



CMSIS Partner Meeting

Embedded World 2018

Reinhard Keil
Sr. Director Embedded Tools

Agenda

Welcome & CMSIS Overview, Status, Plans

CMSIS-Pack / CMSIS-Driver enhancements

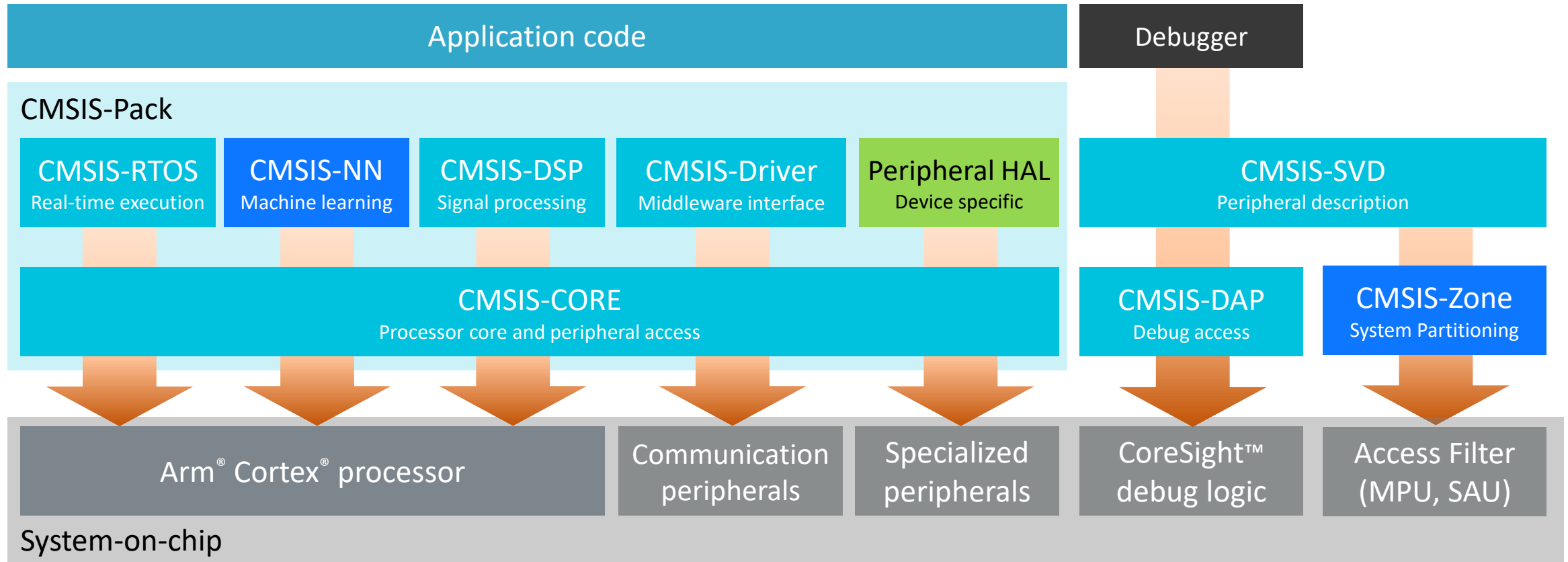
CMSIS-Zone: management and partitioning of complex systems

Summary and discussion

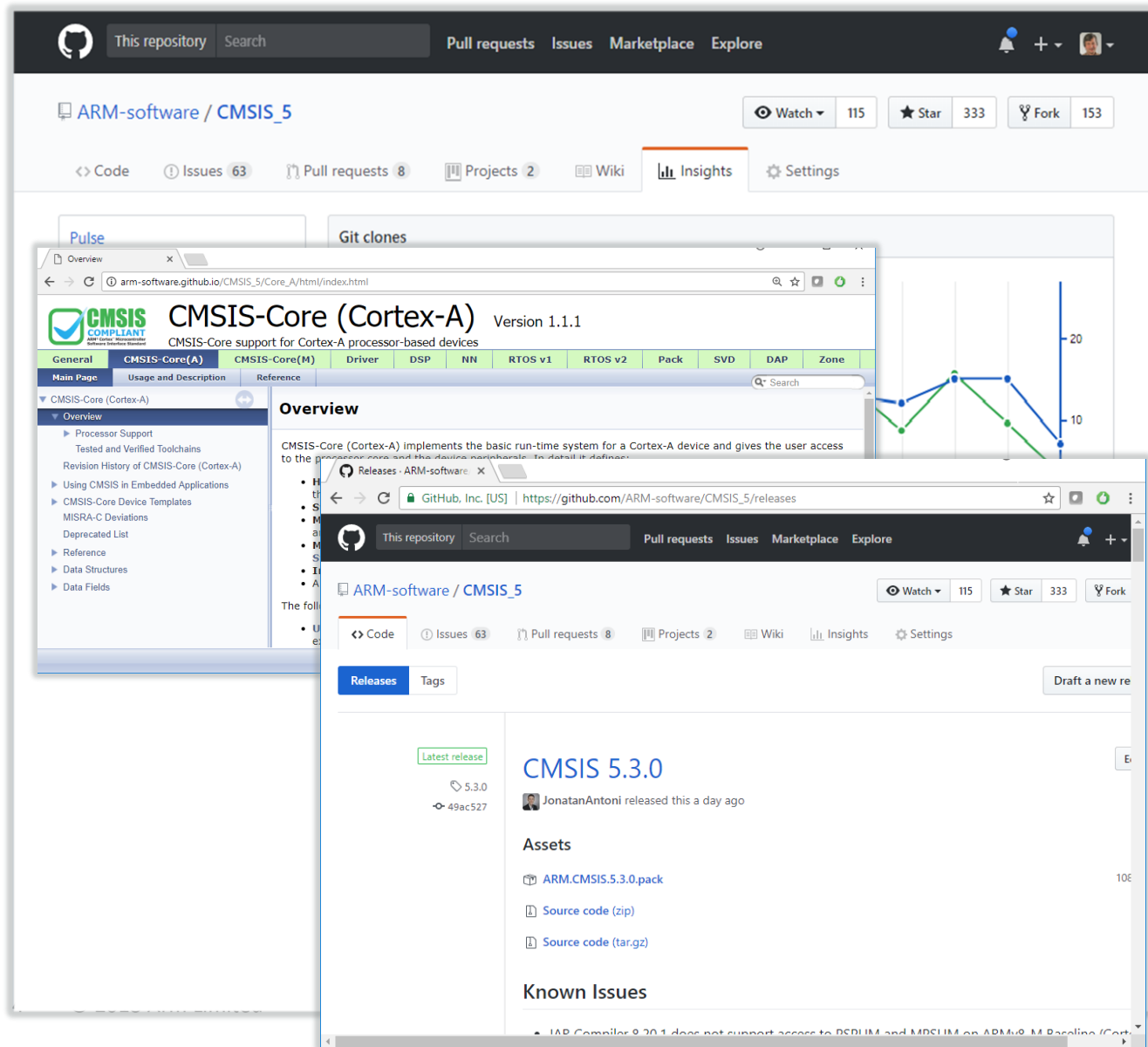
CMSIS 5



Consistent software framework for Arm Cortex-M and Cortex-A5/A7/A9 based systems



CMSIS development on github.com/ARM-software/CMSIS_5



Gives partners full visibility to our development activities

Good contribution from community

Feedback channel for partners and customers

Access to documentation:

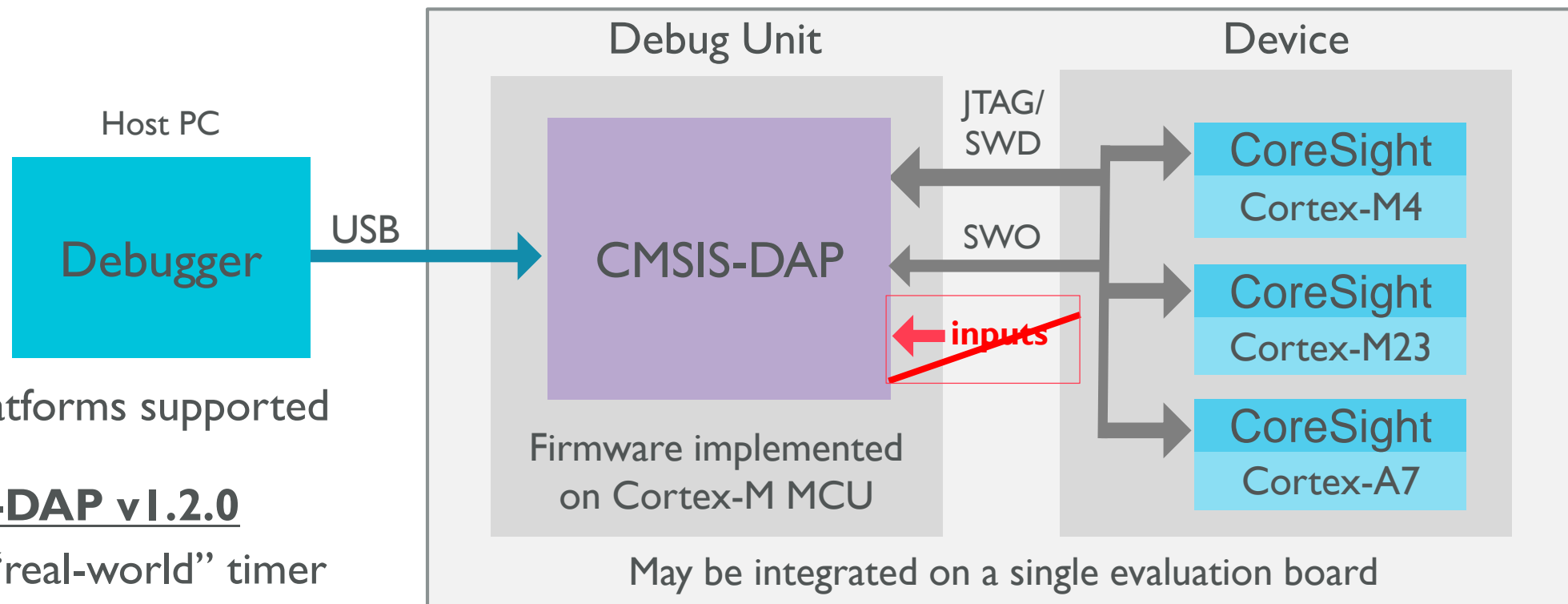
arm-software.github.io/CMSIS_5/General/html/index.html

Access to release information:

github.com/ARM-software/CMSIS_5/releases

Thank you for all your work!

CMSIS-DAP: Overview + Enhancements (v1.2.0)

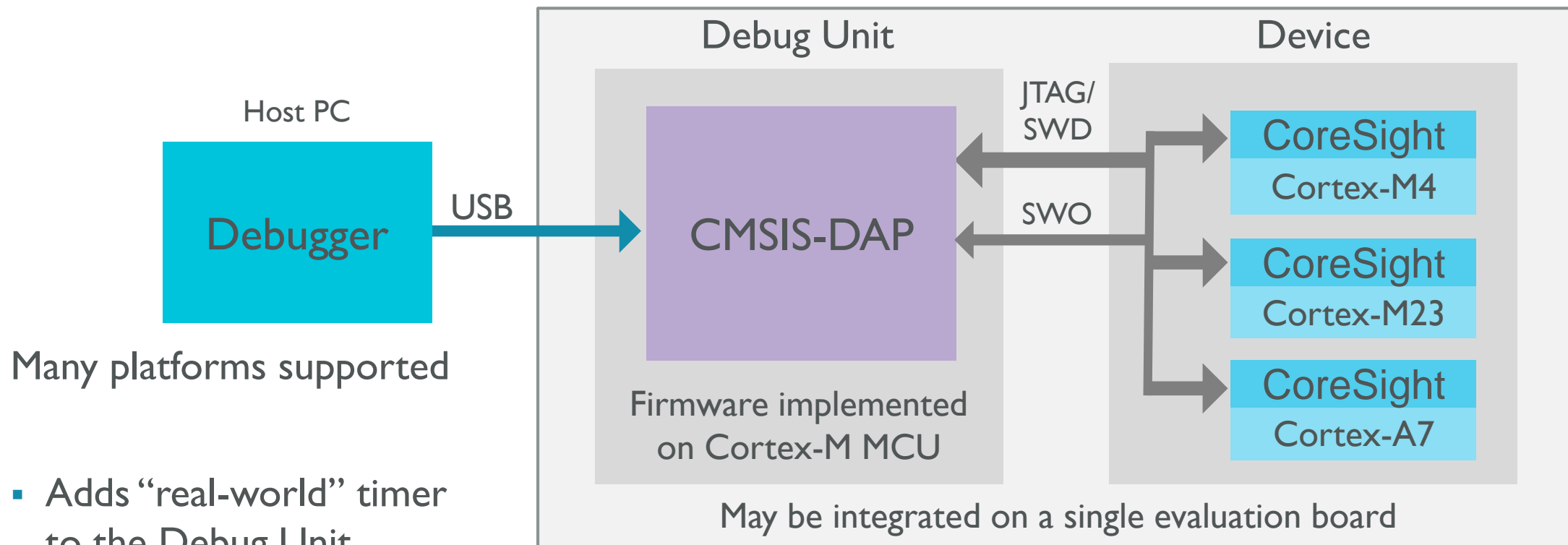


Many platforms supported

CMSIS-DAP v1.2.0

- Adds “real-world” timer to the Debug Unit
- Introduces trace recording for custom Performance Counters (**inputs**), for example:
 - Power measurement (U, I) from A/D converters
 - Performance parameters from a external system (i.e. wait states)
 - Transfer parameters of an RF interface

CMSIS-DAP: Released is now v1.2.0 + v2.0.0



Many platforms supported

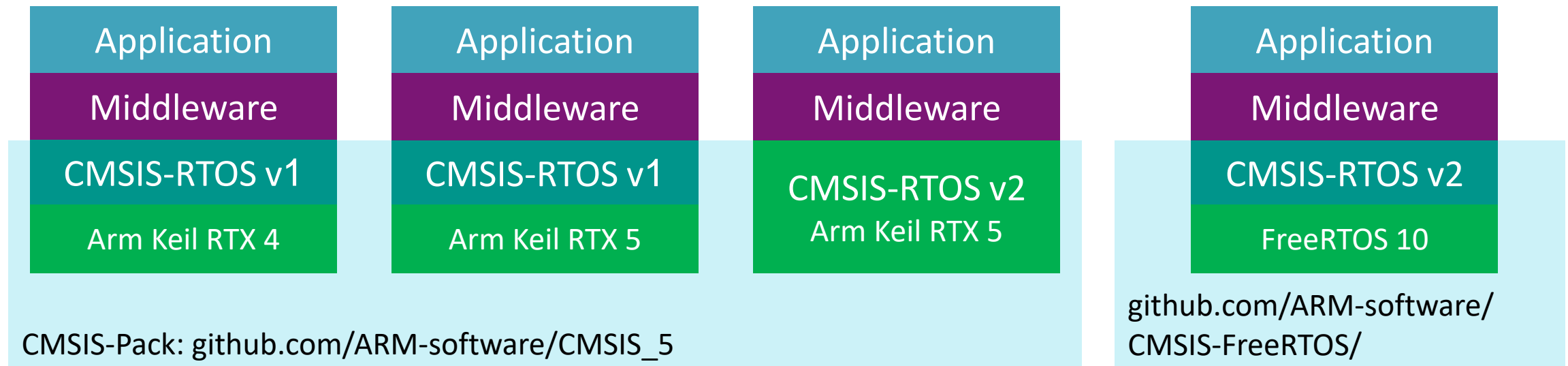
- Adds “real-world” timer to the Debug Unit
- CMSIS v1.2.0: continues to support USB HID as interface
- CMSIS v2.0.0: introduces USB WIN support with >5x better performance
 - SWO streaming via separate pipe allows significant better trace bandwidth
 - Windows 10 does not require USB driver installation

CMSIS-RTOS Implementations

Independent RTOS implementations for usage with application code and middleware

All implementations are available under permissive license, are compatible with various compilation toolchains and support CMSIS-RTOS v1 (legacy)

RTX5 and FreeRTOS support CMSIS-RTOS v2 and CMSIS-RTOS v1 via legacy layer



Software building blocks for functional safety

Real-time operating system and FuSa C library for Arm Cortex-M

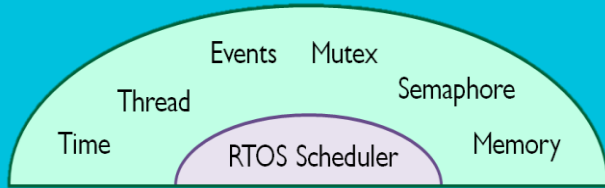


CMSIS allows commercial products

Application code

Arm Keil RTX5

Versatile, lightweight RTOS



Software test library (STL)

Runtime processor system verification

Arm FuSa C library

Performance-optimized commonly used functions

Arm C/C++ Compiler

Efficient, architecturally-accurate code generation

Arm Cortex-M processor

World's most popular 32-bit microcontroller architecture

Ready-to-use software framework:

- Fully qualified by TÜV for projects that require ISO 26262 (ASIL-D), IEC 61508 (SIL3), IEC 62304, EN 50128
- Developers can concentrate on their application code certification
- Faster time-to-market
- Optimized by the architecture experts
- One-stop shop for all software components

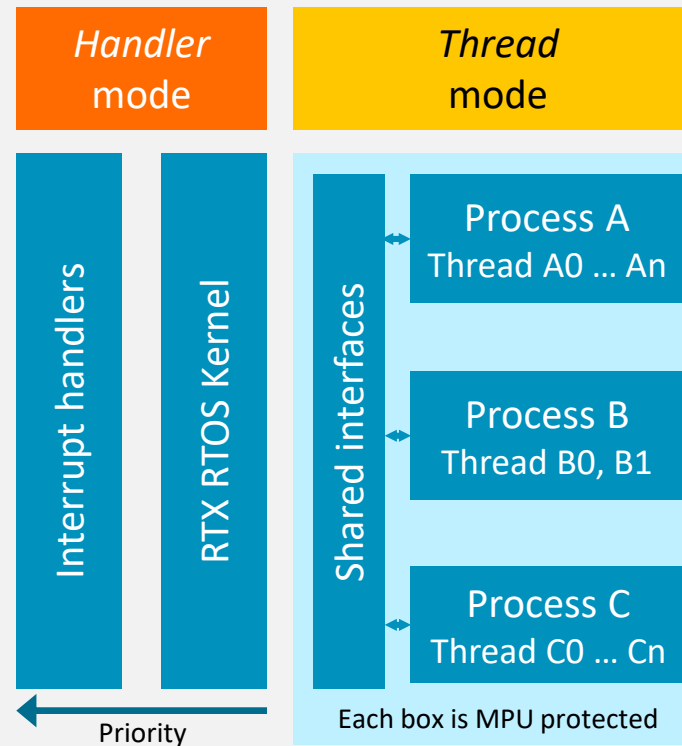
RTX – RTOS with optional process isolation



CMSIS-Zone System Partitioning

Name	Access	Size	Process A	Process B
STM32F407IG				
memory				
IRAM1	rwX	128 KB	0x20000000	0x20000000
RamA	rwX	64 KB	0x20000000	0x20000000
RamB	rwX	64 KB	0x20010000	0x20010000
IRAM2	rwX	64 KB	0x10000000	0x10000000
App	rx	256 KB	0x8010000	0x8010000
Storage	r	128 KB	0x8090000	0x8090000
Scratch	r	256 KB	0x8050000	0x8050000
peripherals				
ADC1	prw	1 KB	0x40012000	0x40012000
ADC2	prw	1 KB	0x40012100	0x40012100
ADC3	prw	1 KB	0x40012200	0x40012200
C_ADC	prw	1 KB	0x40012300	0x40012300
GPIOA	prw	1 KB	0x40020000	0x40020000
GPIOB	prw	1 KB	0x40020400	0x40020400
GPIOC	prw	1 KB	0x40020800	0x40020800
GPIOD	prw	1 KB	0x40020C00	0x40020C00
GPIOE	prw	1 KB	0x40021000	0x40021000
GPIOF	prw	1 KB	0x40021400	0x40021400

RTX5 RTOS Safe Process Isolation



RTX optional uses the Protection Unit (MPU) of Cortex-M processors

The MPU isolates processes and protects from incorrect accesses to data and peripherals

CMSIS-Zone simplifies the setup of MPU protected execution zones



arm

Applied Machine Learning (ML)
on Low-energy Platforms

Optimum ML performance on Arm for any application

Arm NN software translates existing NN frameworks:

- TensorFlow, Caffe, Android NNAPI, MXNet etc
- Developers maintain existing workflow and tools
- Reduces overall development time
- Abstracts away the complexities of underlying hardware

CMSIS-NN 5x better efficiency and performance for NN functions

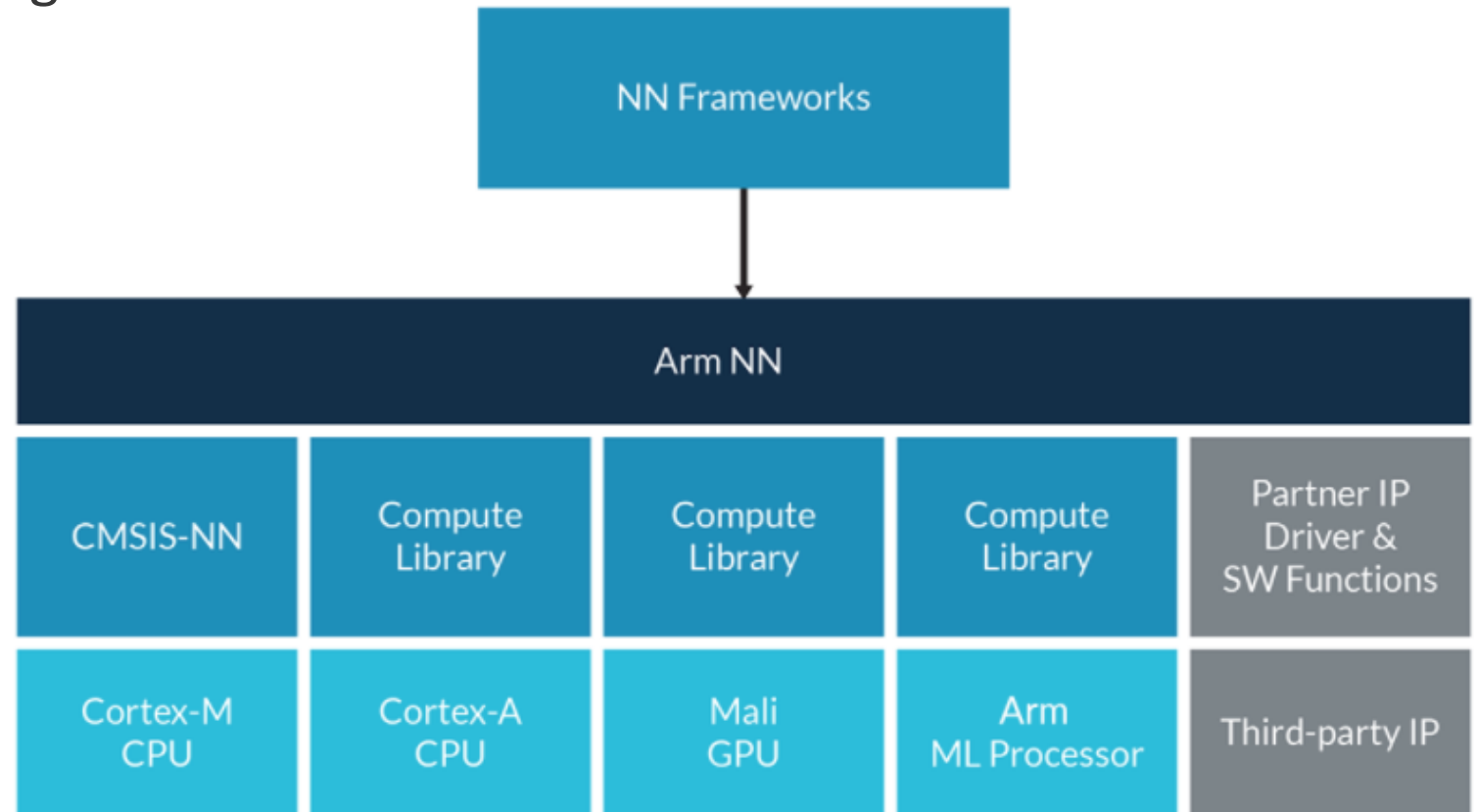
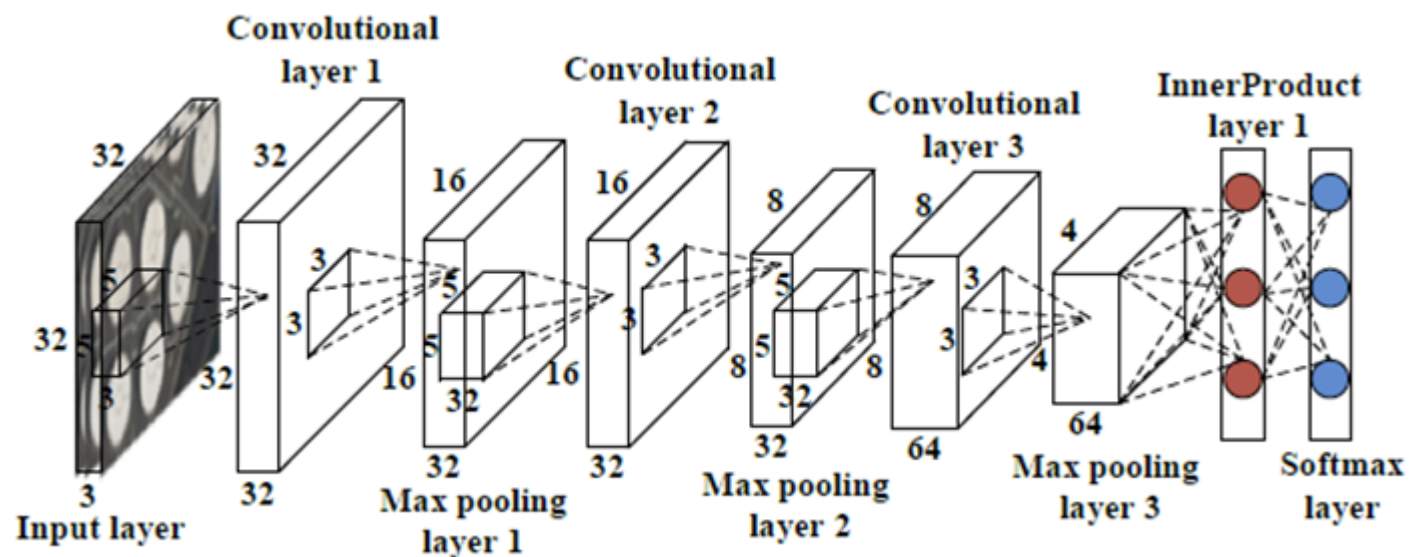


Image classification - Convolutional neural network

- CIFAR-10 classification – classify images into 10 different object classes
- 3 convolution layer, 3 pooling layer and 1 fully-connected layer (~80% accuracy)



airplane

automobile

bird

cat

deer

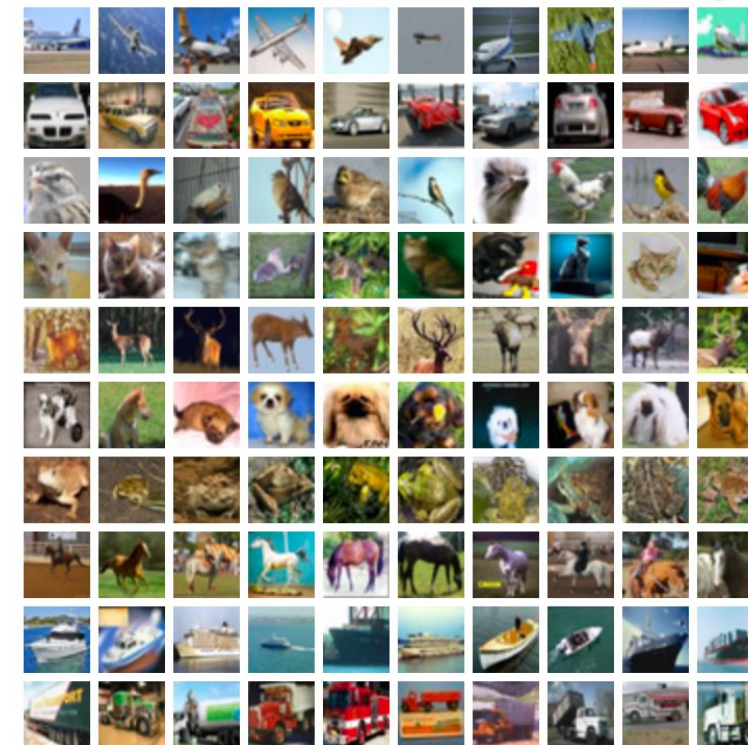
dog

frog

horse

ship

truck





CMSIS-Pack and CMSIS Driver

Joachim Krech
Director of Engineering, CMSIS & MCU Tools

CMSIS-Pack

Software Pack Use Cases

Revision History of CMSIS-Pack

- ▶ Software Packs Overview
- ▶ Pack with Software Components
- ▶ Pack with Device Support
- ▶ Pack with Board Support

Tutorials

Pack Example

▶ Utilities for Creating Packs

▶ Publish a Pack

- ▶ Pack Description (*.PDSC) Format
- ▶ Configuration Wizard Annotations
- ▶ Flash Programming
- ▶ Debug Setup with CMSIS-Pack
- ▶ Project Description (*.CPDSC) Format
- ▶ System Description File (*.SDF) Format
- ▶ CMSIS-Pack Index Files

CMSIS-Pack Documentation

CMSIS-Pack describes a delivery mechanism for software components, device parameters, and evaluation board support. The XML-based package description (PDSC) file describes the content of a **Software Pack** (file collection) that includes:

- Source code, header files, and software libraries
- Documentation and source code templates
- Device parameters along with startup code and programming algorithms
- Example projects

Software Pack Use Cases



Software Pack Use Cases

**28 vendors
published
5339 devices**

One pack, many IDE's: IAR, MDK, DS-MDK, Eclipse-Pack ...

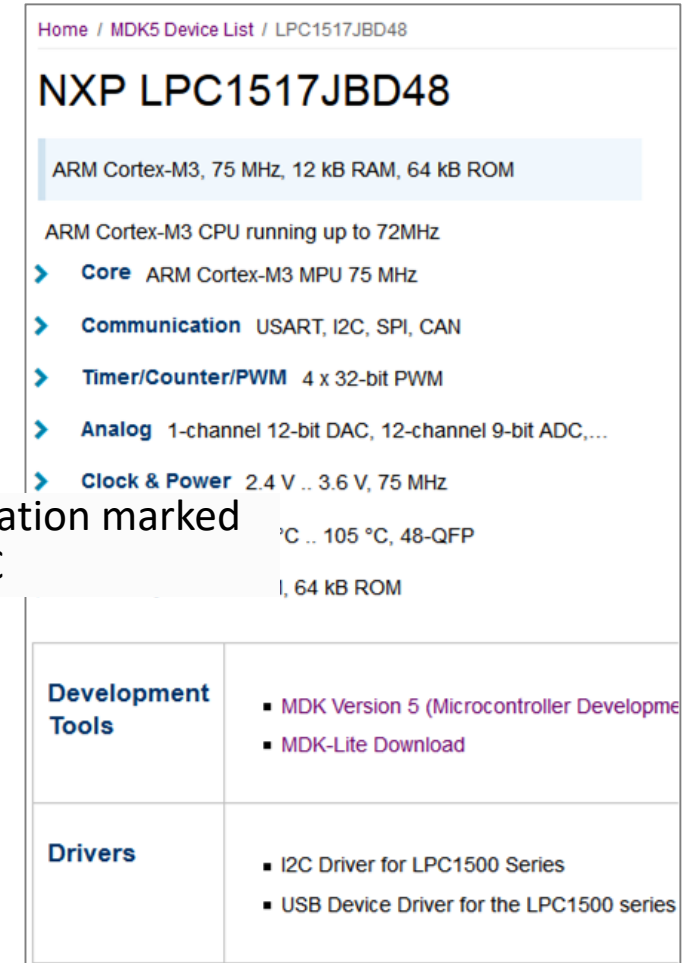
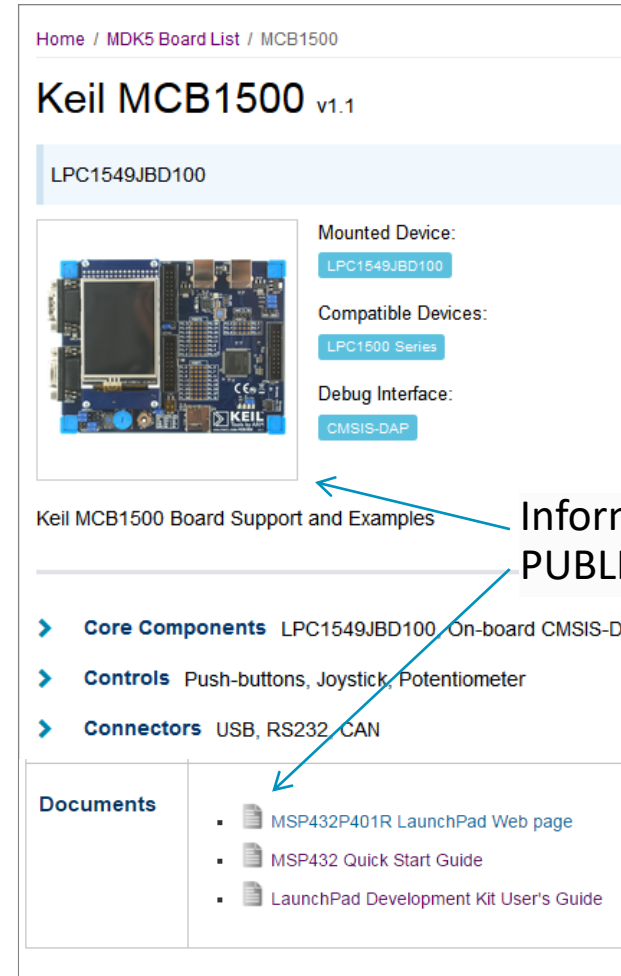
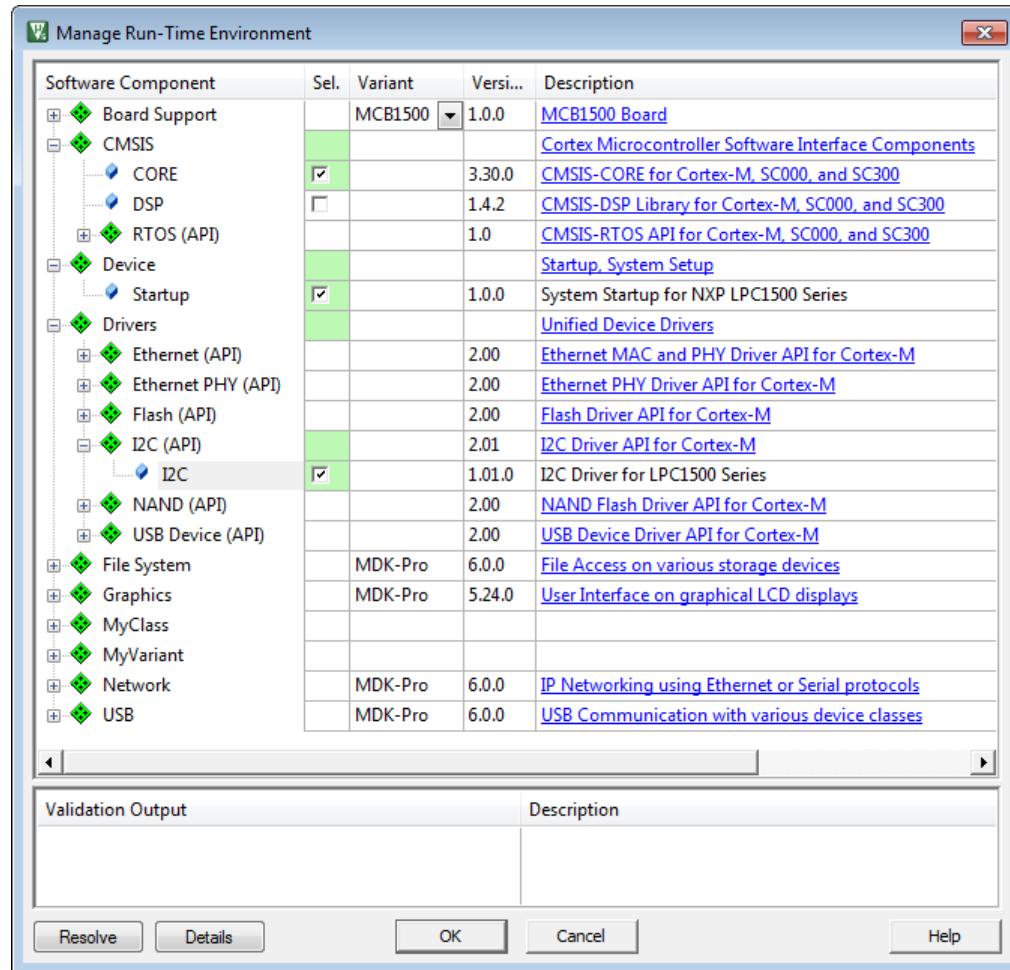
CMSIS-Pack: Device Family Pack - Features

CMSIS-Pack allows to deliver the following information about a device:

- Information about the processor and its features.
- C and assembly files for the device startup and access to the memory mapped peripheral registers.
- Parameters, technical information, and data sheets about the device family and the specific devices
- Device description and available peripherals
- Memory layout of internal and external RAM and ROM address ranges
- Flash algorithms for programming the device
- **Debug Description** that specifies debug and trace configurations – and in future Flash download
- System View Description (CMSIS-SVD) files for memory mapped peripheral registers

CMSIS-Pack is Designed for Tools and Web Portals

Information in Packs is Shown in Tools and on Web Pages



CMSIS-Pack – what is coming in 2018

Extension to: complex systems, security, and generic project templates

Config Wizard extension: access enum's for configuration information – already implemented

System Description SDF Format: describe more complex debug topologies than with a Debug Description in a tool agnostic way

Github based workflow: allows to develop software packs using github infra-structure

Flash algorithm via debugger: Some TrustZone enable devices cannot execute RAM. Commands that allow flash programming will be added to Debug Description.

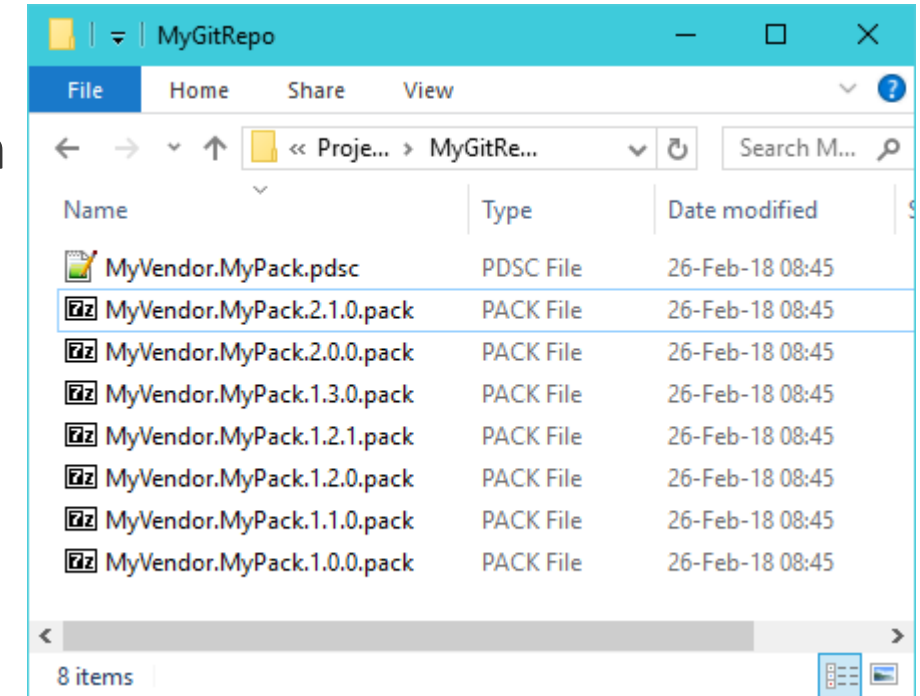
CPDSC project file format: allows project templates that are agnostic of an IDE

Minimize need for IDE specific settings: CMSIS-Pack supports IDE specific parameters. Analyze and minimize

Current public repository work-flow

Public repositories and CMSIS-Packs:

- Requires a dedicated repository or repository branch
 - latest version of the package description file (*.pdsc)
 - **all** downloadable pack versions (*.pack)
- Require a vendor index file referencing the pdsc file
 - MyVendor.pidx:



```
<pdsc url="https://github.com/MyGitRepo/" vendor=" " name=" " version=" " />
```

Adding flexibility to use github during development (Proposal)

Filename and relative path to pack per release:

```
<release version="2.1.0" filename="archive/2.1.0.zip">
```

Location and type of repository:

```
<repo type="git">https://github.com/arm-software/CMSIS-Driver</repo>
```

Tag of the release:

```
<release version="2.1.0" tag="2.1.0">
```

Pack location specified relative to <repo> or <url> (pdsc file)

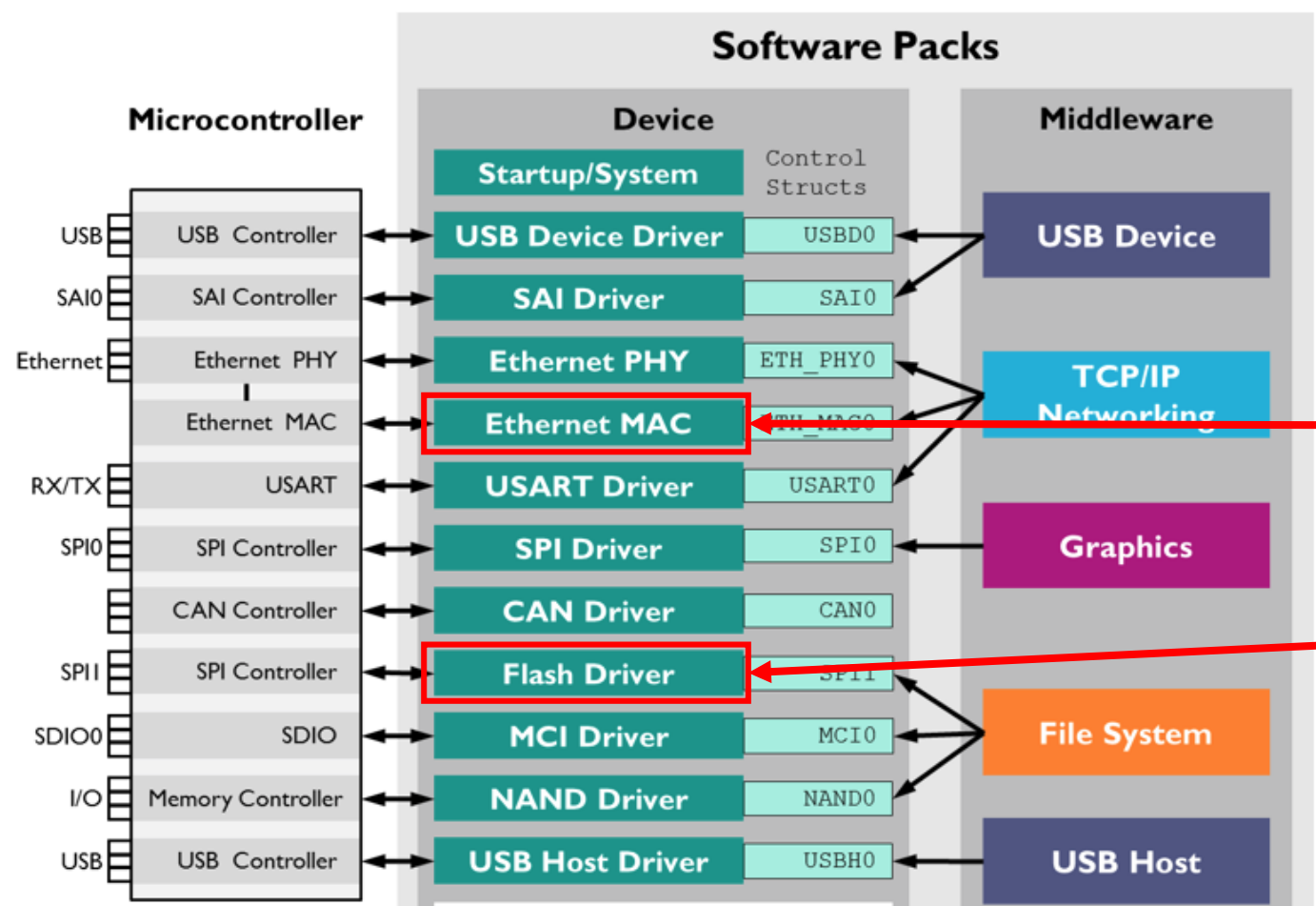
```
<release version="2.1.0" repo="true" filename="archive/2.1.0.zip">
```



- ▼ CMSIS-Driver
 - Overview
 - Revision History of CMSIS-Driver
 - ▶ Theory of Operation
 - ▶ Reference Implementation
 - ▶ Driver Validation
 - ▶ Reference
 - ▶ Data Structures
 - Data Structure Index
 - ▶ Data Fields

Overview

The CMSIS-Driver specification is a software API that describes peripheral driver interfaces for middleware stacks and user applications. The CMSIS-Driver API is designed to be generic and independent of a specific RTOS making it reusable across a wide range of supported microcontroller devices. The CMSIS-Driver API covers a wide range of use cases for the supported peripheral types, but can not take every potential use-case into account. Over time, it is indented to extend the CMSIS-Driver API with further groups to cover new use-cases.



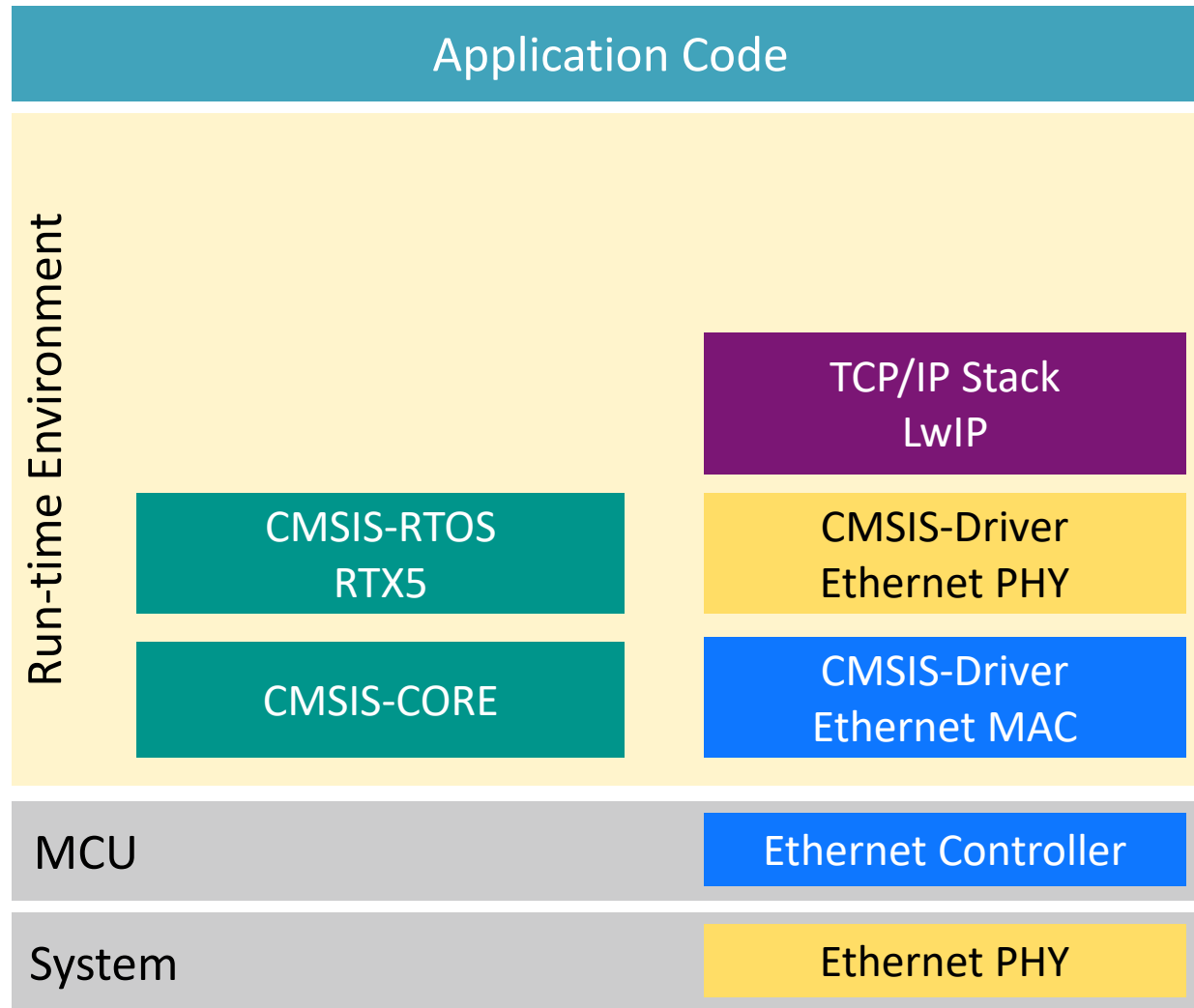
CMSIS-Driver Pack Generic Drivers

that are device independent are now available as separate software pack.

[github.com/
ARM-software/
CMSIS-Driver](https://github.com/ARM-software/CMSIS-Driver)



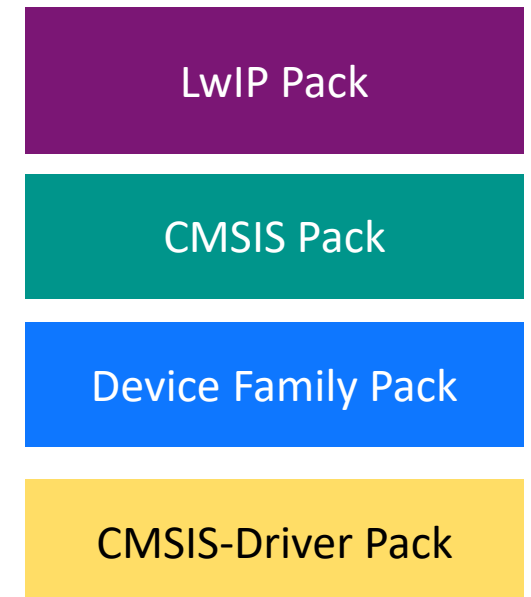
CMSIS-Driver use case example



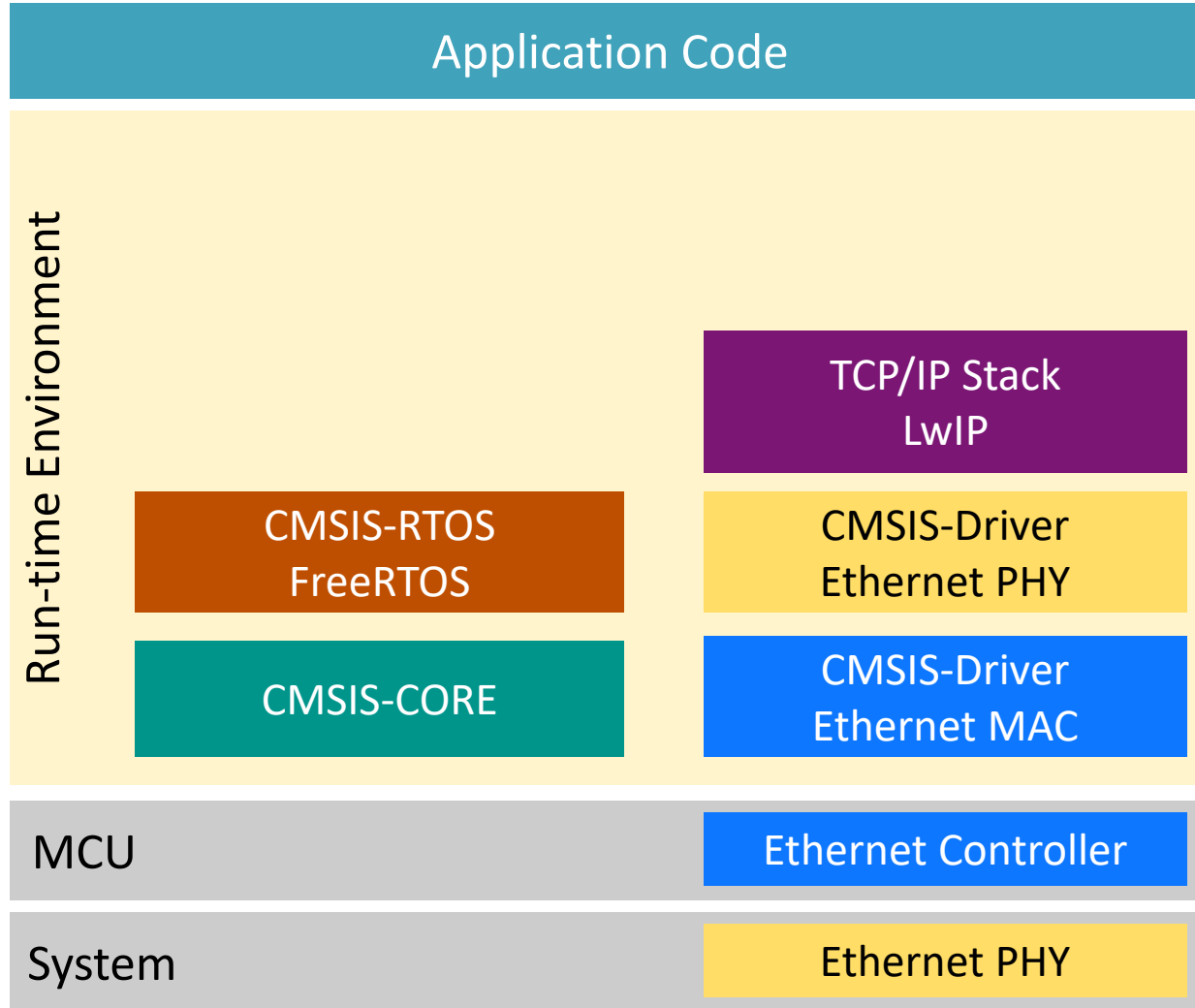
Each software pack simplifies the update of related software components

Generic Driver Pack gives access to several Ethernet PHY drivers:

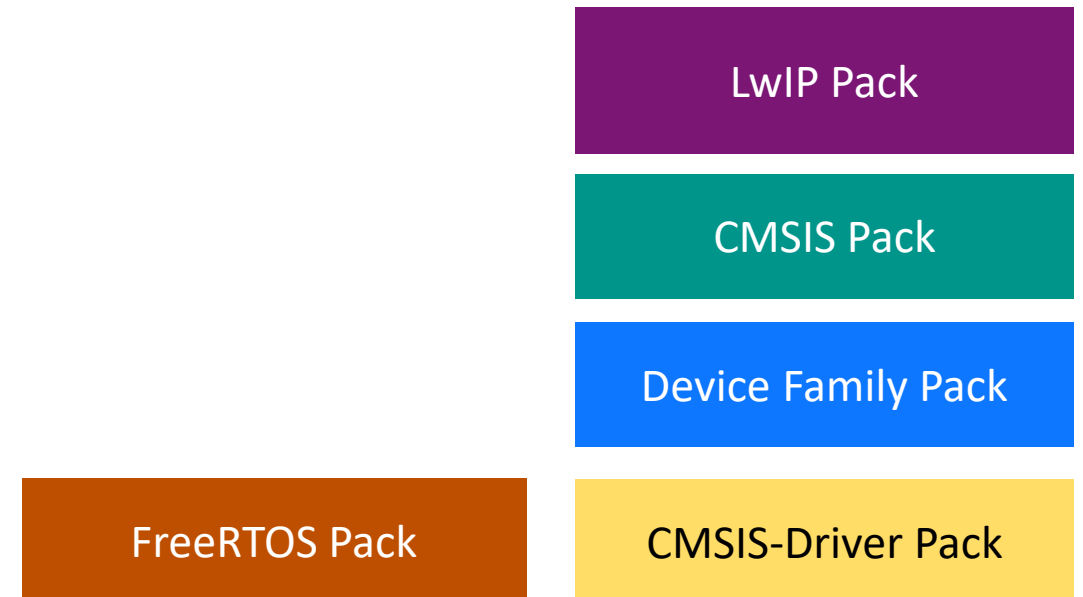
- DP83848
- KSZ8061RNB
- KSZ8081RNA
- LAN8710A
- LAN8720
- LAN8742A
- ST802RT1



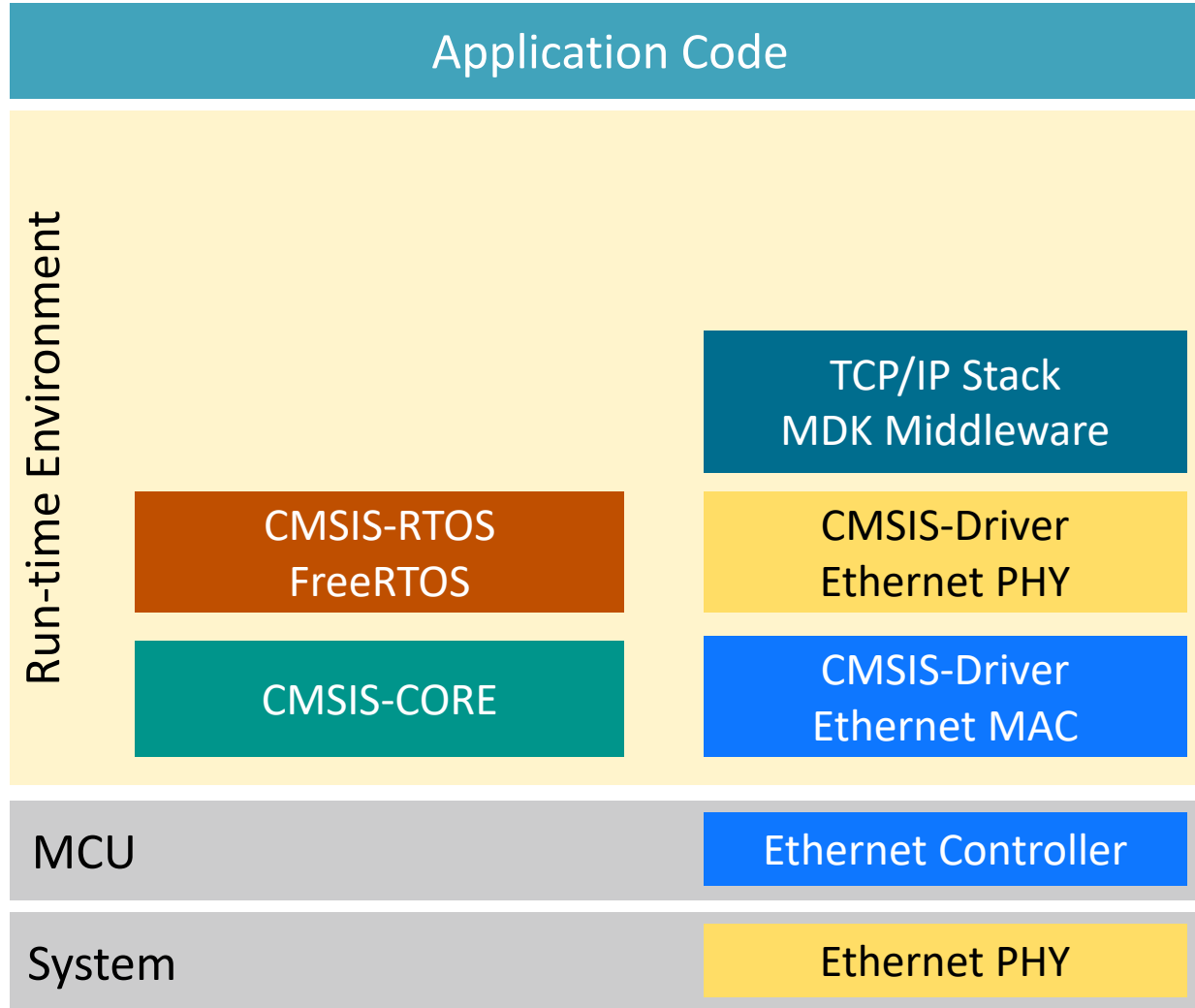
CMSIS-Driver use case example



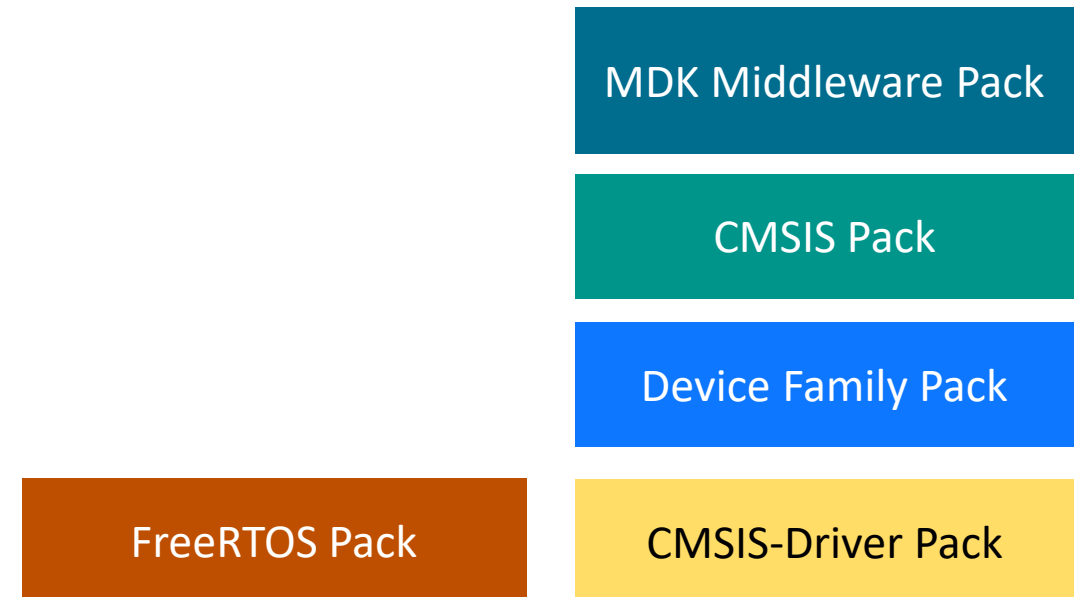
Software components can be exchanged easily



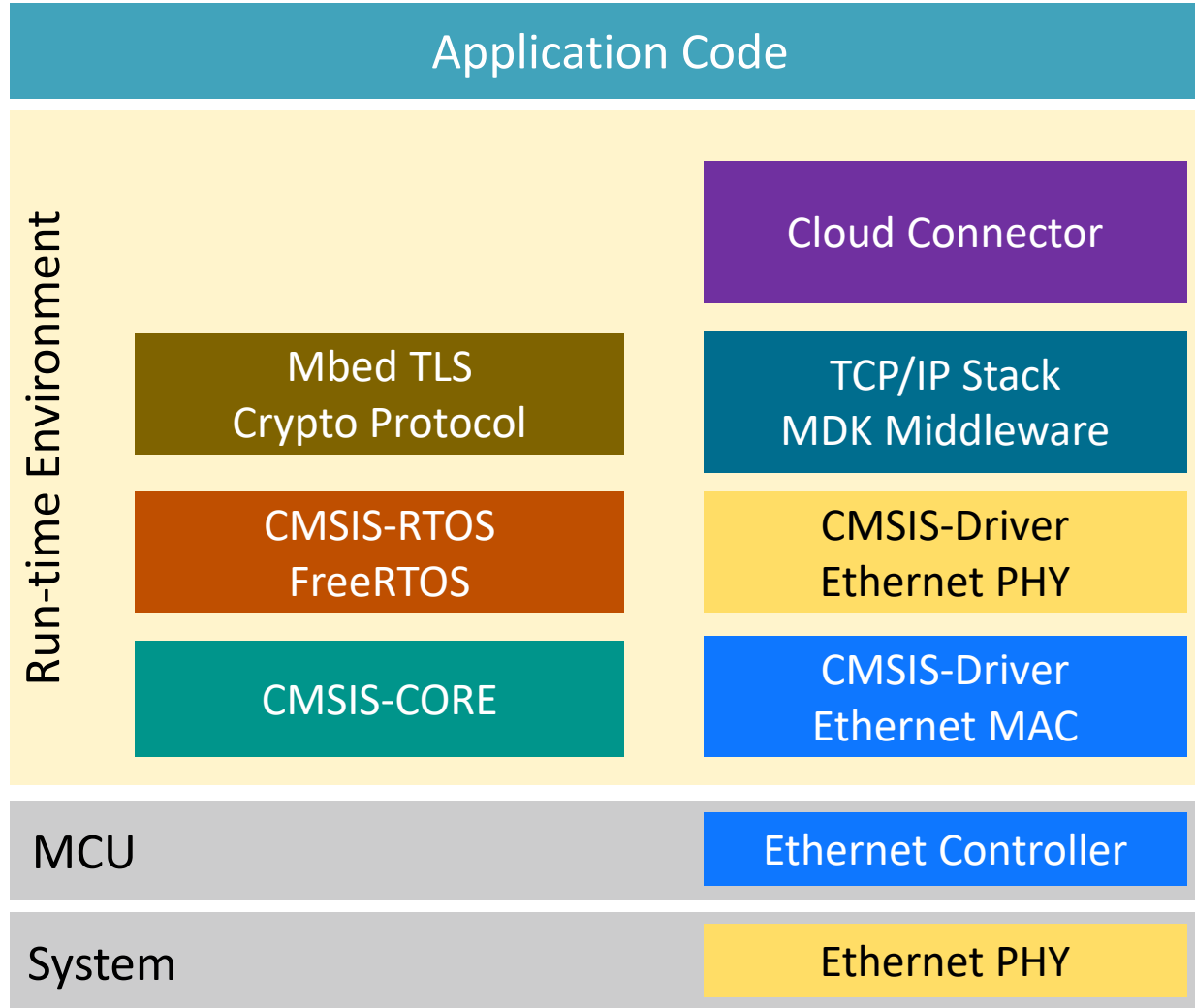
CMSIS-Driver use case example



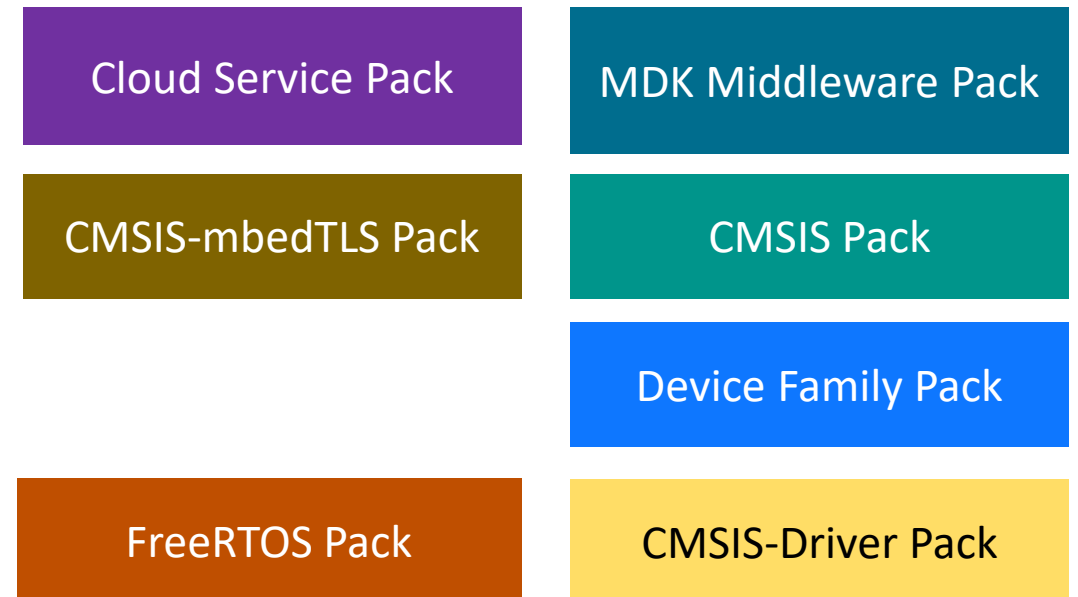
Software components can be exchanged easily



CMSIS-Driver use case example



Functionality can be added



CMSIS-Driver usage example – step-by-step

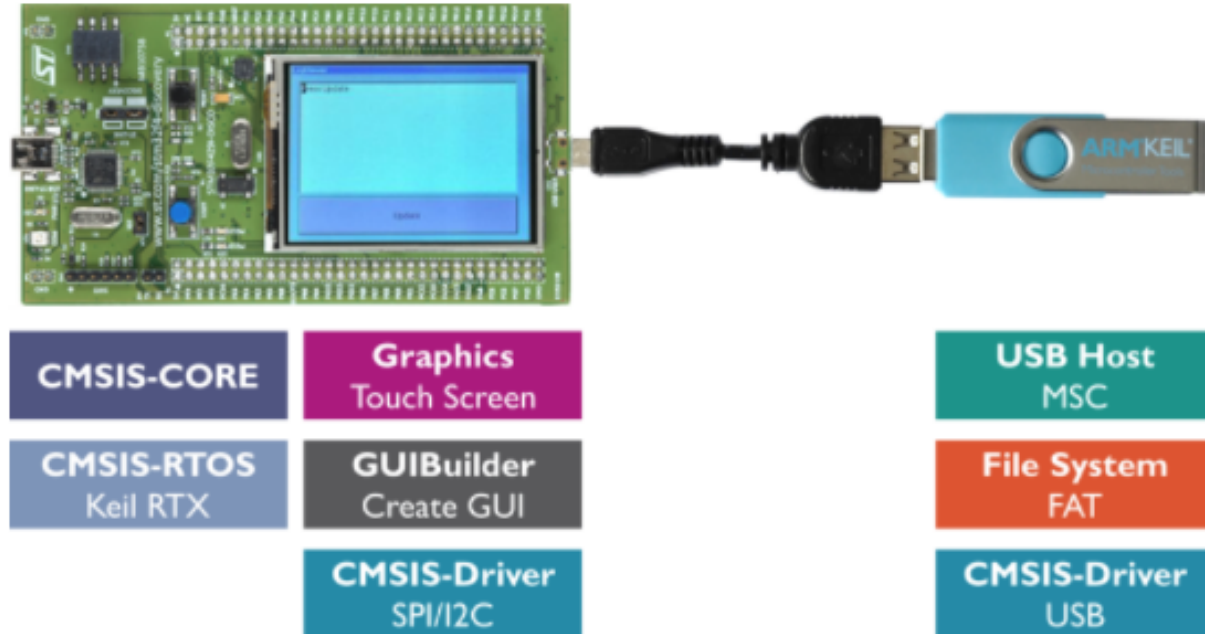
USB Host Application with File System and Graphical User Interface

This tutorial shows you step-by-step how to create a complete embedded application. The application uses middleware components to read the contents of a text file from an USB memory stick attached to a development board. After pressing the update button this content is shown on the LCD. The following MDK-Professional middleware components are incorporated:

- USB Host supporting Mass Storage Class (MSC)
- File System for FAT devices
- Graphics for displaying a graphical user interface (GUI) using a touch screen

Various **CMSIS-Driver** connect the application code and the middleware to the device peripherals:

- USB for attaching the memory stick
- SPI for connecting the TFT LCD
- I2C for controlling the touch interface of the display



 Download

The application note explains the required steps to create the USB project on a [STM32F429I-Discovery kit](http://www.keil.com/learn/usb_host/). It contains in-depth instructions and all the required code snippets.

www.keil.com/mdk5/learn/usb_host/

Easy to get middleware up and running for devices

Redpine Signals RS14100_4MB

ARM Cortex-M4, 200 MHz, 3 MB ROM, 191 kB RAM

Redpine Signals RS14100 WiSeMCU family of chips and modules device is the industry's first Wireless Secure MCU family with a comprehensive multi-protocol wireless sub-system. It has an integrated ultra-low-power microcontroller, a built-in wireless subsystem, advanced security, high performance mixed-signal peripherals and integrated power-management.

Quick Links

- [Board List](#)
- [Software Packs](#)
- [MDK Version 5](#)
- [Legacy Support](#)
- [Feedback](#)

CMSIS Drivers	<ul style="list-style-type: none">▪ CAN Driver API for Cortex-M▪ Ethernet MAC API for Cortex-M▪ USB Device Driver API for Cortex-M▪ USB Host Driver API for Cortex-M▪ MCI Driver API for Cortex-M▪ I2C Driver API for Cortex-M▪ SPI (SSP) Driver API for Cortex-M▪ USART Driver API for Cortex-M▪ SAI (I2S) Driver API for Cortex-M
Examples	<ul style="list-style-type: none">▪ Blinky▪ CMSIS-RTOS Blinky▪ CAN▪ CCI_MASTER▪ COMPARATOR <div>RS14100 RS14100 RS14100 RS14100 RS14100</div>

Arm eco-system provides ready-to-use software frameworks:

- Several software components use already CMSIS-Driver interfaces
- Templates help to develop CMSIS-Drivers for new silicon devices.
- Once verified, the driver set enables connection to established middleware
- Drivers can be a part of the device family pack (DFP) which simplifies overall installation
- Some SiPs do this already for new microcontroller devices

Please get involved!



CMSIS-Zone

Jonatan Antoni
CMSIS Technical Lead

▼ CMSIS-Zone (Preview)

- ▶ CMSIS-Zone Use Cases
Revision History of CMSIS-Zone
- ▶ Zone Description Format
- ▶ Generator Data Model

CMSIS-Zone (Preview) Documentation

This is a preview of **CMSIS-Zone** which is scheduled for release in Q1'2018. The final release of CMSIS-Zone will provide:

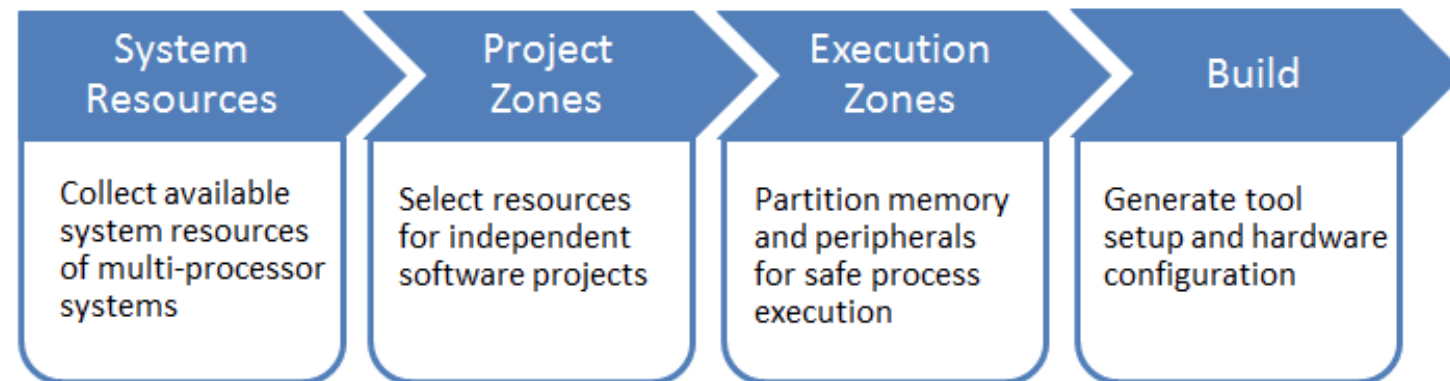
- **Zone Description Format** (XML based) which stores consistent system setup information.
- CMSIS-Zone configuration utility for system partitioning. The **data** captured can be exported to various project and configuration files using file templates.

CMSIS-Zone defines methods to describe system resources and to partition these resources into multiple projects and execution areas. The system resources may include multiple processors, memory areas, and peripherals. The system resource and partitioning information is stored in **Zone Description Format** (XML based).

CMSIS-Zone includes an interactive tool that manages files in the **Zone Description Format** which allows to:

- create system resource information from existing CMSIS-SVD and CMSIS-Pack descriptions.
- partition system resources into various project zones.
- partition a project zone into multiple execution zones.
- generate various configuration files for tool set-up and hardware initialization.

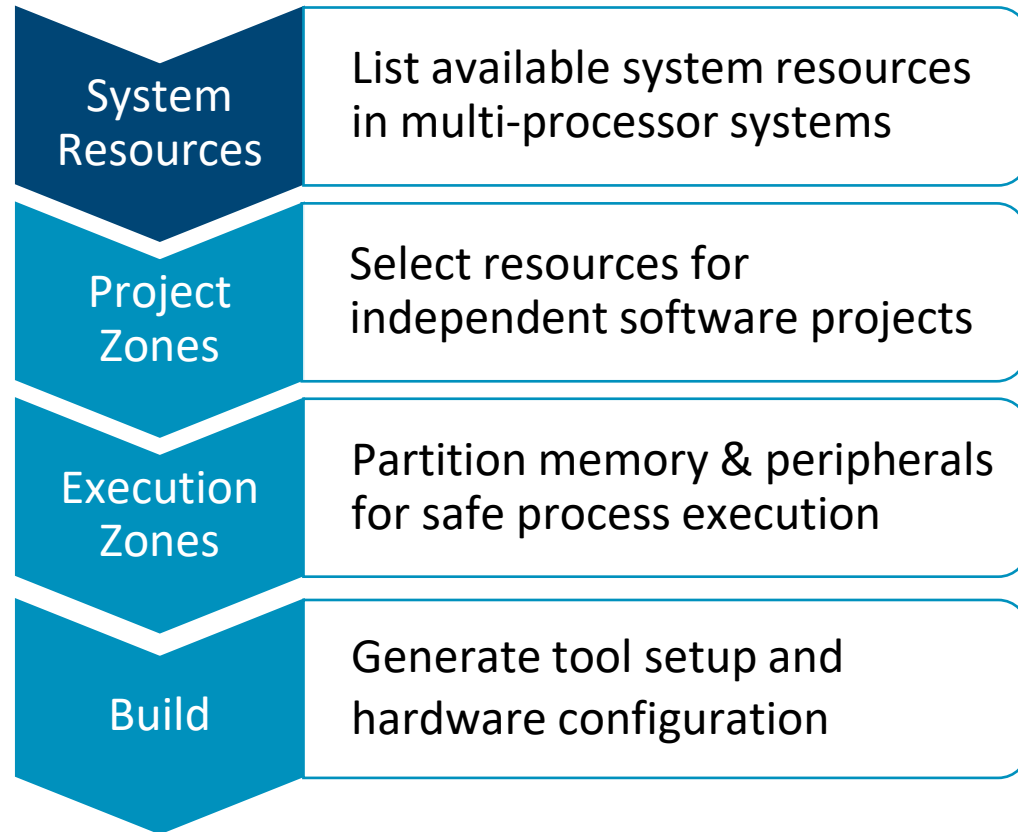
The following diagram explains the development flow when using the **CMSIS-Zone** management tool.



CMSIS-Zone Development Flow

CMSIS-Zone – Development Workflow

Resource configuration for multi-processor systems and execution regions



MCBSTM32F400.szone

System Map [Icons] Generate code

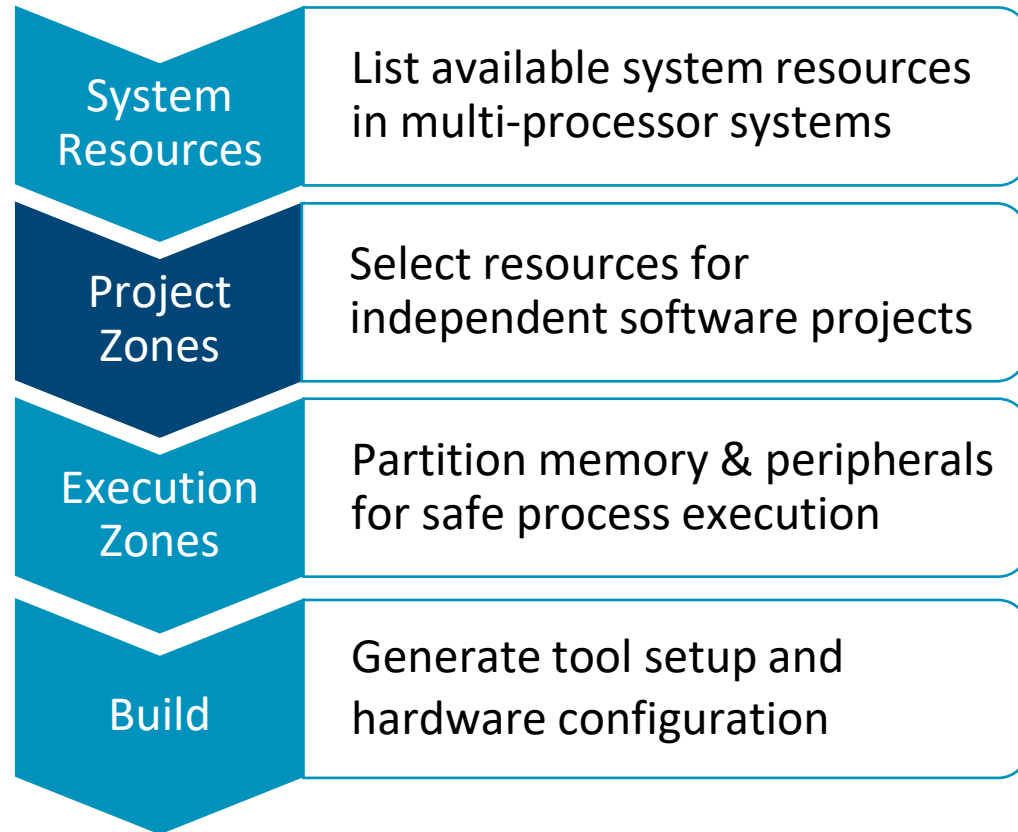
Name	Access	Size	Cortex-M4	Description
memory				
IRAM1	rwX	128 KB	0x20000000	
IRAM2	rwX	64 KB	0x10000000	
IROM1	rx	1 MB	0x08000000	
peripherals				
ADC				
CAN				
CRC	prw	1 KB	0x40023000	Cryptographic processor
DAC	prw	1 KB	0x40007400	Digital-to-analog converter
DBG	prw	1 KB	0xE0042000	Debug support
DCMI	prw	1 KB	0x50050000	Digital camera interface
DMA				
Ethernet				
EXTI	prw	1 KB	0x40013C00	External interrupt/event
FLASH	prw	1 KB	0x40023C00	FLASH
FSMC	prw	1 KB	0xA0000000	Flexible static memory contrc
GPIO				

System map | Project zone map | Blinky_MPU | CoProc

Resources may be imported from CMSIS-Pack and CMSIS-SVD files

CMSIS-Zone – Development Workflow

Resource configuration for multi-processor systems and execution regions



MCBSTM32F400.szone

Blinky_MPU

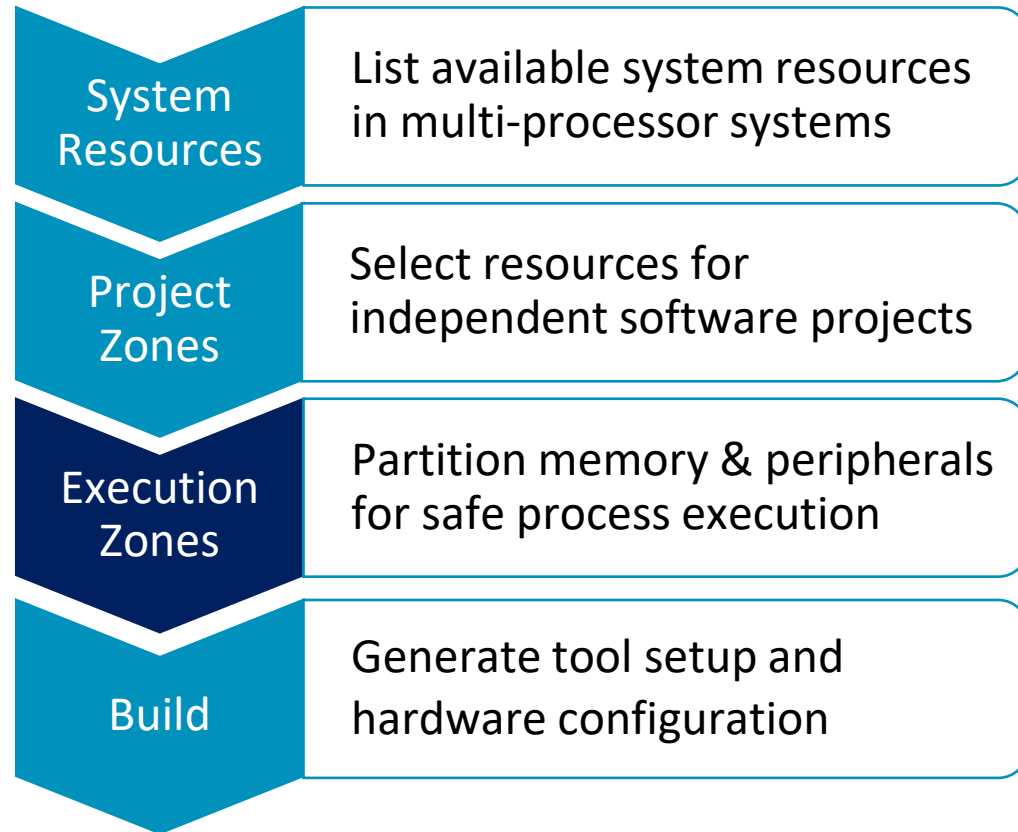
Generate code

Name		Access	Size	Start	Description
STM32F407IG					ARM Cortex-M4 168 MHz,
memory					
IRAM1	<input checked="" type="checkbox"/>	rwX	128 KB	0x20000000	
IRAM2	<input type="checkbox"/>	rwX	64 KB	0x10000000	
IROM1	<input checked="" type="checkbox"/>	rx	1 MB	0x08000000	
peripherals					
ADC					
ADC1	<input checked="" type="checkbox"/>	prw	1 KB	0x40012000	Analog-to-digital converter
ADC2	<input checked="" type="checkbox"/>	prw	1 KB	0x40012100	
ADC3	<input checked="" type="checkbox"/>	prw	1 KB	0x40012200	
C_ADC	<input checked="" type="checkbox"/>	prw	1 KB	0x40012300	Common ADC registers
CAN	<input type="checkbox"/>				
CRC	<input type="checkbox"/>	prw	1 KB	0x40023000	Cryptographic processor
DAC	<input type="checkbox"/>	prw	1 KB	0x40007400	Digital-to-analog converter
DBG	<input type="checkbox"/>	prw	1 KB	0xE0042000	Debug support
DCMI	<input type="checkbox"/>	prw	1 KB	0x50050000	Digital camera interface
DMA	<input type="checkbox"/>				

System map | Project zone map | Blinky_MPU | CoProc

CMSIS-Zone – Development Workflow

Resource configuration for multi-processor systems and execution regions

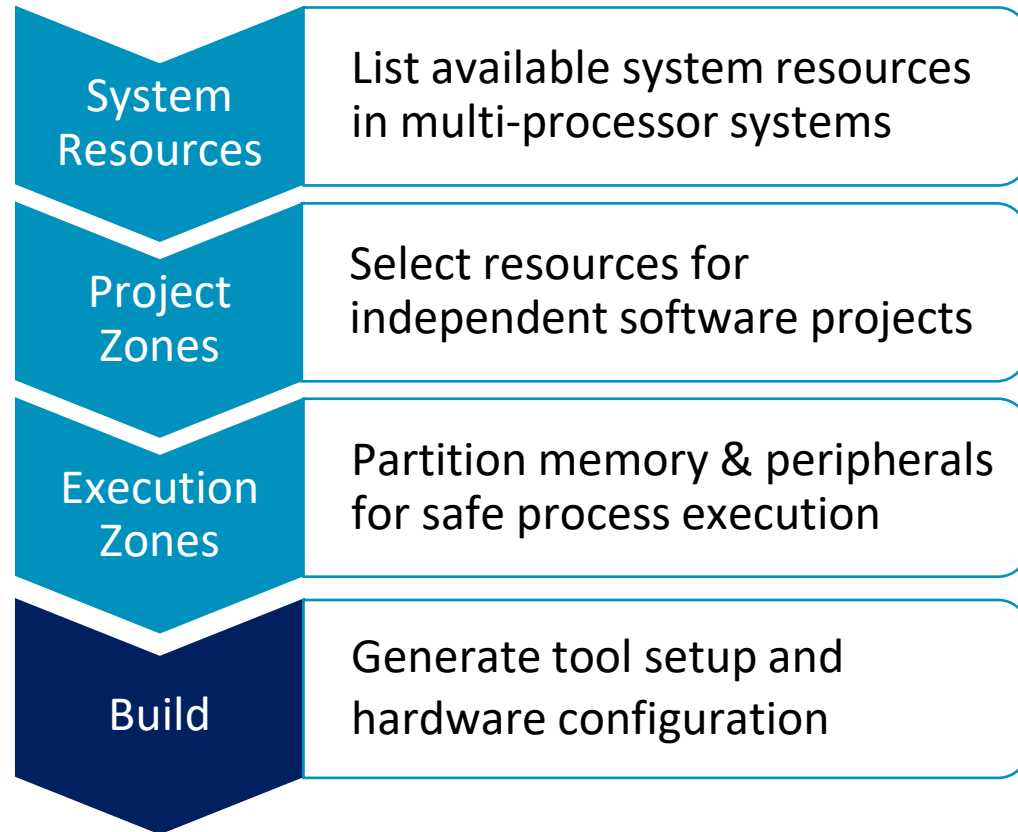


The screenshot shows the 'Execution zone map' tool interface. The top bar displays the project name 'MCBSTM32F400.szone' and the target 'Blinky_MPU.pzone'. The main table lists the execution zones for the 'STM32F407IG' target, showing the allocation of memory and peripherals for the RTOS, ProcessA, and ProcessB.

Name	RTOS	ProcessA	ProcessB	Description
STM32F407IG				ARM Cortex-M4 168 M
memory				
IRAM1	0x20000000	0x20000000	0x20000000	
priv	0x20000000	0x20000000	0x20000000	Privileged RW data
shared	0x20008000	0x20008000	0x20008000	Shared RW data
processA	0x20010000	0x20010000	0x20010000	Process A RW data
processB	0x20018000	0x20018000	0x20018000	Process B RW data
IROM1	0x08000000	0x08000000	0x08000000	
flash	0x80000000	0x80000000	0x80000000	Flash ROM
peripherals				
ADC1	0x40012000	0x40012000	0x40012000	Analog-to-digital con
ADC2	0x40012100	0x40012100	0x40012100	
ADC3	0x40012200	0x40012200	0x40012200	
C_ADC	0x40012300	0x40012300	0x40012300	Common ADC register
GPIOA	0x40020000	0x40020000	0x40020000	General-purpose I/Os
GPIOB	0x40020400	0x40020400	0x40020400	General-purpose I/Os

CMSIS-Zone – Development Workflow

Resource configuration for multi-processor systems and execution regions



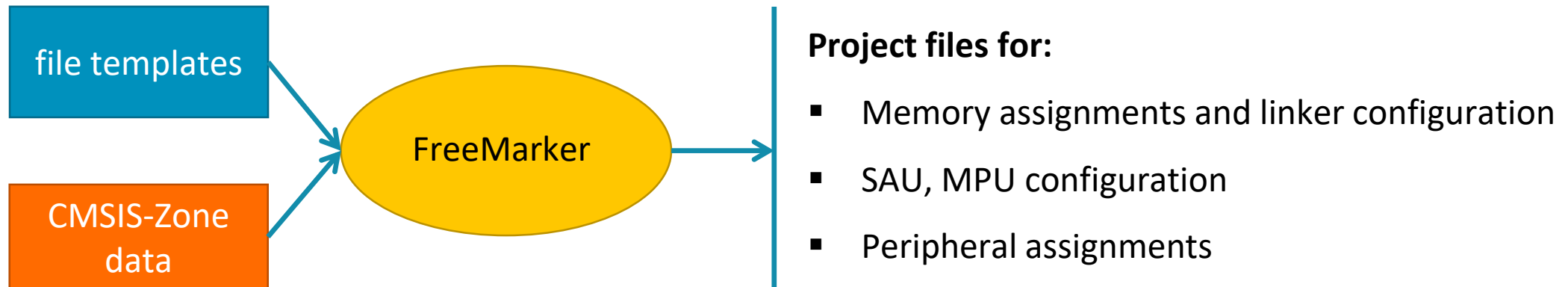
```
MCBSTM32F400.szone Blinky_MPU.pzone scatter.sct
9 LR_flash 0x08000000 0x00080000 { ; load region
10 ER_flash 0x08000000 0x00080000 { ; Flash ROM
11 *.o (RESET, +First)
12 *(InRoot$$Sections)
13 .ANY (+RO)
14 .ANY (+XO)
15 }
16 RW_priv 0x20000000 0x00008000 { ; Privileged RW data
17 .ANY (+RW +ZI)
18 .ANY (.data.priv*)
19 .ANY (.bss.priv*)
20 }
21 RW_shared 0x20008000 0x00008000 { ; Shared RW data
22 .ANY (.data.shared*)
23 .ANY (.bss.shared*)
24 }
25 RW_processA 0x20010000 0x00000200 { ; Process A RW data
26 .ANY (.data.processA*)
27 .ANY (.bss.processA*)
28 }
29 RW_processB 0x20010400 0x00000200 { ; Process B RW data
30 .ANY (.data.processB*)
31 .ANY (.bss.processB*)
```


CMSIS-Zone – data export for projects

FreeMarker template engine allows to export CMSIS-Zone data to arbitrary formats

Build

Flexible data export for project build supports many different use cases:
i.e. device configuration, MPU setup, linker scripts, etc.



CMSIS-Zone – Generator example: Linker Script

Derived linker sections for project and process segregation

Template

```
22 // <compress>
23
24 LR_<@blockSpec erBlocks[0]/> { ; load region
25 <#list erBlocks as block>
26   ER_<@blockSpec block/> { ; ${block.info!""}
27   <#if block?index == 0>
28     *.o (RESET, +First)
29     *(InRoot$$Sections)
30     .ANY (+RO)
31     .ANY (+XO)
32   </#if>
33   }
34 </#list>
35 <#list rwBlocks as block>
36   RW_<@blockSpec block/> { ; ${block.info!""}
37   <#if block?index == 0>
38     .ANY (+RW +ZI)
39   </#if>
40     .ANY (.data.${block.name}*)
41     .ANY (.bss.${block.name}*)
42   }
43 </#list>
44 }
```

Generator Output

```
9 LR_flash 0x08000000 0x00080000 { ; load region
10   ER_flash 0x08000000 0x00080000 { ; Flash ROM
11     *.o (RESET, +First)
12     *(InRoot$$Sections)
13     .ANY (+RO)
14     .ANY (+XO)
15   }
16   RW_priv 0x20000000 0x00008000 { ; Privileged RW data
17     .ANY (+RW +ZI)
18     .ANY (.data.priv*)
19     .ANY (.bss.priv*)
20   }
21   RW_shared 0x20008000 0x00008000 { ; Shared RW data
22     .ANY (.data.shared*)
23     .ANY (.bss.shared*)
24   }
25   RW_processA 0x20010000 0x00000200 { ; Process A RW data
26     .ANY (.data.processA*)
27     .ANY (.bss.processA*)
28   }
29   RW_processB 0x20010400 0x00000200 { ; Process B RW data
30     .ANY (.data.processB*)
31     .ANY (.bss.processB*)
32   }
```

CMSIS-Zone – Generator example: MPU Descriptors

Derived MPU regions for process segregation

Template

```
5
6 // Device:    ${device.name}
7 // Processor: ${processor.name}
8 // Project:   ${name}
9
10 #include "mputable.h"
11
12 const ARM_MPU_Region_t mpuTable[${regions?size}][${maxBlocks}]
13 <#list regions as name, region>
14     /* ${name} */
15     {
16 <#list fillup(region["blocks"], maxBlocks) as block>
17 <#if block?index lt region["blocks"?size>
18         // ${block["name"]}
19         { .RBAR = ${num2hex(block["start"], "0x", 8)}U | (${num2hex
20 <#else>
21         { .RBAR = (${num2hex(region["offset"]+block?index, "0x", 2)
22 </#if>
23 </#list><#nt>
24     }<#sep>,
25 </#list><#nt>
26 };
27
```

Generator Output

```
4
5 // Device:    STM32F407IG
6 // Processor: Cortex-M4
7 // Project:   Blinky_MPU
8
9 #include "mputable.h"
10
11 const ARM_MPU_Region_t mpuTable[4][3] = {
12     /* default */
13     {
14         // flash
15         { .RBAR = 0x08000000U | (0x00U & 0x0FU) | 0x10U, .RASR = 7
16         // data
17         { .RBAR = 0x08080000U | (0x01U & 0x0FU) | 0x10U, .RASR = 7
18         // shared
19         { .RBAR = 0x20008000U | (0x02U & 0x0FU) | 0x10U, .RASR = 7
20     },
21     /* RTOS */
22     {
23         // priv
24         { .RBAR = 0x20000000U | (0x03U & 0x0FU) | 0x10U, .RASR = 7
25         { .RBAR = (0x04U & 0x0FU) | 0x10U, .RASR = 0U },
26         { .RBAR = (0x05U & 0x0FU) | 0x10U, .RASR = 0U }
```

CMSIS-Zone – Generator example: SAU Regions

Derived SAU regions for TrustZone configuration

Template

```
97 //  <e>Initialize SAU Region ${sauRegion?index}
98 //  <i> Setup SAU Region ${sauRegion?index} memory attributes
99 */
100 #define SAU_INIT_REGION${sauRegion?index}    ${sauRegion.init}
101
102 /*
103 //      <o>Start Address <0-0xFFFFFFFFE0>
104 */
105 #define SAU_INIT_START${sauRegion?index}      ${num2hex(sauRegi
106
107 /*
108 //      <o>End Address <0x1F-0xFFFFFFFF>
109 */
110 #define SAU_INIT_END${sauRegion?index}        ${num2hex(max(sau
111
112 /*
113 //      <o>Region is
114 //          <0=>Non-Secure
115 //          <1=>Secure, Non-Secure Callable
116 */
117 #define SAU_INIT_NSC${sauRegion?index}        ${sauRegion.nsc}
118 /*
119 //      </>
```

Generator Output

```
45 //  <e>Initialize SAU Region 0
46 //  <i> Setup SAU Region 0 memory attributes
47 */
48 #define SAU_INIT_REGION0    1
49
50 /*
51 //      <o>Start Address <0-0xFFFFFFFFE0>
52 */
53 #define SAU_INIT_START0      0x001FF000      /* start address
54
55 /*
56 //      <o>End Address <0x1F-0xFFFFFFFF>
57 */
58 #define SAU_INIT_END0        0x001FFFFFF      /* end address of
59
60 /*
61 //      <o>Region is
62 //          <0=>Non-Secure
63 //          <1=>Secure, Non-Secure Callable
64 */
65 #define SAU_INIT_NSC0        1
66 /*
67 //      </>
```

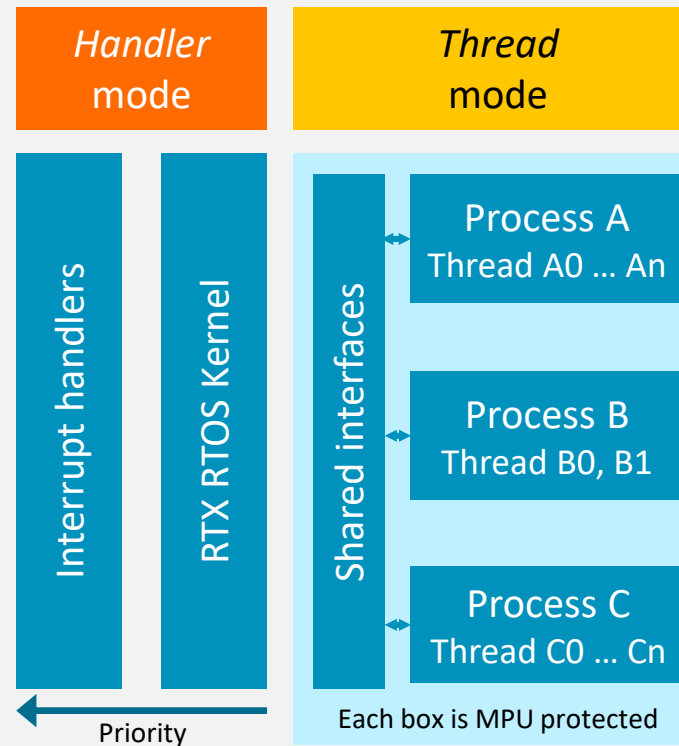
RTX – RTOS with optional process isolation



CMSIS-Zone System Partitioning

Name	Access	Size	Process A	Process B
STM32F407IG				
memory				
IRAM1	rwX	128 KB	0x20000000	0x20000000
RamA	rwX	64 KB	0x20000000	0x20000000
RamB	rwX	64 KB	0x20010000	0x20010000
IRAM2	rwX	64 KB	0x10000000	0x10000000
App	rx	256 KB	0x8010000	0x8010000
Storage	r	128 KB	0x8090000	0x8090000
Scratch	r	256 KB	0x8050000	0x8050000
peripherals				
ADC1	prw	1 KB	0x40012000	0x40012000
ADC2	prw	1 KB	0x40012100	0x40012100
ADC3	prw	1 KB	0x40012200	0x40012200
C_ADC	prw	1 KB	0x40012300	0x40012300
GPIOA	prw	1 KB	0x40020000	0x40020000
GPIOB	prw	1 KB	0x40020400	0x40020400
GPIOC	prw	1 KB	0x40020800	0x40020800
GIOD	prw	1 KB	0x40020C00	0x40020C00
GPIOE	prw	1 KB	0x40021000	0x40021000
GPIOF	prw	1 KB	0x40021400	0x40021400

RTX5 RTOS Safe Process Isolation



RTX optional uses the Protection Unit (MPU) of Cortex-M processors

The MPU isolates processes and protects from incorrect accesses to data and peripherals

CMSIS-Zone simplifies the setup of MPU protected execution zones



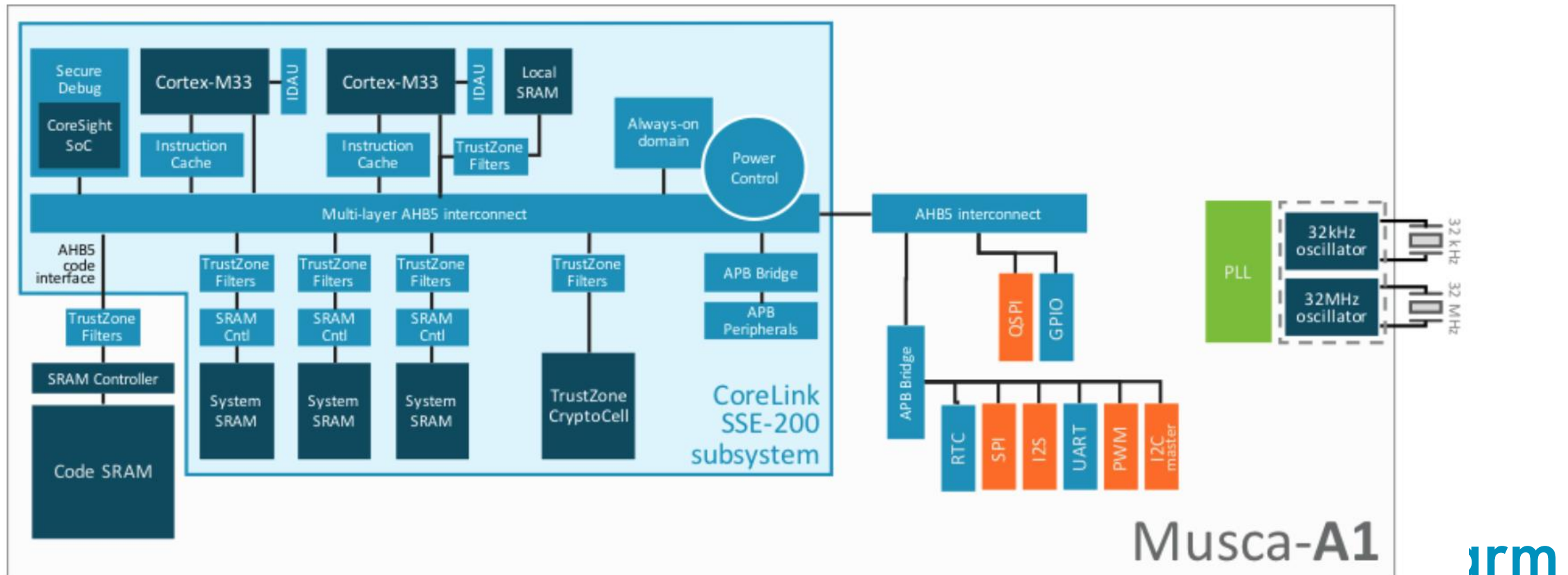
Summary

Reinhard Keil
Sr. Director Embedded Tools

Simplify software development for complex systems

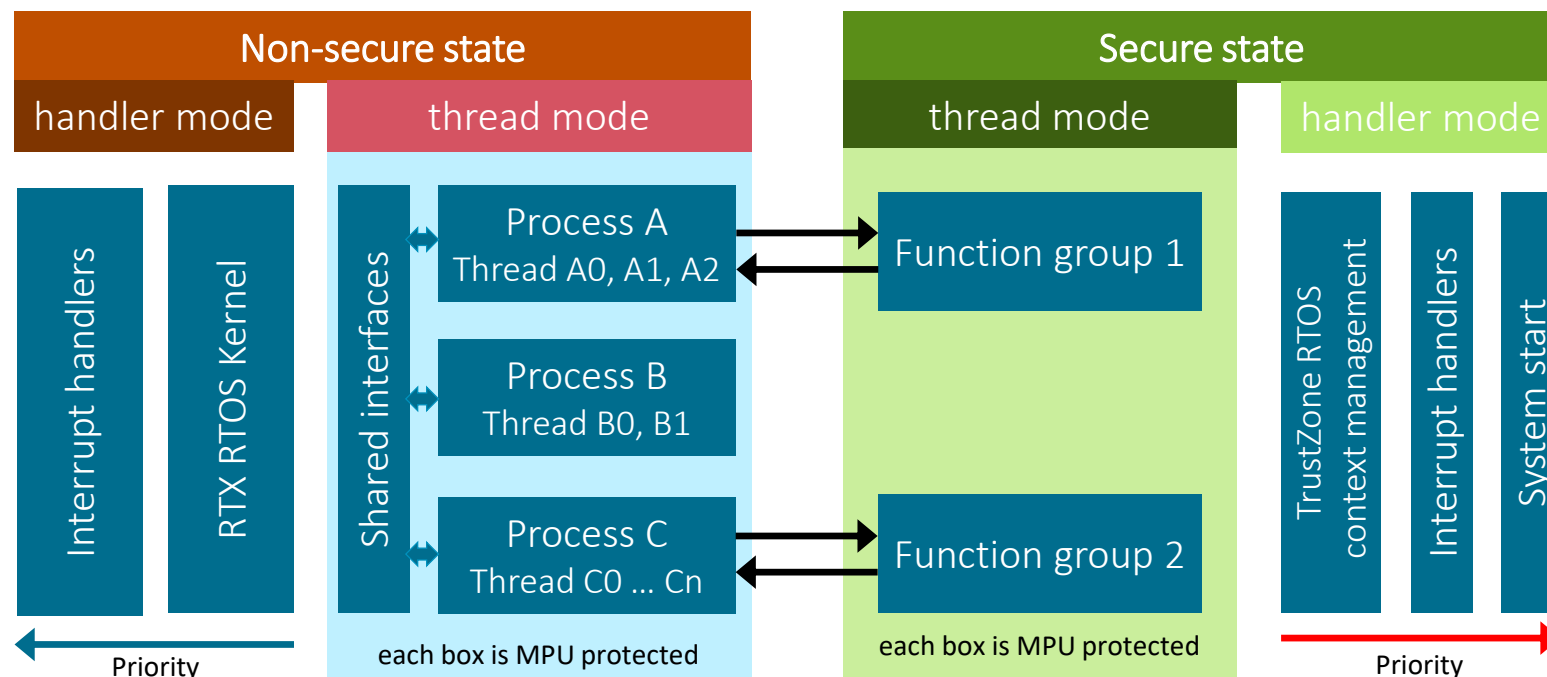
CMSIS-Zone is design to configure security and/or complex multi-processor systems

- Specifies resources (processor, memory, peripherals)
- Generates hardware configuration files
- Working on examples for Musca-A1, i.MX7
- CMSIS-Zone Prototype available in April 2018



Example with RTOS and MPU setup

Add memory protection unit to extend security to process/thread execution



CMSIS-Zone can be used to generate setup of single core systems.

Utilizes Cortex-M processor features:

- MPU setup for process isolation
- SAU setup for Cortex-M33

Platform Security Architecture - Standardized Interfaces

PSA specifies interfaces to decouple components.

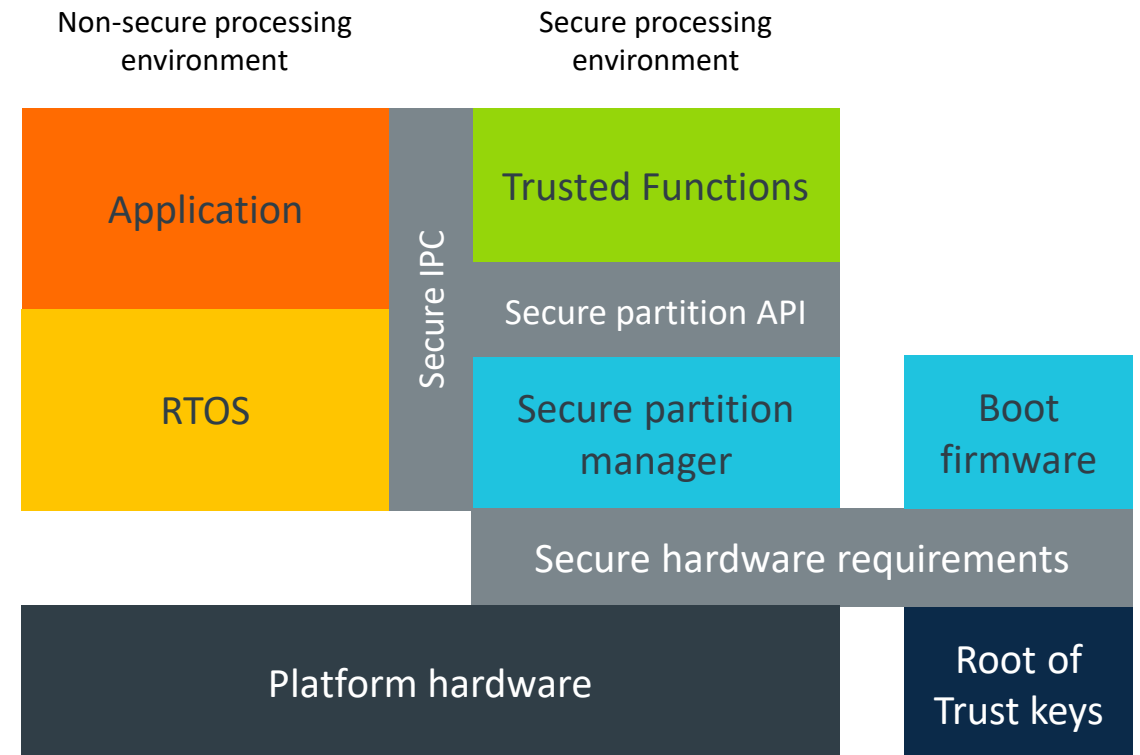
- Enables reuse of components in other device platforms
- Reduces integration effort

Partners can provide alternative implementations.

- Necessary to address different cost, footprint, regulatory or security needs

PSA provides an architectural specification.

- Hardware, firmware and process requirements and interfaces



Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!

arm