

AVH-CM33 BSP/DFP	STM32U5 DFP	B-U585-IOT02A BSP
{psa&tfm} image {mcuboot} project {board} layer	{psa&tfm} image {mcuboot} project	{board} layer {mcuboot-board} layer

MyApplication.csolution.yml

```yaml
target-types:
  - type: AVH                 # simulated target for CI testing
    board: AVH-CM33           # Arm Virtual Hardware Cortex-M33 board
  - type: Board
    board: B-U585I-IOT02A     # STM32U5 Test Board

  - type: Custom              # simulated target for CI testing
    device: STM32U575RG       # device in custom hardware

projects:
  - project: myCloudApp.cproject.yml
  - image: {psa&tfm}          # pre-build image, target-specific
  - project: {mcuboot}        # target-specific project
```

myCloudApp.cproject.yml

```yaml
layers:
  - layer: interface.clayer.yml
  - layer: {board}            # target-specific board interface
    not-for-type: .Custom

  - layer: myHardware.clayer.yml
    for-type: .Custom
```

`{name&variant}` are copied from software packs to the project

[Read more on Open-CMSIS-Pack-Spec: Add <csolution> element to *.PDSC format]

RTE Folder – Local Configuration for Layers?

Configuration Files for Components

Current: All Config Files share same RTE

RTE Benefits

- Support for Product Life Cycle management
- Component configuration has a central place

./RTE/

```
Board_Support/MIMXRT1052DVL6B
Board_Support/STM32U565AIIx
Compiler/
CMSIS Driver/STM32U565AIIx
CMSIS Driver
Device/MIMXRT1052DVL6B
Device/STM32U565AIIx
Device/SSE-300-MPS3
```

Issues

- Potential conflicts when layers use different configurations
- Unclear how to remove a layer including related config files

RTE Self-Contained in Layer

```
layer:          // preferred solution by RK
  RTE-paths:
    - Board_Support: <local-path>
    - Compiler: <local-path>
    - CMSIS Driver: <local-path>
    - Device: <local-path>
```

```
layer:        // alternative solution
  RTE:
    path: <local-path>
    component-classes:
      - Board_Support
      - Compiler
      - CMSIS Driver
      - Device
```

<local-path>

- Relative to clayer.yml
- Makes component classes managed by a layer obvious
- Layers can be shared with multiple projects

arm

Pack Generation

Start to engage with wider Industry
Pack Generation Examples

Common Device Interfaces, where to start

- IoT-Socket, PSA
- What's wrong with CMSIS-Driver

CI System for Validation of Software Stacks

Pack Generation Examples

How packs are generated in practice

github.com/MDK-Packs/IoT_Socket - Native Pack project, PDSC file manually created

- IoT-Socket interface that is proposed in Open-CMSIS-CDI, during development, the repository can be directly accessed as pack (using cpackget)
- CMSIS utilities are used to validate the creation (XML schema check, PackChk), gen_pack.sh script is used to create the final pack
- Distribution of public packs uses a separate github repository (github.com/MDK-Packs/Pack)
- Pack Index file gives a vendor full control over the pack publishing process

github.com/lvgl/lvgl/tree/master/env_support/cmsis-pack - Graphic Library that uses gen_pack.sh

- PDSC file is created and maintained manually

https://github.com/MDK-Packs/tensorflow-pack - TFLu project + Arm ML components

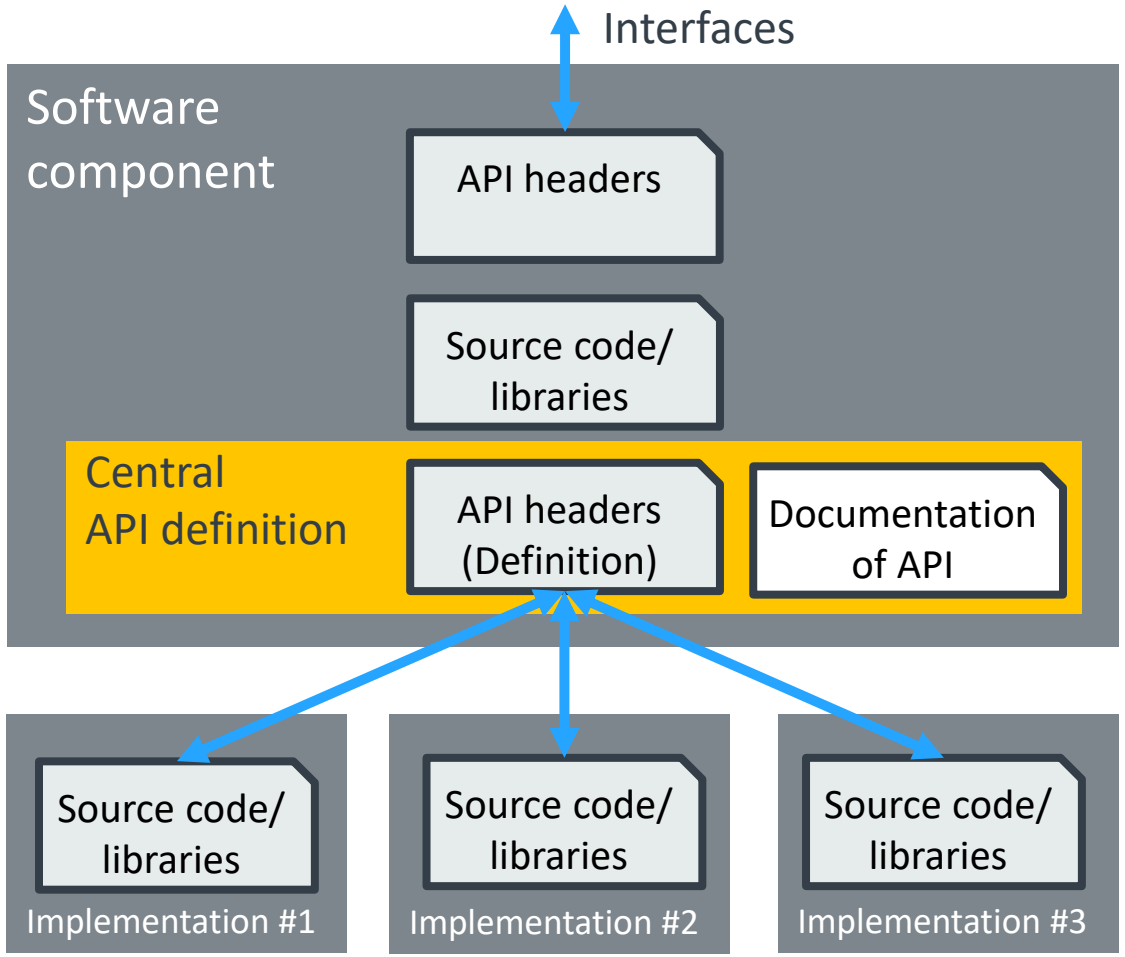
- Pack generation (PDSC file) is automated with Python scripts and derived from the underlying open-source projects.

https://github.com/FreeRTOS/CMSIS-Packs - AWS FreeRTOS packs (created from CMake based projects)

- Pack generation (PDSC file) is automated PackGen and manifest.yml file

CDI-Pack: Central API Interface definition in CMSIS-Pack format

Ensuring consistent interfaces and naming taxonomy across the industry



- Organizes the taxonomies of standard APIs that are essential for re-useable software stacks
- Solves a common problem: API headers evolve over time.

A central API definition shares header file and documentation of an API interface across multiple other software components to ensure consistency.

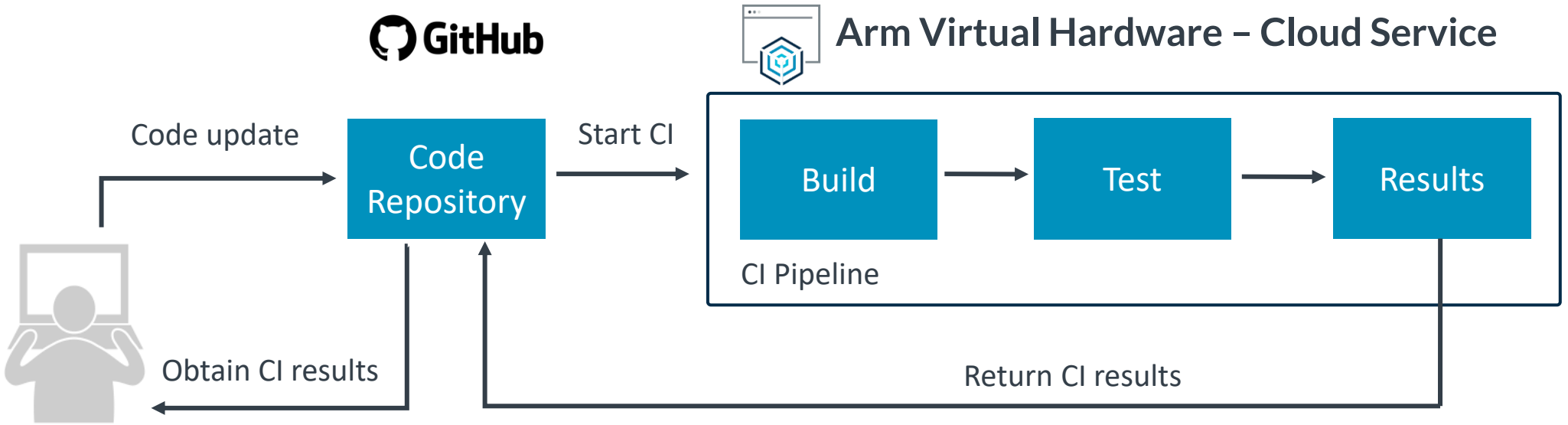
The API interface is distributed separate or as part of the software component that defines this interface. The API header file is therefore consistent.

An example is the CMSIS-Driver pack that contains various Ethernet and Flash drivers – all compatible with the CMSIS-Driver APIs that are published in the CMSIS Pack.

arm

Development Workflow (exemplified with GitHub)

github.com/ARM-software/AVH-GetStarted



1. **Local development:** use a classic embedded toolchain such as Keil MDK and with Arm Virtual Hardware Target for MCU simulation. A GitHub repository is used as a source code management system for synchronization, storage and version control.
2. **CI pipeline setup:** a GitHub Action implements the CI pipeline that gets triggered on every code update in the target repository.
3. **CI execution:** automated program build and testing with cloud-based Arm Virtual Hardware; results reported back to repository.
4. **Failure analysis and local debug:** developer can observe the CI test results. Failures can be reproduced and debugged locally.

Arm Virtual Hardware (AVH) at AWS Marketplace

Complete software toolset with AVH Fast Models for Corstone and Cortex-M CPUs

- **CI/CD Usage**
  avhclient controls AWS infrastructure
  - start / stop EC2 instances
  - upload / run / download
  - integrates with git services such as:

GitHub GitLab circleci Jenkins

- **Interactive Usage**
  SSH connection to remote machine
  - Linux environment for build, test and debug.
  - IDE interface via VS Code

- **MLOps Usage**
  optimize Machine Learning (ML) models

Arm Virtual Hardware – AWS cloud infrastructure

AWS EC2 – Elastic Cloud Compute

A secure and scalable compute server that runs the AMI. Cost effective as it starts and stops jobs on demand.

AWS S3 – Simple Storage Service

A temporary file storage for the build and test process. Available during EC2 execution of the AMI.

AWS EFS – Elastic File System

A permanent file storage that is project-specific. Stores artifacts such as software components or test scripts.

AWS AMI
Amazon Machine Image

A ready-to-use configuration of standard software development tools for IoT, ML, and embedded.

- Ubuntu Linux
- AVH Fast Models for Corstone and Cortex-M
- Arm & GCC C/C++ Compiler
- CMake, CMSIS-Toolbox, Python, …

Runs on AWS data centers that are available within different geographic regions.

arm

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה