Open-CMSIS-Pack
Technical Review Meeting

CMSIS Team
30. Aug 2022

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

Opportunity: Packs give flexibility to the SW Eco-system

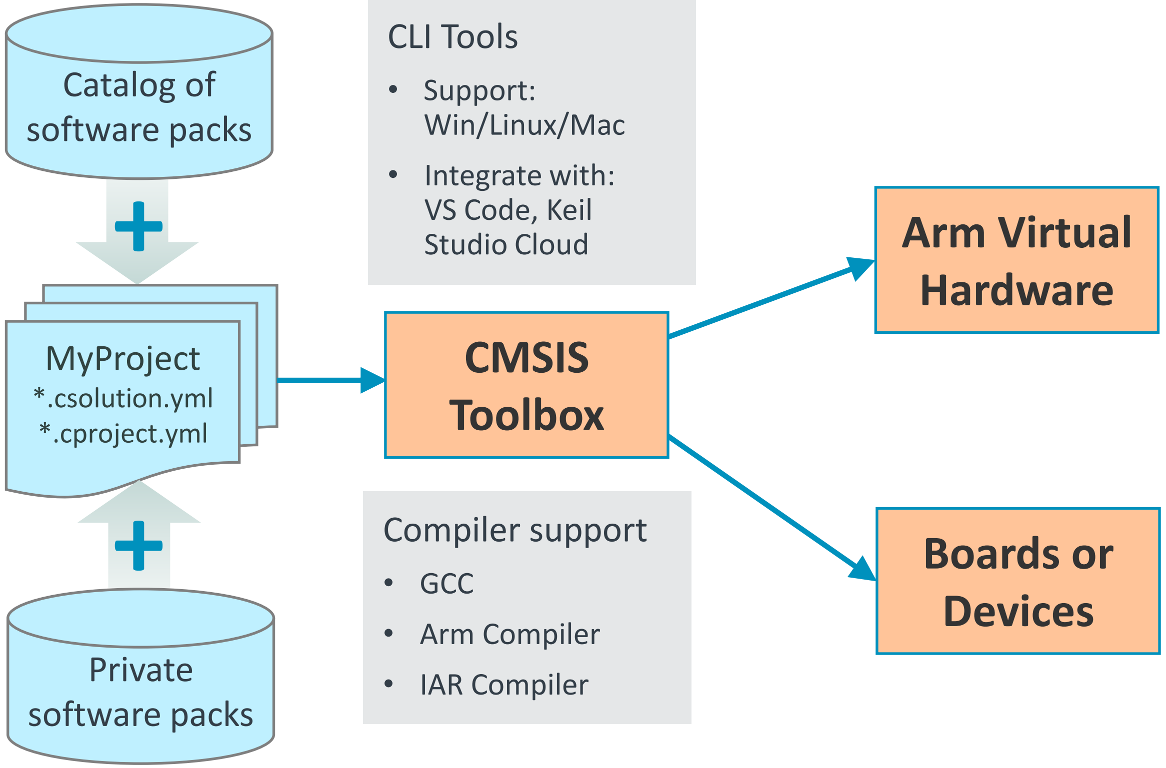
Flexible Development Workflows with Open-CMSIS-Pack

Standard Software exposed to developers:

- Web portals
- Inside IDEs

Confidential software

- Git repo for development
- Released software with versioning



CLI Tools

- Support: Win/Linux/Mac
- Integrate with: VS Code, Keil Studio Cloud

Compiler support

- GCC
- Arm Compiler
- IAR Compiler

AVH Integrates into:

- CI, DevOps, MLOps systems (i.e. GitHub)
- Keil MDK, Arm DS, IAR EW-Arm
- Keil Studio Cloud

AVH Supports:

- CI Validation
- Evaluation
- Training

Platform for Reference Applications:

- Many different boards???

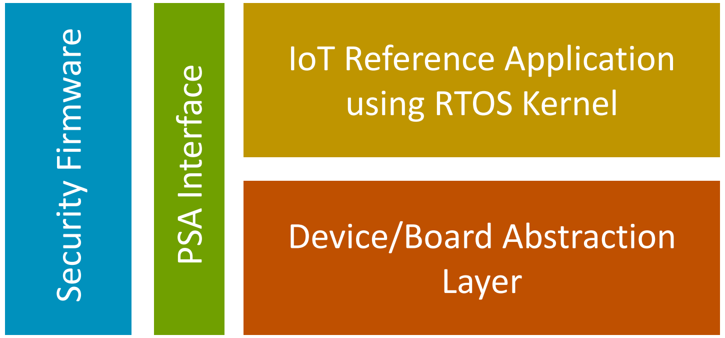
Other tool integrations:

- Arm DS, Keil MDK, IAR EW-Arm, Eclipse
- NXP?
- ST?

github.com/Open-CMSIS-Pack

What do we want to achieve? Plug and Play of SW Building Blocks

Reference Application Framework: map many applications to many boards



SW Building Blocks

- Should come from multiple vendors. Requirement for standardized interface between the components (Open-CMSIS-CDI)
- Reference Application: should be tested with a CI system against a standardized CDI framework
- Should run (within reason) on many different existing v8M and v7M devices (TrustZone optional)
- Should include OTA services with standardize interfaces
- Future variants of the Framework should also support other application types (DSP, ML, Graphics)

Interface Requirements:

IoT Reference Application: assumption connects via WiFi or wired Ethernet

Uses:

- UART for Text, IoT Socket (for WiFi driver or VSocket), Ethernet (for TCP/IP Stack)
- PSA Interface with Storage, Crypto, (OTA optional)
- Heap

Provides:

- Minimum Thread control (wait feature)

Board Layer:

Provides:

- UART for Text, IoT Socket (for WiFi driver or VSocket), Ethernet (for TCP/IP Stack), Heap
- Future interfaces may support other Reference Applications (i.e. for ML Sensor, Audio applications)
- Optional features: Event Recorder?

Security Firmware:

- Based on TF-M framework for TrustZone or mbedTLS for non-TrustZone devices

Other Requirements:

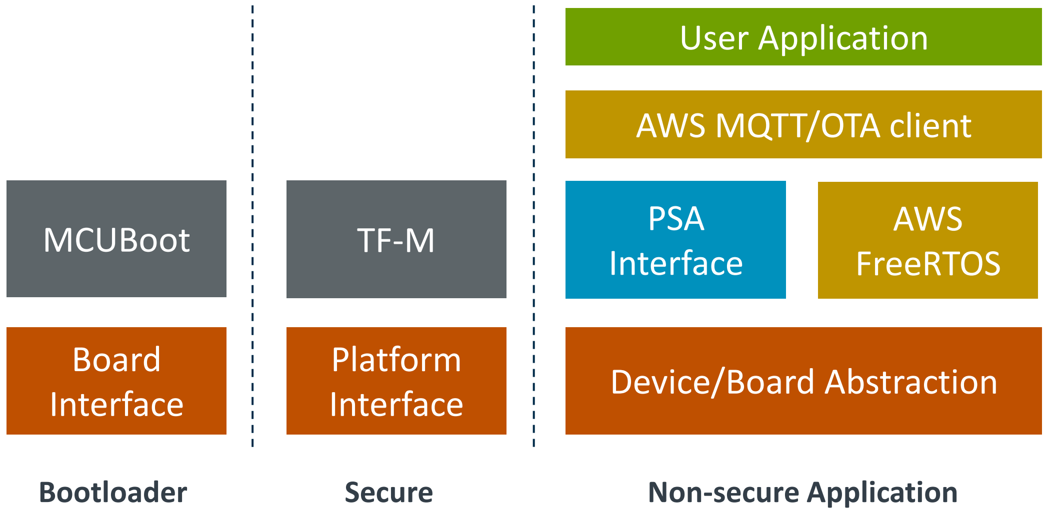
Defined Startup/Call Sequence (see https://github.com/MDK-Packs/CB_Lab4Layer/tree/master/layer)

- Example: https://github.com/MDK-Packs/CB_Lab4Layer/blob/master/layer/Board/MIMXRT1064-EVK/main.c

Can we start with an example project?

Based on https://github.com/mdk-packs/TrustZone

Architecture



Software Packs for Device/Board

AVH-CM33 BSP/DFP
image for PSA / TFM
image for bootloader
layer for board IoT

STM32U5 DFP
image for PSA / TFM
image for boot loader

B-U585-IOT02A BSP
layer for board
variant “IoT”

Development tools could support selection of packs:
https://github.com/Open-CMSIS-Pack/Open-CMSIS-Pack-Spec/issues/134#issuecomment-1174980291

Initial Tasks

Start with an example, based on AWS

- Identify scope of standardized interfaces and create API packs
- Implement standardized interfaces between components
  Do we need a variant with TF-M and a variant without TrustZone
- Board support for AVH and initially one evaluation board (PoC)
- CI test execution with AVH along with interface validation

Extend to other software vendors, i.e. Azure, Matter

- Get feedback from software partners on the interfaces (ongoing)
- Work with selected partners to extend the scope of reference applications

Support more devices and boards

- Commitment from SiPs to implement the standardized interfaces
- Get first implementations for additional boards

Extend to communication technologies

- Implement interfaces to LoRa, BLE, etc.
- Extend scope of interfaces to DSP, ML

Interface: node in cproject.yml / clayer.yml files

cproject.yml

```
Interfaces:
  provides:
  - RTOS2

layer-templates:             # project requires layer templates
  - type: Board              # tool: check for a board layer
    interfaces:
    - Heap: >=50000          # minimum heap configuration
    - CMSIS Driver Ethernet:
      for-type: TCP-IP
    - IoT Socket:
      for-type: WiFi
    - CMSIS Driver USART Print:

# tool identifies compatible layers and lists it, user enters then:
layers:
   - layer: <path to layer.yml>    # tool: check for a board layer
```

clayer.yml

```
layer:
  type: Board
  variant: IoT WiFi
  description: Board setup with WiFi interface
  designed-for:  # key value pairs for gen conditions in PDSC files
    device: device-name
    board: board-name
# for future layer types - ML-framework:  TFLu
# for future layer types - Cloud-Service: Azure

  interfaces:   # interface descriptions
    consumes:
    - RTOS2:
    provides:
    - CMSIS Driver Ethernet: 0       # driver number
    - CMSIS Driver USART Print: 2    # driver number
    - IoT Socket:                    # available
    - Heap : 65536                   # heap size
```


arm

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

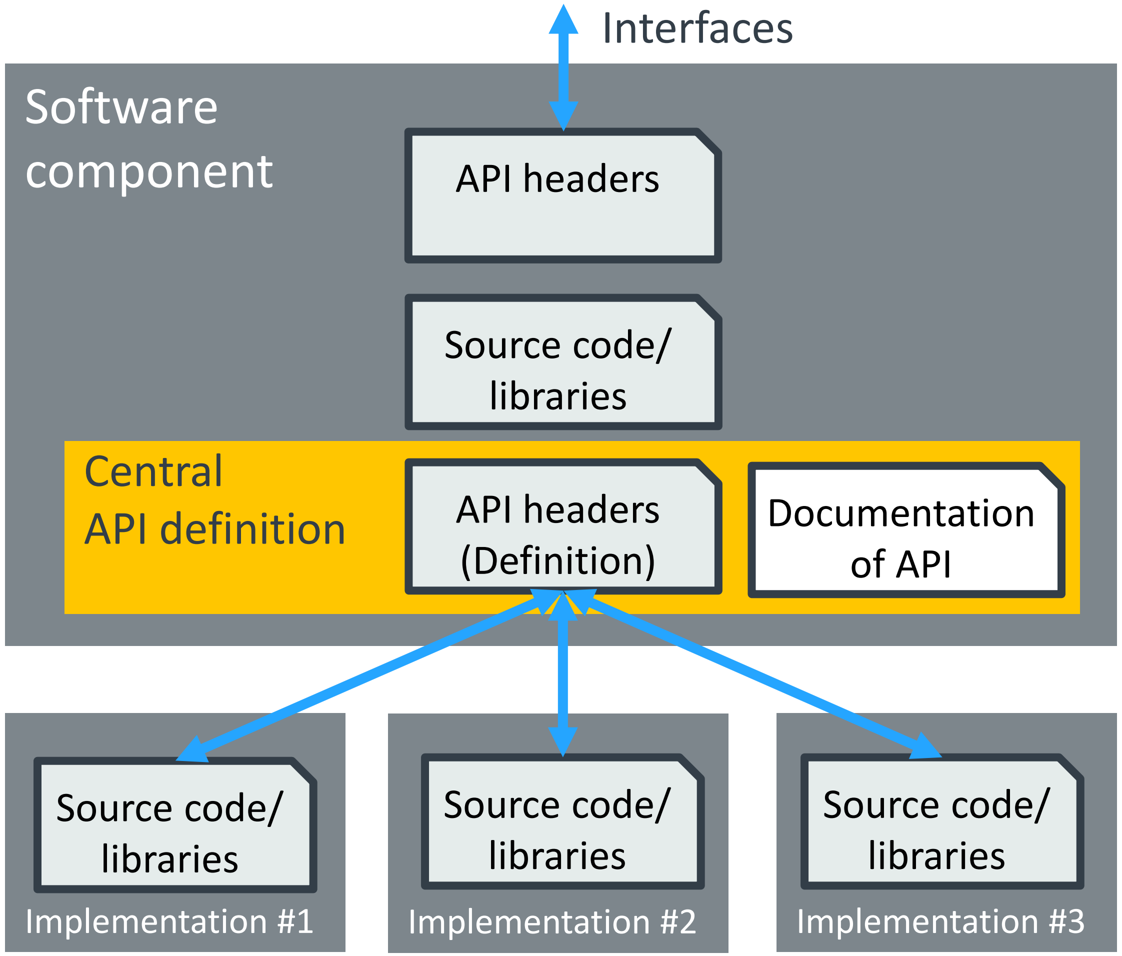
- Pack generation (PDSC file) is automated with Python scripts and derived from the underlying open-source projects.

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.

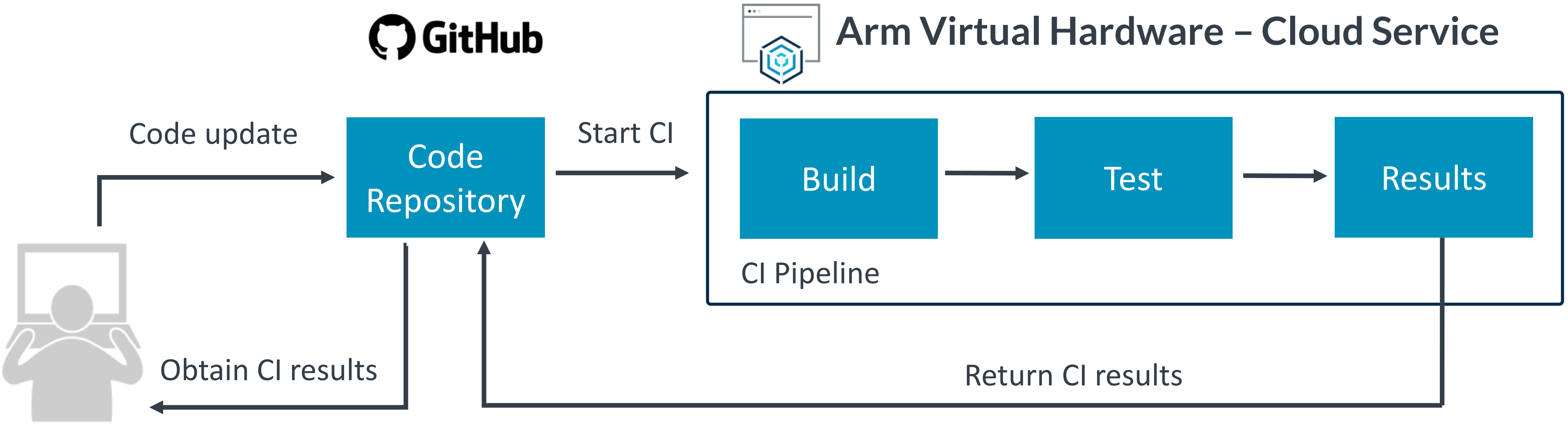
A central API definition shares header file and documentation of an API interface across multiple other software components to ensure consistency.

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.

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.

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