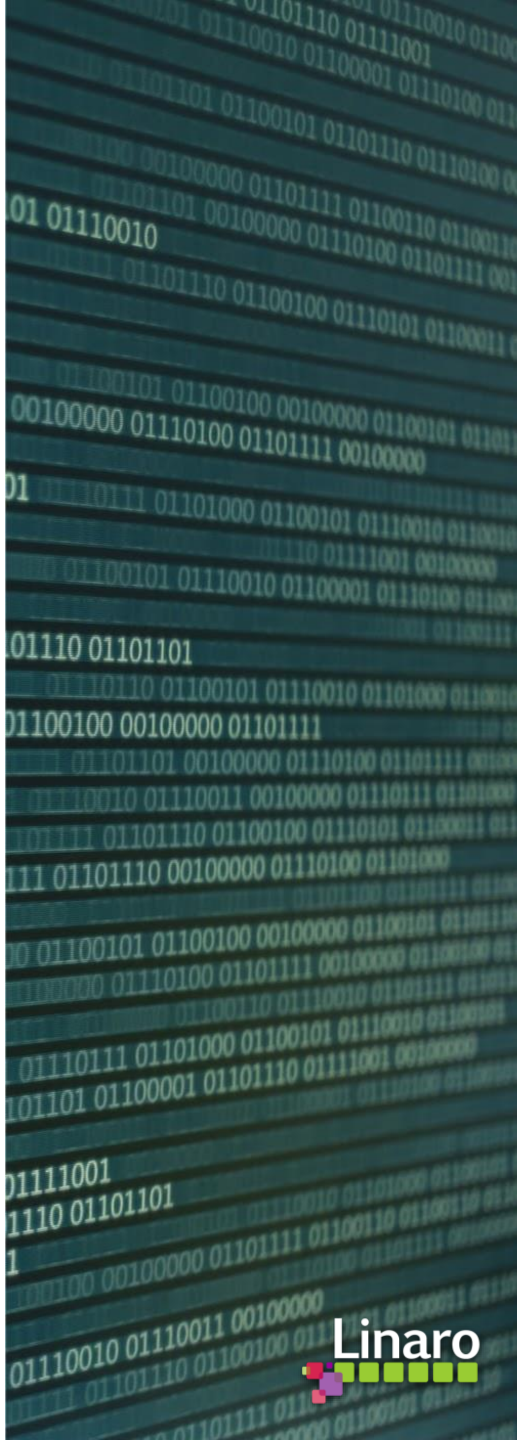
Open-CMSIS-Pack

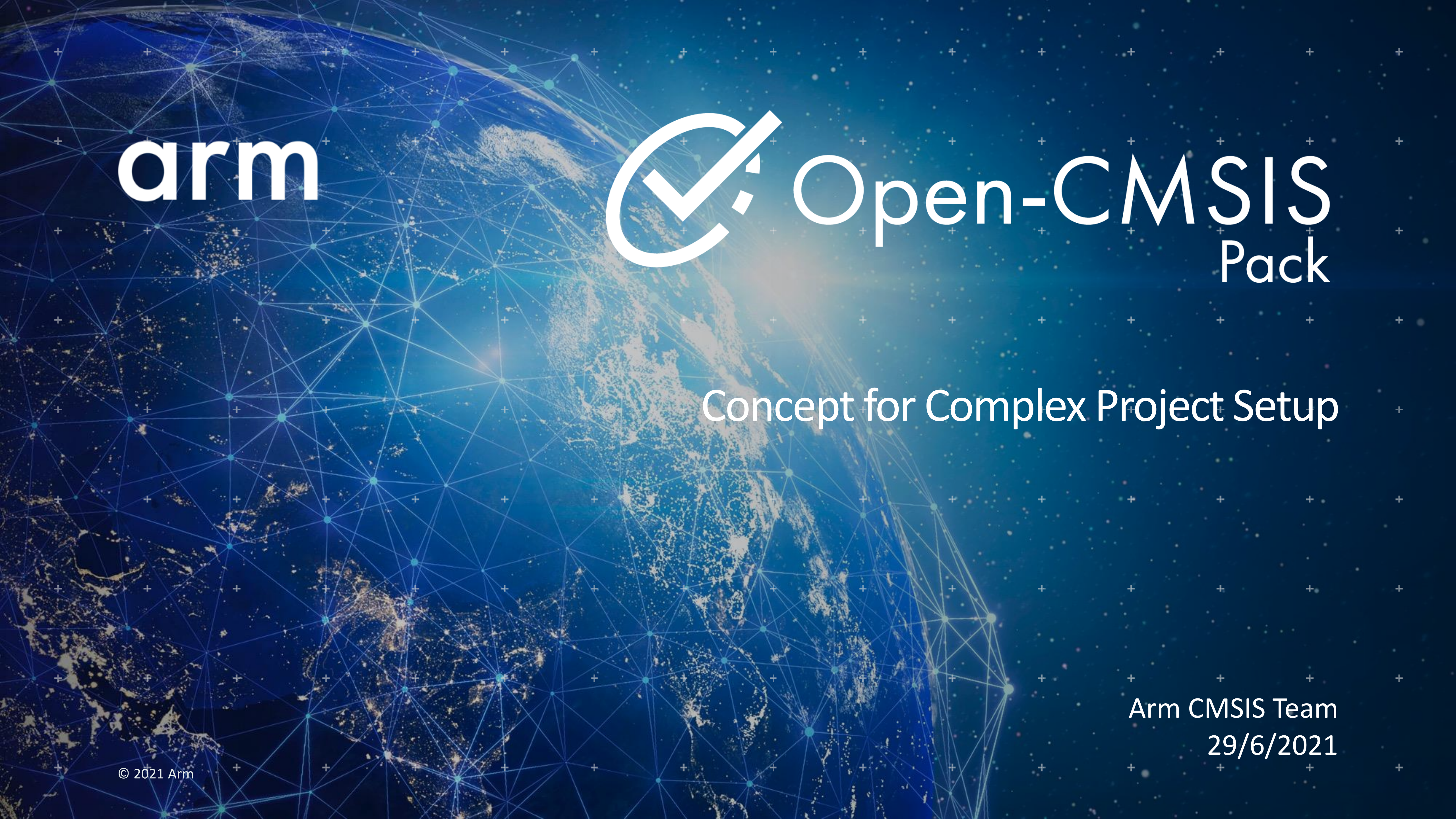
Technical Project Meeting 2021-06-29

This meeting is recorded !

Agenda

- Q&A follow up from CMSIS Review meeting
- Introducing requirements for Multi-Project (Reinhard Keil, Arm)
- Introducing vidx2pidx repository and utility (Charles Oliveira, Linaro)
- Review “way of working”
- Wrap Up
- AoB





arm

Open-CMSIS Pack

Concept for Complex Project Setup

Arm CMSIS Team
29/6/2021

Open-CMSIS-Pack Provisional Roadmap Details

	Jul/Aug 2021	Sep/Oct 2021	Nov/Dec 2021	Jan/Feb 2022	Future
Core Library Components	■ **Base Components** • Open-Source existing Arm implementations	● **Project Format** • CPRJ project examples • UX improvements	⬢ **Collecting Input** • Evolution of specification • Evolution of common components		
Overall Project Concept	■ **Base Technology** • Cmake to Pack conversion • Multi-Project Targets	● **Target Connection** • Debug and download aspects of Multi-Project Targets			
Resource Management	● **Multi-Project** • Review DeviceTree and CMSIS-Zone • Define structure of “Umbrella” projects for multi-core, etc. • Organize taxonomies of standardized API interfaces		• Implement project management for multi-core and secure/non-secure setups • Refine the layer concept for better code re-use		
PoC Tools	■ **Project Build** • Recreate Cbuild in public GitHub Infra-structure • Close gaps in Cbuild • Pack download/install	◆ **Pack Content** • CMake to Pack Converter • Pack Validation (PackChk)	⬢ **Project Management** • Information from CMSIS-Packs • Many boards with reference designs	⬢ **Configuration** • Text based configuration utility (aka Config Wizard)	
Process Improvements	◆ **Optimize Delivery** • Explore potential ways to secure pack content		● **Example Contribution** • Partners are enabled to submit own examples	● **CI for Example/Pack** • Keep examples up-to-date • Submit process for packs with CI	

Last Update: June 21, 2021

■ Started ◆ Development ● Advanced Planning ⬢ Concept

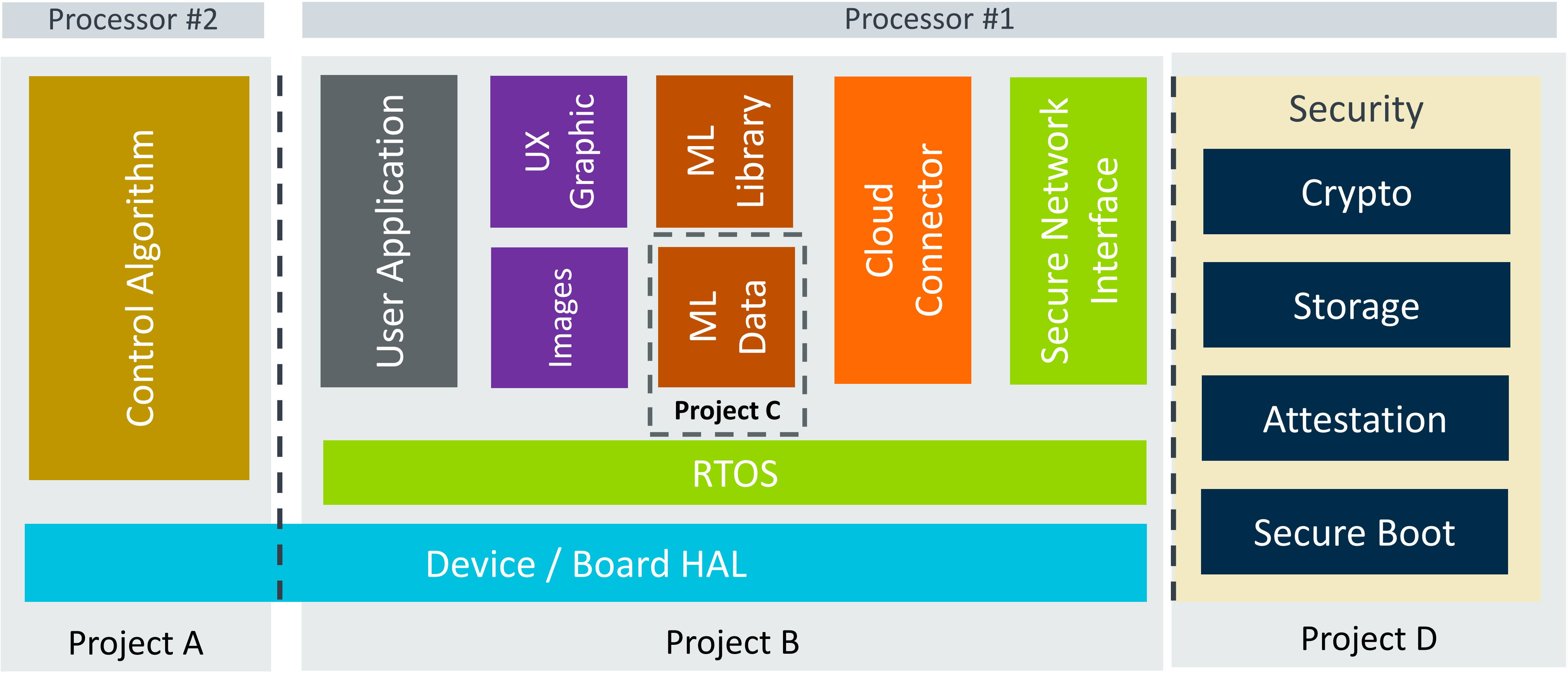
What Requirements should we consider for the future?

Use cases driven by Application Developer

- Holistic view on software projects considering:
  - Structure
    - many dependent/related projects
    - reuse of partial projects
  - Code Generation:
    - build order dependencies
    - multiple build configurations
    - HW resource allocation partitioning and dependencies
    - generated/assisted software configuration
  - Deployment and Download:
    - flash programming setup and configuration
    - Firmware update processes including OTA programming
  - Debugging:
    - debug setup and configuration
- Simplify testing and porting of applications across devices and boards

Multi-Project Requirements

Separate projects independently developed; combined in a multi-project workspace

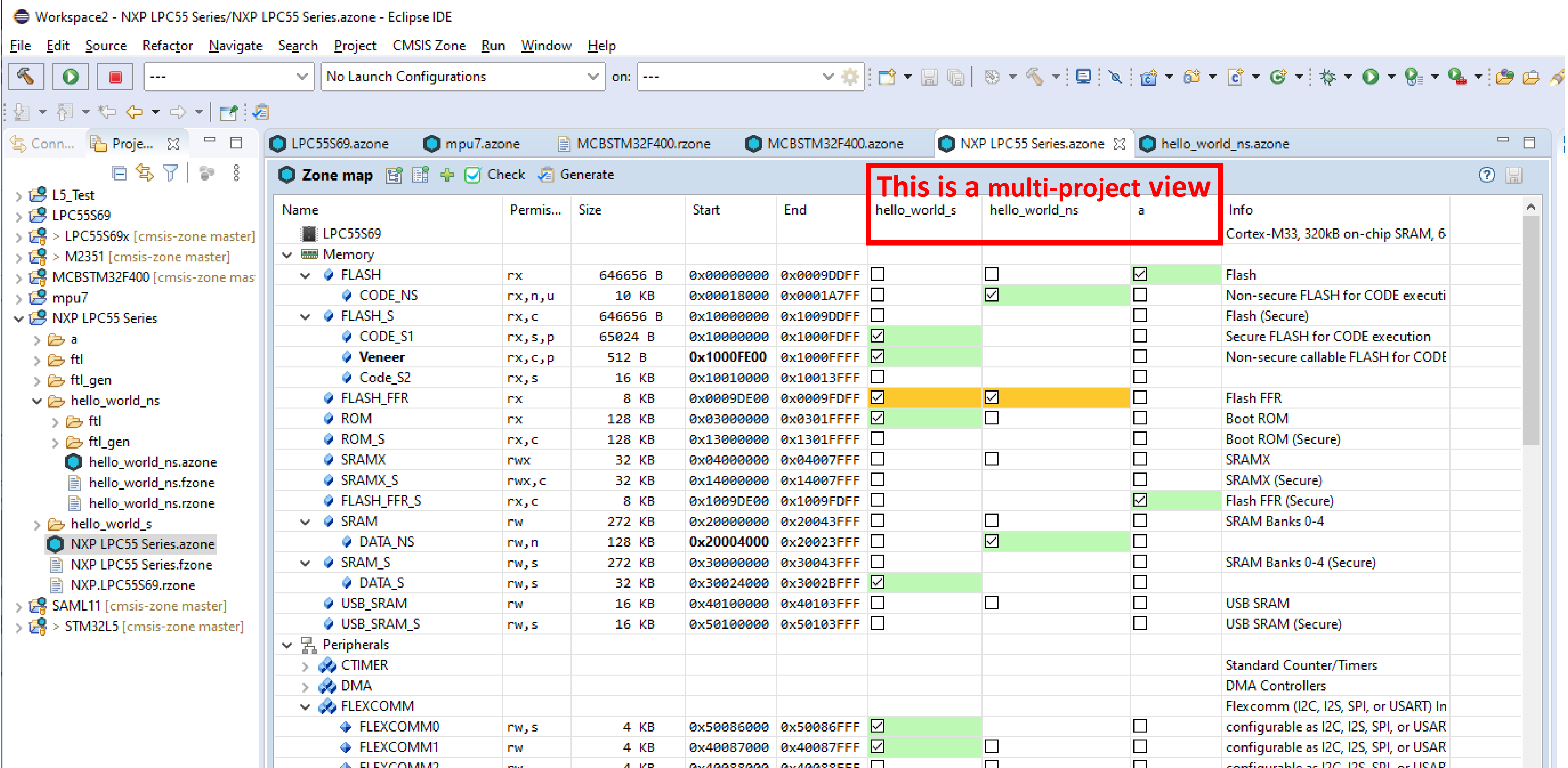


Proposal: Introduce an “umbrella” project file (provisionally called *.ctarget)

Adopt CMSIS-Zone Concepts for Multi-Project Configuration?

Discussion and decision for multi-project configuration: CMSIS-Zone and/or DeviceTree

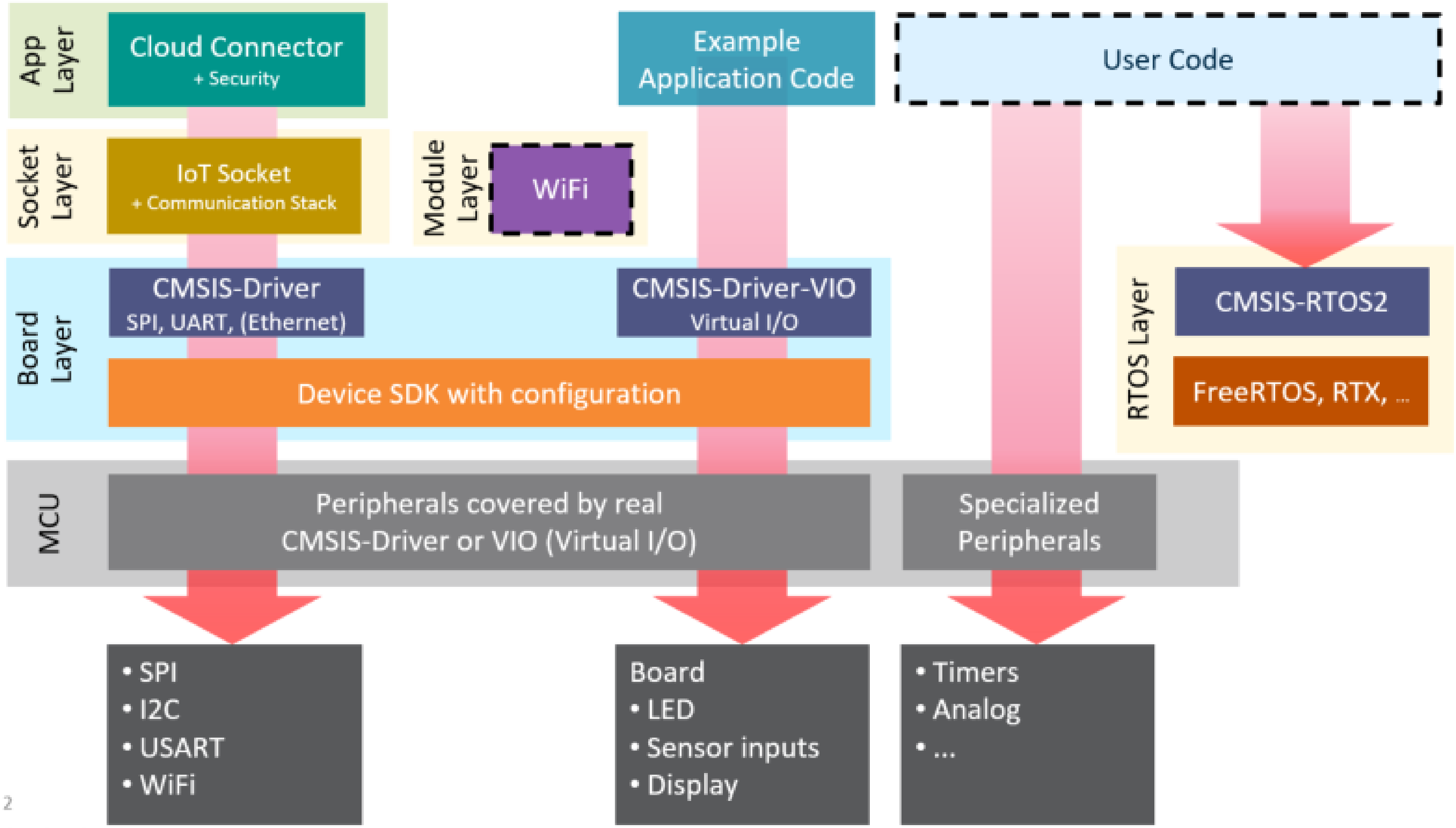
https://community.arm.com/developer/tools-software/tools/b/tools-software-ides-blog/posts/configuring-armv8-m-systems-with-cmsis-zone



Name	Permis...	Size	Start	End	hello_world_s	hello_world_ns	a	Info
LPC55S69								Cortex-M33, 320kB on-chip SRAM, 6
Memory								
FLASH	rx	646656 B	0x00000000	0x0009DDFF	□	□	☑	Flash
CODE_NS	rx,n,u	10 KB	0x00018000	0x0001A7FF	□	☑	□	Non-secure FLASH for CODE executi
FLASH_S	rx,c	646656 B	0x10000000	0x1009DDFF	□		□	Flash (Secure)
CODE_S1	rx,s,p	65024 B	0x10000000	0x1000FDFF	☑		□	Secure FLASH for CODE execution
Veneer	rx,c,p	512 B	**0x1000FE00**	0x1000FFFF	☑		□	Non-secure callable FLASH for CODE
Code_S2	rx,s	16 KB	0x10010000	0x10013FFF	□		□	
FLASH_FFR	rx	8 KB	0x0009DE00	0x0009FDFF	☑	☑	□	Flash FFR
ROM	rx	128 KB	0x03000000	0x0301FFFF	☑	□	□	Boot ROM
ROM_S	rx,c	128 KB	0x13000000	0x1301FFFF	□		□	Boot ROM (Secure)
SRAMX	rwx	32 KB	0x04000000	0x04007FFF	□	□	□	SRAMX
SRAMX_S	rwx,c	32 KB	0x14000000	0x14007FFF	□		□	SRAMX (Secure)
FLASH_FFR_S	rx,c	8 KB	0x1009DE00	0x1009FDFF	□		☑	Flash FFR (Secure)
SRAM	rw	272 KB	0x20000000	0x20043FFF	□	□	□	SRAM Banks 0-4
DATA_NS	rw,n	128 KB	**0x20004000**	0x20023FFF	□	☑	□	
SRAM_S	rw,s	272 KB	0x30000000	0x30043FFF	□		□	SRAM Banks 0-4 (Secure)
DATA_S	rw,s	32 KB	0x30024000	0x3002BFFF	☑		□	
USB_SRAM	rw	16 KB	0x40100000	0x40103FFF	□	□	□	USB SRAM
USB_SRAM_S	rw,s	16 KB	0x50100000	0x50103FFF	□		□	USB SRAM (Secure)
Peripherals								
CTIMER								Standard Counter/Timers
DMA								DMA Controllers
FLEXCOMM								Flexcomm (I2C, I2S, SPI, or USART) In
FLEXCOMM0	rw,s	4 KB	0x50086000	0x50086FFF	☑		□	configurable as I2C, I2S, SPI, or USAR
FLEXCOMM1	rw	4 KB	0x40087000	0x40087FFF	☑	□	□	configurable as I2C, I2S, SPI, or USAR
FLEXCOMM2	rw	4 KB	0x40088000	0x40088FFF	□	□	□	configurable as I2C, I2S, SPI, or USAR

Layers: set of pre-configured software components

GitHub - MDK-Packs/CB_Lab4Layer: CMSIS-Build Lab with Layers



Lab4Layer shows the following concepts:

Examples that are composed from different layers.

Readme.md files that explain the examples: generated from *.md file snippets

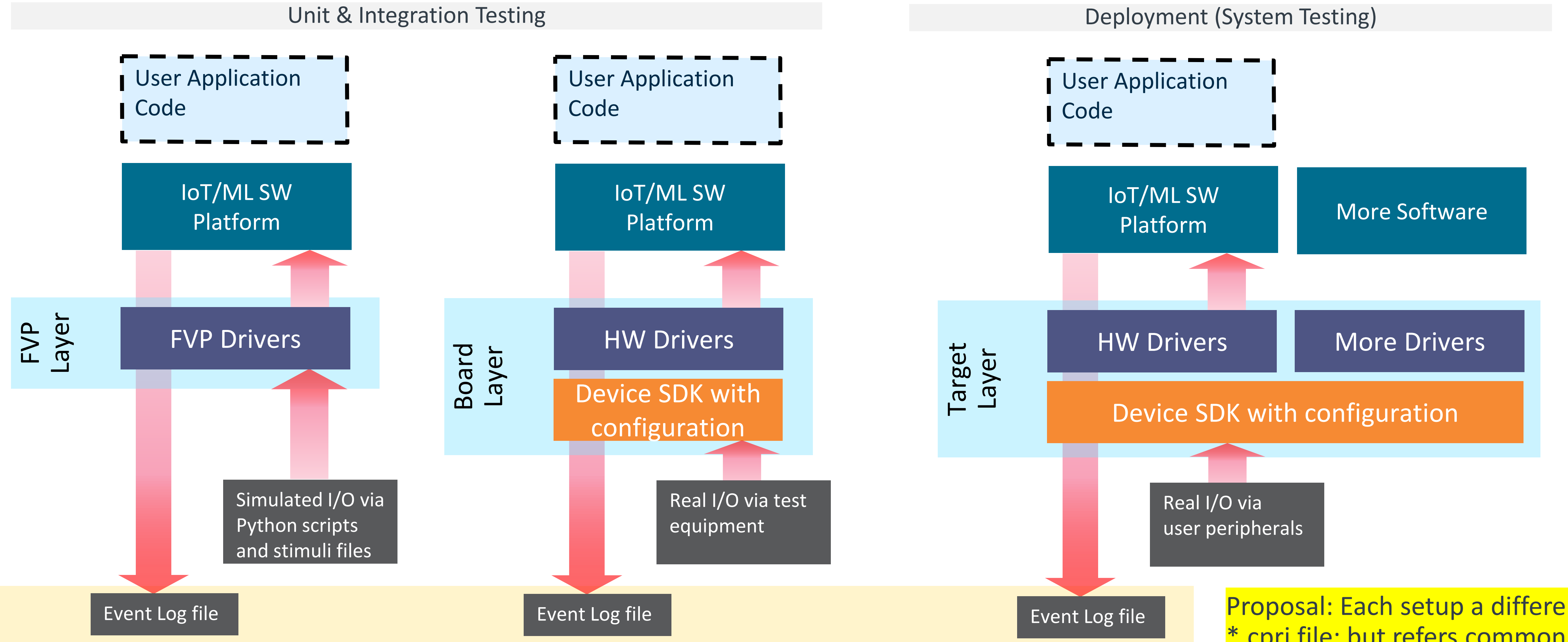
Interfaces that are:

- provided by a layer
- consumed by a layer

This allows to auto-generate a list of possible combinations.

Layers: deployment to different targets for test automation

CI/CD environment for test automation – scale from Simulation to Hardware to Deployment



Unit & Integration Testing

Deployment (System Testing)

User Application Code

IoT/ML SW Platform

FVP Layer

FVP Drivers

Simulated I/O via Python scripts and stimuli files

Event Log file

User Application Code

IoT/ML SW Platform

Board Layer

HW Drivers

Device SDK with configuration

Real I/O via test equipment

Event Log file

User Application Code

IoT/ML SW Platform

More Software

Target Layer

HW Drivers

More Drivers

Device SDK with configuration

Real I/O via user peripherals

Event Log file

Essential the same event logs are generated across the different deployments. This ensures correctness.

Initially: *Keil Studio could focus on Ease-of-Use for this process and Event Log Analysis*

Proposal: Each setup a different *.cprj file; but refers common *.clayer files.

This is a proposal

File	points to…	Description
*.pack	*.rzone *.svd	Device Family Pack: describes device, device variants, processor cores, memory, and debug options.
*.rzone		Resource file: describes device, processor cores, memory, and peripherals. Duplicates to some extend *.pack and *.svd, but does not cover device variants. Content can be generated to some extend using *.pack and *.svd, but will be incomplete (and in may cases incorrect). *.rzone files describe also sub-systems (after an assignment). Perhaps the *.rzone file can replace some content in the *.pack file or the *.pack/*.svd file can be extended to allow full generation of the *.rzone master file.
*.azone	*.rzone	Describes assignments to project zones (that can be viewed as projects) or execution zones (to setup MPU within a project).
*.cprj	(*.clayer proposed)	Describes a project and repeats device information. Linker setup could be potentially achieved using *.azone files.
*.clayer		Describes a layer with pre-configured software components and additional source files.
*.flm		Flash algorithm files; contain information about flash sector and block sizes. Somewhat duplicated in *.rzone alignment information.

New proposed files:

File	points to…	Description
*.ctarget	*.azone *.cprj (multiple) *.cdebug	Describes the projects that compose a target application. Potentially this could be merged with the *.azone file.
*.cdebug		Debug configuration for the target. Potentially a *.ctarget could have multiple debug configurations, i.e. for CMSIS-DAP, ULINK, J-Link, DSTREAM, etc.

Note: for stand-along tools like CMSIS-Build or a command-line debug tool (DSE) it should be sufficient to use a *.cprj or *.cdebug file to configure the tool. It is therefore OK to somewhat duplicate information in the various files.

Important is a consistency check of the various information across the files.

Close gaps on *.cprj

- Enable GCC and AC6 based compilation with same *.cprj file
- Pre-build/post-build steps, ideally consistent across Linux/Win/MacOS
- Integration of generators (review *.gpdsc concept)
  - Consider both: Cloud and Desktop flows

- Different build configuration (release, debug, test) with a default setting
- Stand-alone build (allow to integrate *.clayer)
  - Should a *.cprj that refers external *.clayer files have a different extension, i.e. *.cprjx?

IMPORTANT NOTE: CMSIS-Build and the underlying project format(s) should support Command-Line and IDE flows (we use it in Keil Studio as the primer format)

CMSIS-Pack index file processing (vidx2pidx)

- Charles Oliveira works for Linaro
- First Open-CMSIS-Pack CLI tool development
- https://github.com/Open-CMSIS-Pack/vidx2pidx
- Using GO language and GitHub Actions
- DEMO
- Q&A

Way of working …

- Definition of done?
  - A User Story is done, when an ADR is accepted?
    - User stories have acceptance criteria giving the DoD
    - Acceptance criteria typically involve the user being able to replay the story; requiring some sort of implementation
  - An ADR is done when an implementation is available?
    - ADR is recording a design decision
    - ADRs can be proposed and accepted
    - ADRs are long-running and cannot be "done"
    - ADRs might be violated
    - ADRs can be revoked or superseded by new ones

Way of working …

- User Stories:
    - GitHub issues – Propose (1) -> In Review (2) -> Accepted (3) -> JIRA issue (4) -> Prioritize backlog
    - Sprint planning -> Break down -> Development
- Contribute via “Pull requests”
    - Need to define criteria for accepting contributions (Volunteers?)
    - Need repository gate keeper

Wrap Up

Next week:

- Assisted CMSIS-Pack generation of CMake-based projects (CMSIS-13)

July:

- Protecting CMSIS-Pack from malicious tempering (TBD)

August:

- Kick-off development for project creation and maintenance MVP CMSIS-12

- Actions:

- Next Meeting: Tuesday July 6th 2021, 15:00 – 16:00 (UK)

Thank you