How to validate re-usable software components

Linaro WG Meeting

Arm MCU Tools Team
16 May 2023

Agenda

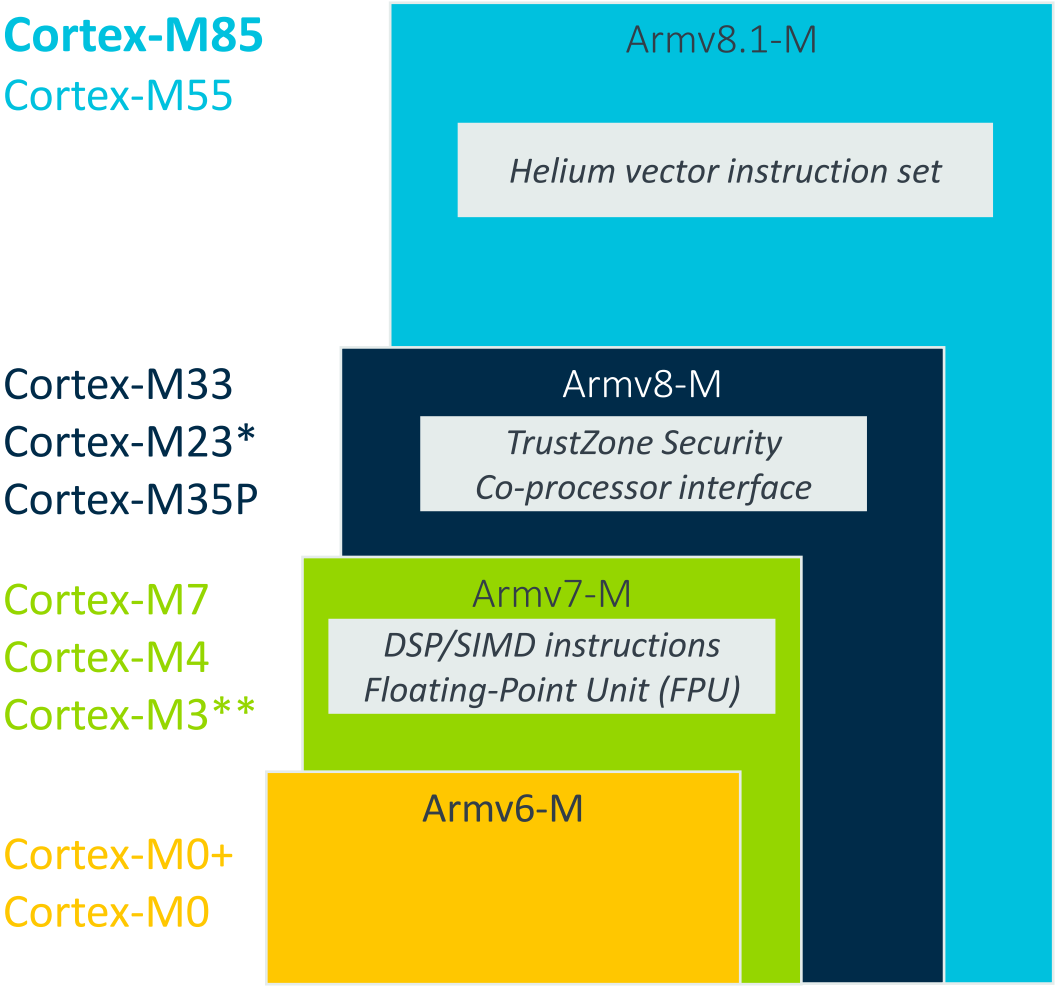
- What are “re-usable software components”?
- Industry-standard test processes (DevOps, Test Driven Development)
- Continuous Integration (CI) Build Process with CMSIS-Toolbox
  - Example projects
  - Build for multiple compilers and targets
- Test-Process Arm Virtual Hardware (AVH)
  - Using Virtual Interfaces
- CMSIS-VIO: a simple I/O interface for testing and example projects
- Open items and Discussion

What are “re-usable software components”?

Re-useable software components …

- Allow integration into many different software projects and different targets.
- Work with different toolchains and different compiler options.
- Use standardized interfaces to connect with device specific I/O.
- Use established verification and validation development processes.
  - that are independent of final target hardware.
- Still the software components are optimized for the target architecture.
  - Algorithms are optimized towards the processor architecture.
  - Device specific I/O interfaces are flexible enough to support different methods (IRQ, DMA).

Tools for testing on whole Cortex-M Processor Portfolio



CMSIS-Toolbox – supports multiple compilers, multiple target-types, and multiple build-types.

- CMSIS_DFP defines setup for all processors
- `Cbuild --toolchain` switches compilers

Arm Virtual Hardware / FVP supports all Cortex-M processors with Compiler, Simulation Models

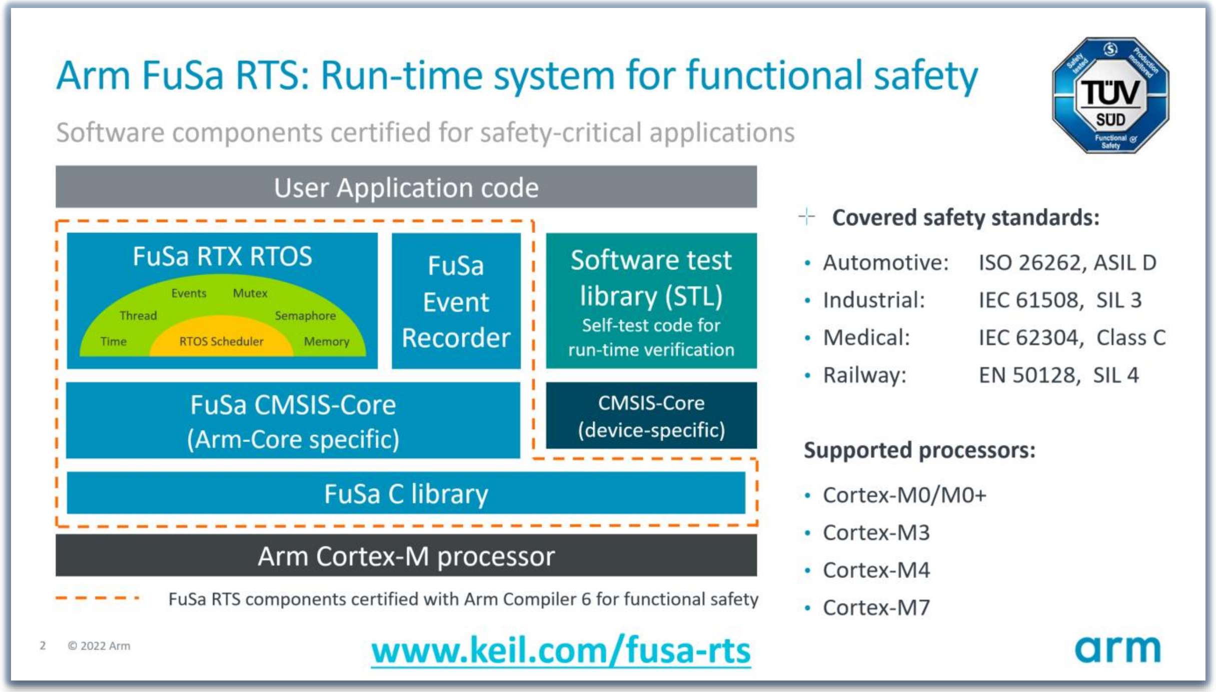
- Available as Cloud service and Desktop variant
- AVH models offer virtual I/O interfaces for
  - Simple I/O (LED, buttons)
  - Data streaming (Sensor, Audio, Video)
  - Connectivity via Ethernet and Socket

MDK supports desktop development

Arm uses these tools widely

For testing of CMSIS components, Arm FuSa RTS, TF-M, Compiler development

- Test process is predominately done on models
  - Only very few physical targets are used to show consistency



Arm FuSa RTS: Run-time system for functional safety

Software components certified for safety-critical applications

User Application code

FuSa RTX RTOS

FuSa Event Recorder

Software test library (STL)
Self-test code for run-time verification

FuSa CMSIS-Core (Arm-Core specific)

CMSIS-Core (device-specific)

FuSa C library

Arm Cortex-M processor

FuSa RTS components certified with Arm Compiler 6 for functional safety

Covered safety standards:

Automotive:	ISO 26262, ASIL D
Industrial:	IEC 61508, SIL 3
Medical:	IEC 62304, Class C
Railway:	EN 50128, SIL 4

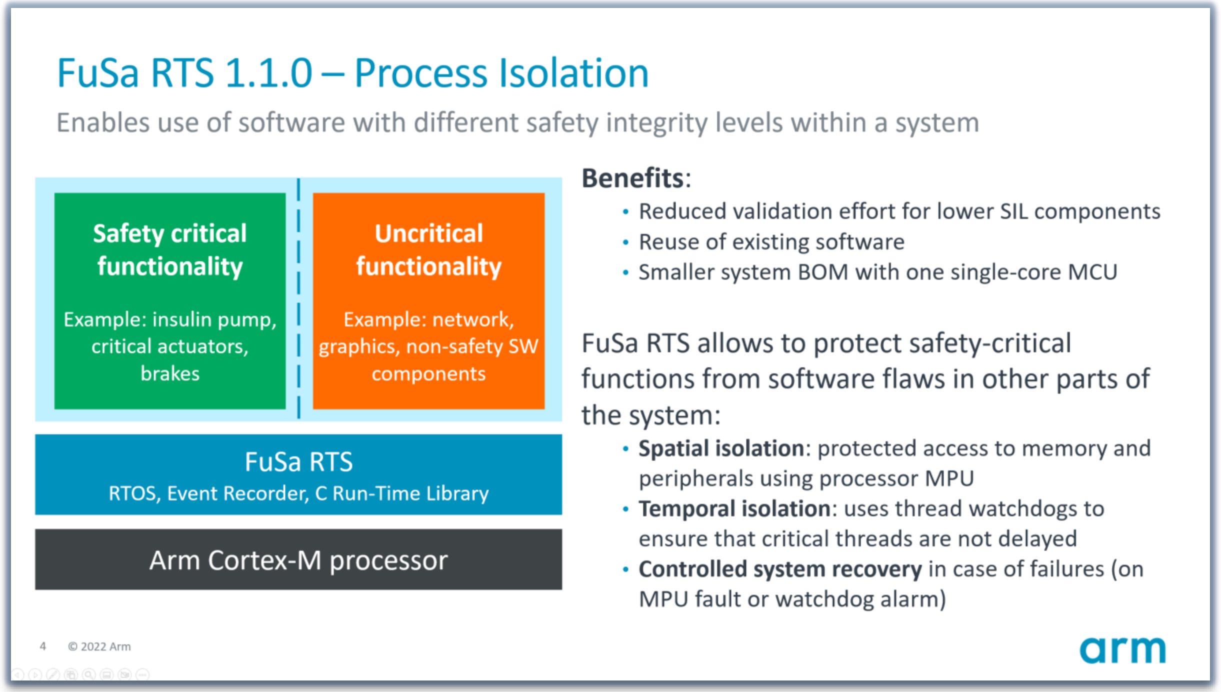
Supported processors:

- Cortex-M0/M0+
- Cortex-M3
- Cortex-M4
- Cortex-M7

2 © 2022 Arm

www.keil.com/fusa-rts

arm



FuSa RTS 1.1.0 – Process Isolation

Enables use of software with different safety integrity levels within a system

Safety critical functionality
Example: insulin pump, critical actuators, brakes

Uncritical functionality
Example: network, graphics, non-safety SW components

FuSa RTS
RTOS, Event Recorder, C Run-Time Library

Arm Cortex-M processor

Benefits:

- Reduced validation effort for lower SIL components
- Reuse of existing software
- Smaller system BOM with one single-core MCU

FuSa RTS allows to protect safety-critical functions from software flaws in other parts of the system:

- **Spatial isolation**: protected access to memory and peripherals using processor MPU
- **Temporal isolation**: uses thread watchdogs to ensure that critical threads are not delayed
- **Controlled system recovery** in case of failures (on MPU fault or watchdog alarm)

4 © 2022 Arm

arm

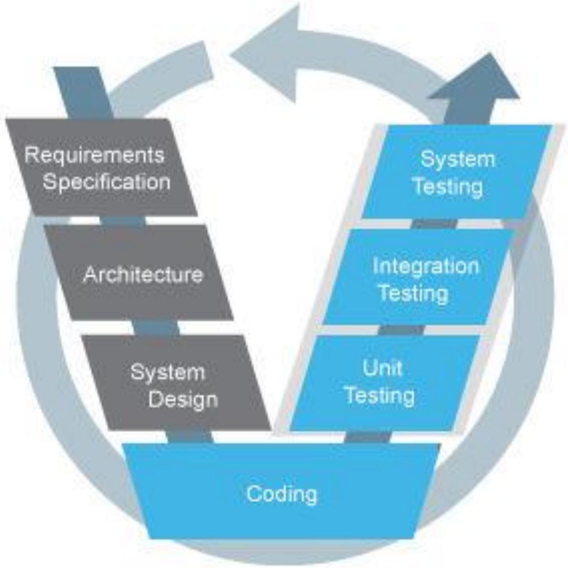
Cloud-based Continuous Integration (CI)

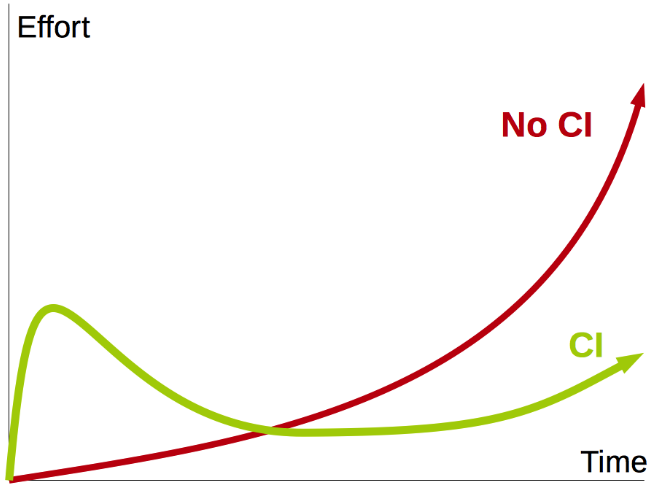
Test and Verification with
Arm Virtual Hardware (AVH)

Types of Software Testing

Better quality faster, conforming to safety standards

- Unit Testing
  - Test little chunks of code at a time
  - Tested against your ‘test’ build
- Integration Testing
  - Test whether two components work together when they are combined Verifies that the interface between them works properly
  - Tested against your ‘test’ build
- System (Black-box) Testing
  - Test that final system works as expected. Control external controls & stimuli to system and measure response
  - Tested against your ‘release’ build
- **Regression Testing**
  - **Suite of tests (unit & integration tests) & run continuously upon version control updates**
  - **Used in Continuous Integration (CI)**





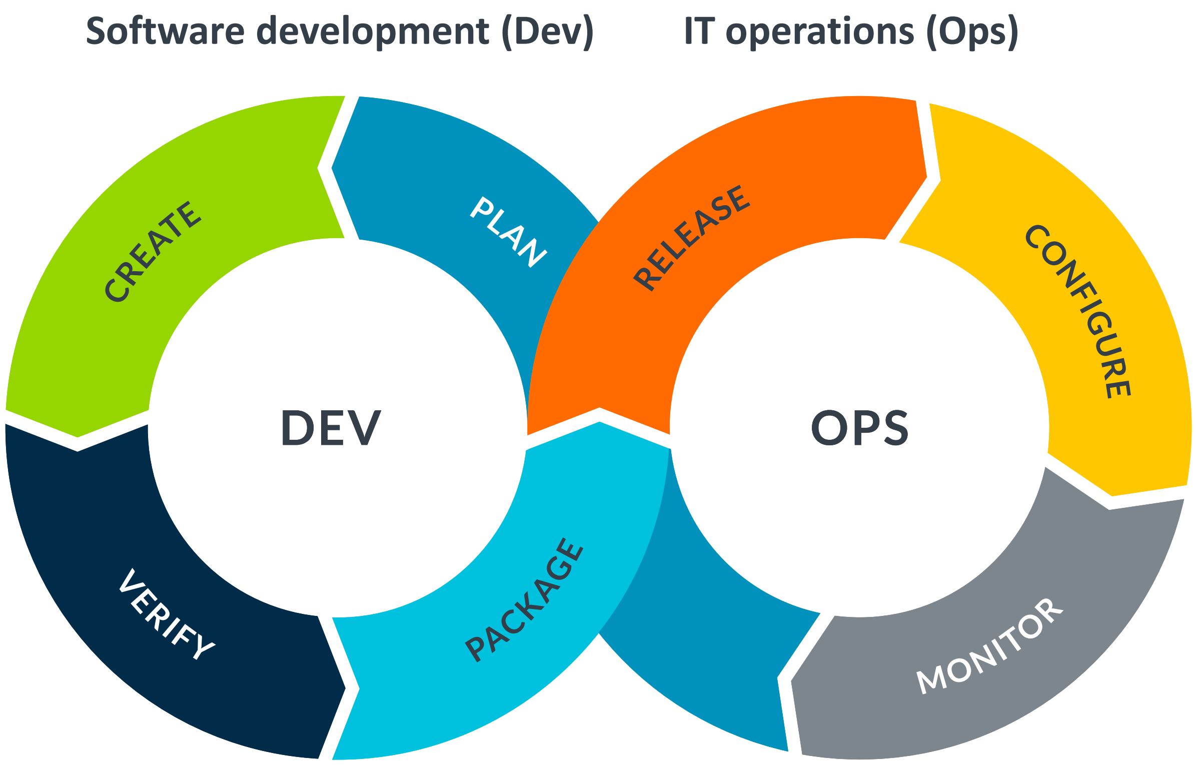
What is Test-driven Development, DevOps, and CI/CD

Test-driven development embraces that software requirements are converted to test cases before software is fully developed. It implements the test-first programming concepts of extreme programming.

DevOps combines software development (Dev) and IT operations (Ops) to shorten system development by providing continuous integration, test, and delivery.

CI/CD services (provide the Ops part)

- Integrate incremental code changes of several developers into production code
- Run automated tests to verify functionality
- Deploy firmware images to test fleets
- Large scale delivery to many IoT endpoint devices



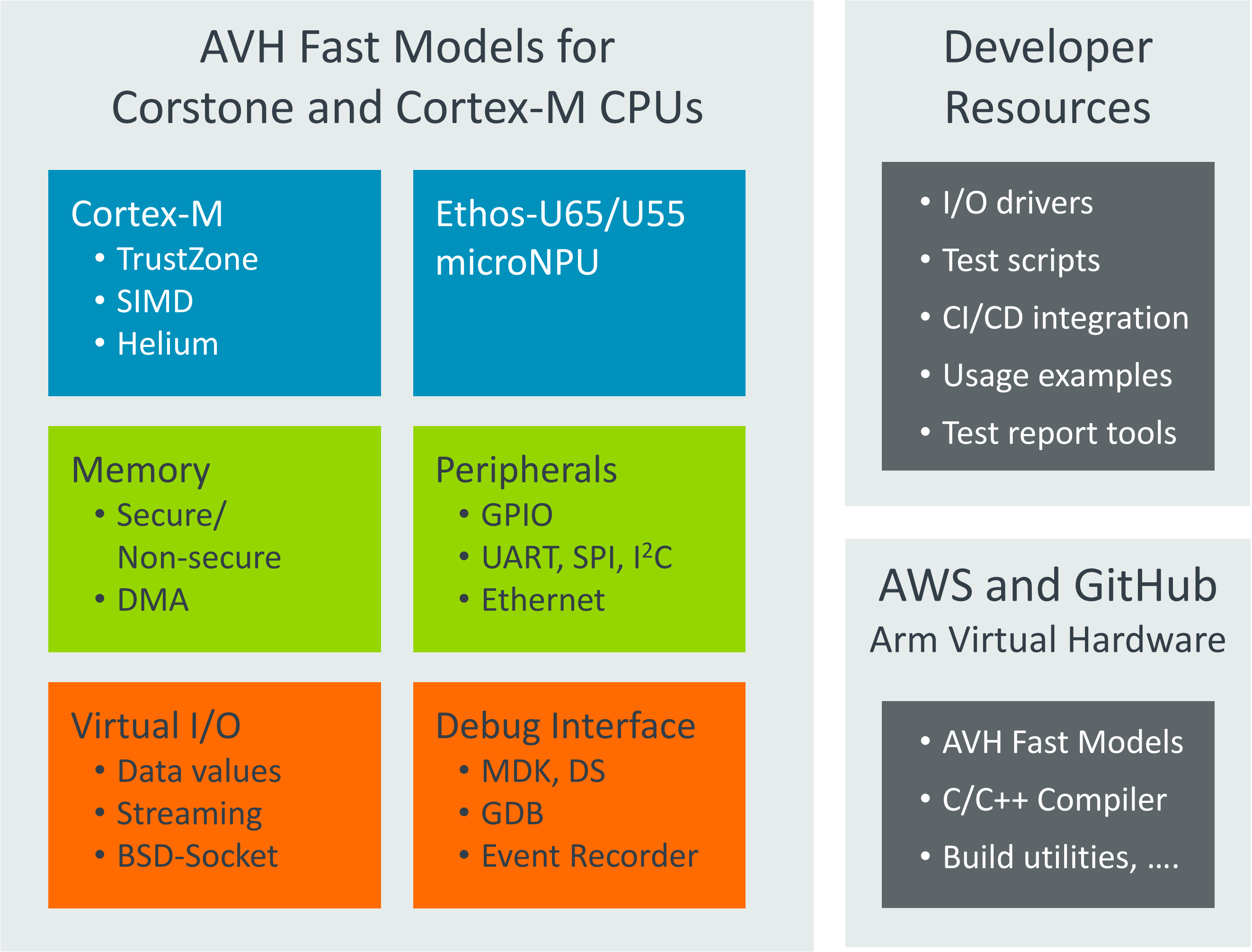
Source https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg

Combined, these techniques enable agile software development with the ability to make code changes quicker!

Arm Virtual Hardware (AVH) – Corstone and Cortex-M CPUs

www.arm.com/virtual-hardware

- Precise **simulation models** of Cortex-M device sub-systems designed for complex software verification and testing
- Runs any RTOS or bare metal code
- Provides virtual peripheral interfaces for I/O simulation
- Enables test automation of diverse software workloads, including unit, integration tests, and fault injection
- Cloud service that can be integrated in **CI/CD** and **MLOps** development flows



AVH Fast Models for Corstone and Cortex-M CPUs

Cortex-M
- TrustZone
- SIMD
- Helium

Ethos-U65/U55 microNPU

Memory
- Secure/ Non-secure
- DMA

Peripherals
- GPIO
- UART, SPI, I^2C
- Ethernet

Virtual I/O
- Data values
- Streaming
- BSD-Socket

Debug Interface
- MDK, DS
- GDB
- Event Recorder

Developer Resources

- I/O drivers
- Test scripts
- CI/CD integration
- Usage examples
- Test report tools

AWS and GitHub

Arm Virtual Hardware

- AVH Fast Models
- C/C++ Compiler
- Build utilities, ….

Create and Debug

IDE development
Local installation

Keil MDK with:

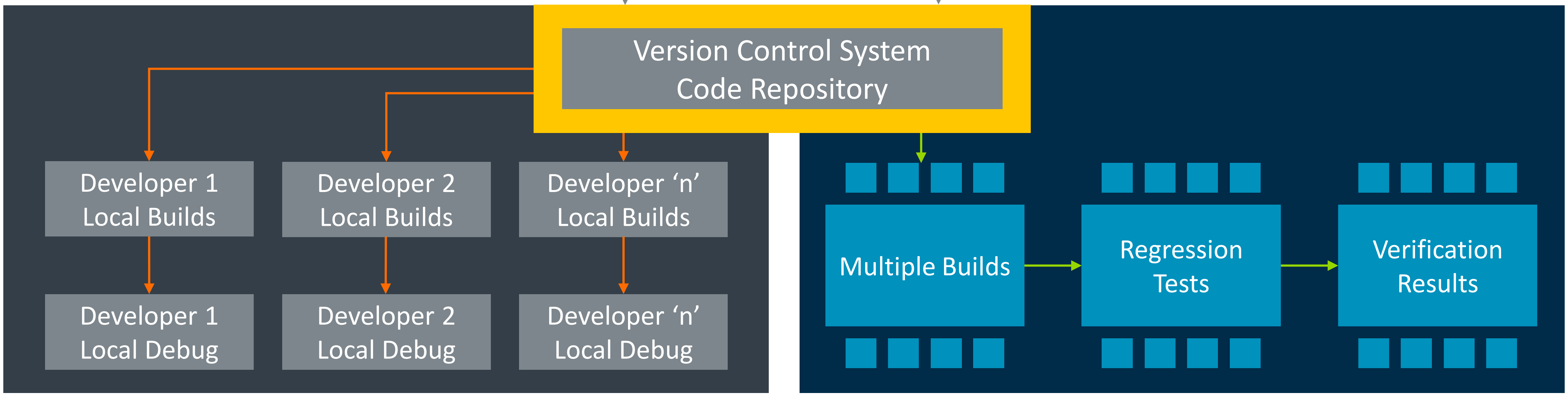
- Interactive Debug and Trace Views
- Arm Compiler
- Arm Virtual Hardware

Test Automation

CI development
Jenkins or GitHub

Cloud Service with:

- Arm Virtual Hardware
- Arm Compiler
- Compatible build tools



Version Control System
Code Repository

Developer 1
Local Builds

Developer 2
Local Builds

Developer ‘n’
Local Builds

Developer 1
Local Debug

Developer 2
Local Debug

Developer ‘n’
Local Debug

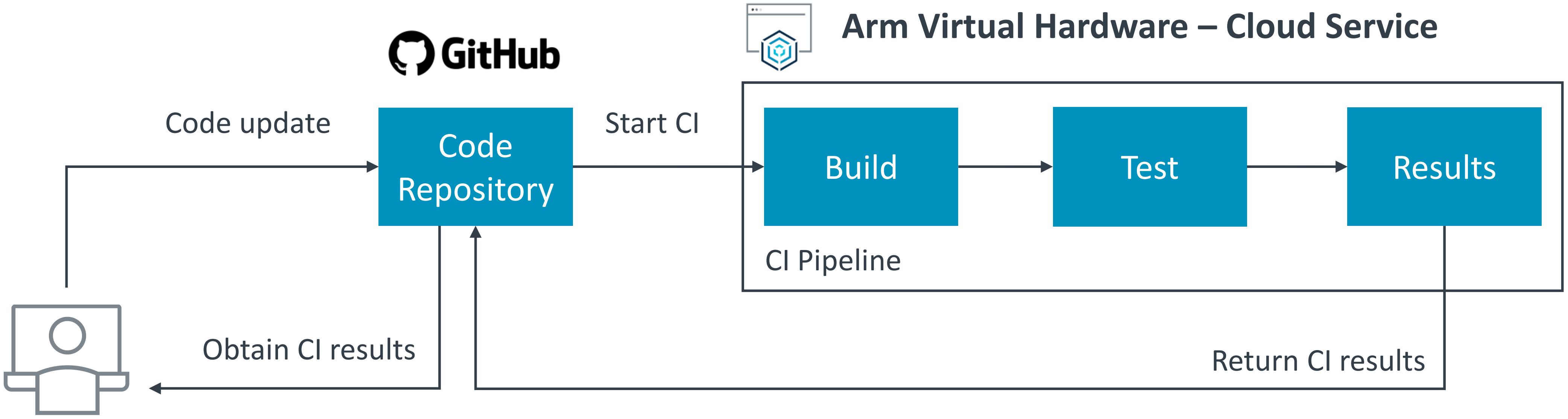
Multiple Builds

Regression
Tests

Verification
Results

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.

CMSIS-Toolbox: Test System Configuration

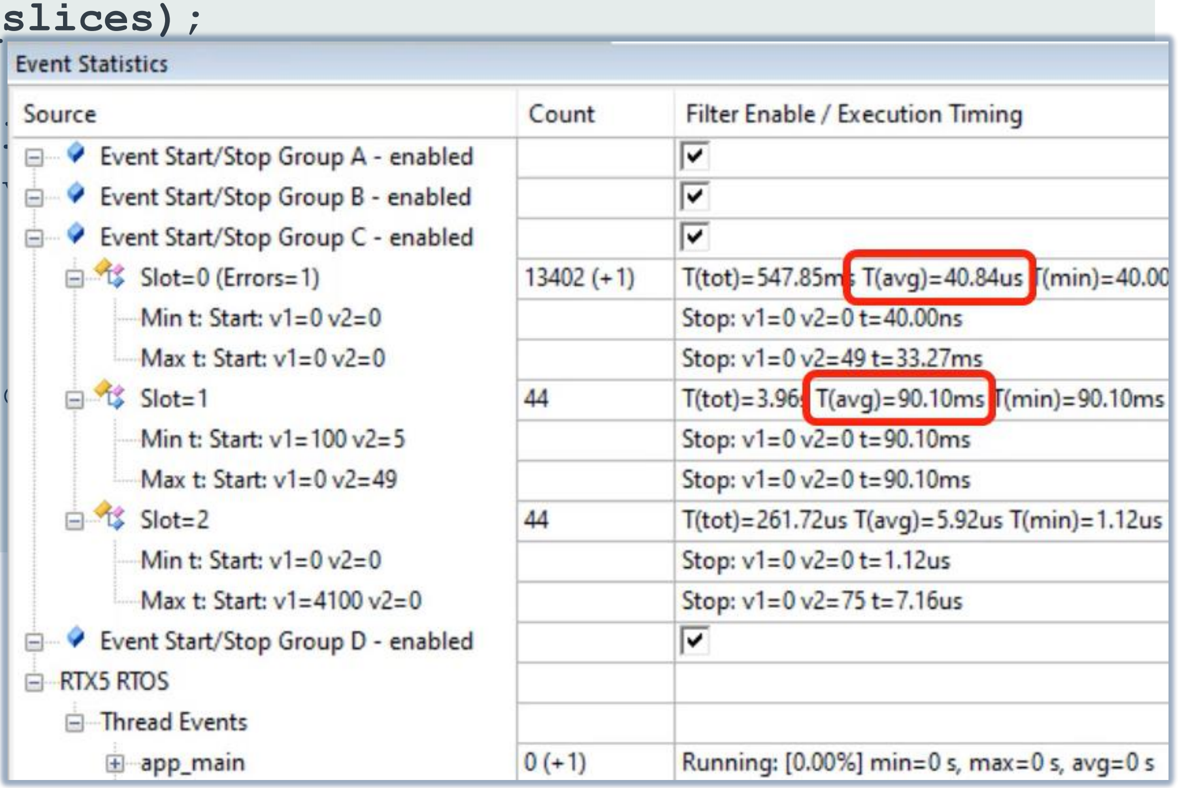
Build support for multiple compilers, multiple target-types, and multiple build-types.

- Overview of Operation describes elements for setup of “test.csolution.yml”
  - cdefault.yml allows to switch compilers
  - target-types allow to define multiple test targets (i.e. Cortex-M3, Cortex-M4, …, Cortex-M85)
  - build-types allow to define build variants (could be different compiler optimizations)
- A “test.csolution.yml” can have multiple projects that share this common configuration
  - Enables unit test projects for verification, i.e. with Arm Virtual Hardware
- **CMSIS-Toolbox cbuild** is designed for effective build orchestration:

```
> cbuild list toolchains      # Generate all project variants for AC6
AC6@6.19.0                    > cbuild test.csolution.yml -r -p --toolchain AC6
GCC@12.2.1
                              # Generate all project variants for GCC
                              > cbuild test.csolution.yml -r -p --toolchain GCC
```

Get Execution Details with CMSIS-View Event Annotations

```
void loop() {
   :
  EventStartCv(0, current_time, previous_time);
  TfLiteStatus feature_status = feature_provider->PopulateFeatureData(error_reporter,
                                                    previous_time, current_time, …)
  EventStopCv(0, feature_status, how_many_new_slices);
   :
  EventStartCv(1, current_time, how_many_new_s
  TfLiteStatus invoke_status = interpreter->In
  EventStopCv(1, invoke_status, 0U);
   :
  EventStartCv(2, current_time, 0U);
  TfLiteStatus process_status = recognizer->Pr

  EventStopCv(2, process_status, score);
```



Event Statistics

Source	Count	Filter Enable / Execution Timing
⊟ Event Start/Stop Group A - enabled		☑
⊟ Event Start/Stop Group B - enabled		☑
⊟ Event Start/Stop Group C - enabled		☑
⊟ Slot=0 (Errors=1)	13402 (+1)	T(tot)=547.85ms T(avg)=40.84us T(min)=40.00
Min t: Start: v1=0 v2=0		Stop: v1=0 v2=0 t=40.00ns
Max t: Start: v1=0 v2=0		Stop: v1=0 v2=49 t=33.27ms
⊟ Slot=1	44	T(tot)=3.96 T(avg)=90.10ms T(min)=90.10ms
Min t: Start: v1=100 v2=5		Stop: v1=0 v2=0 t=90.10ms
Max t: Start: v1=0 v2=49		Stop: v1=0 v2=0 t=90.10ms
⊟ Slot=2	44	T(tot)=261.72us T(avg)=5.92us T(min)=1.12us
Min t: Start: v1=0 v2=0		Stop: v1=0 v2=0 t=1.12us
Max t: Start: v1=4100 v2=0		Stop: v1=0 v2=75 t=7.16us
⊟ Event Start/Stop Group D - enabled		☑
⊟ RTX5 RTOS		
⊟ Thread Events		
⊞ app_main	0 (+1)	Running: [0.00%] min=0 s, max=0 s, avg=0 s

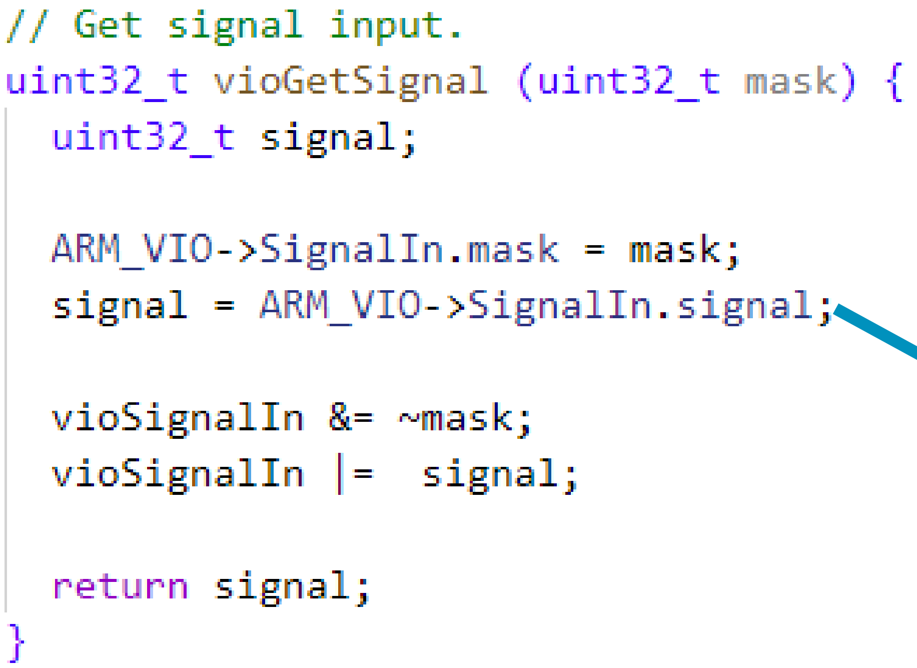
More information:

- Event Statistics – code annotation
- eventlist – command line utility

CI Example: VIO_Blinky using CMSIS-VIO

Test-Automation with simple I/O (on physical board this would be LEDs and buttons)

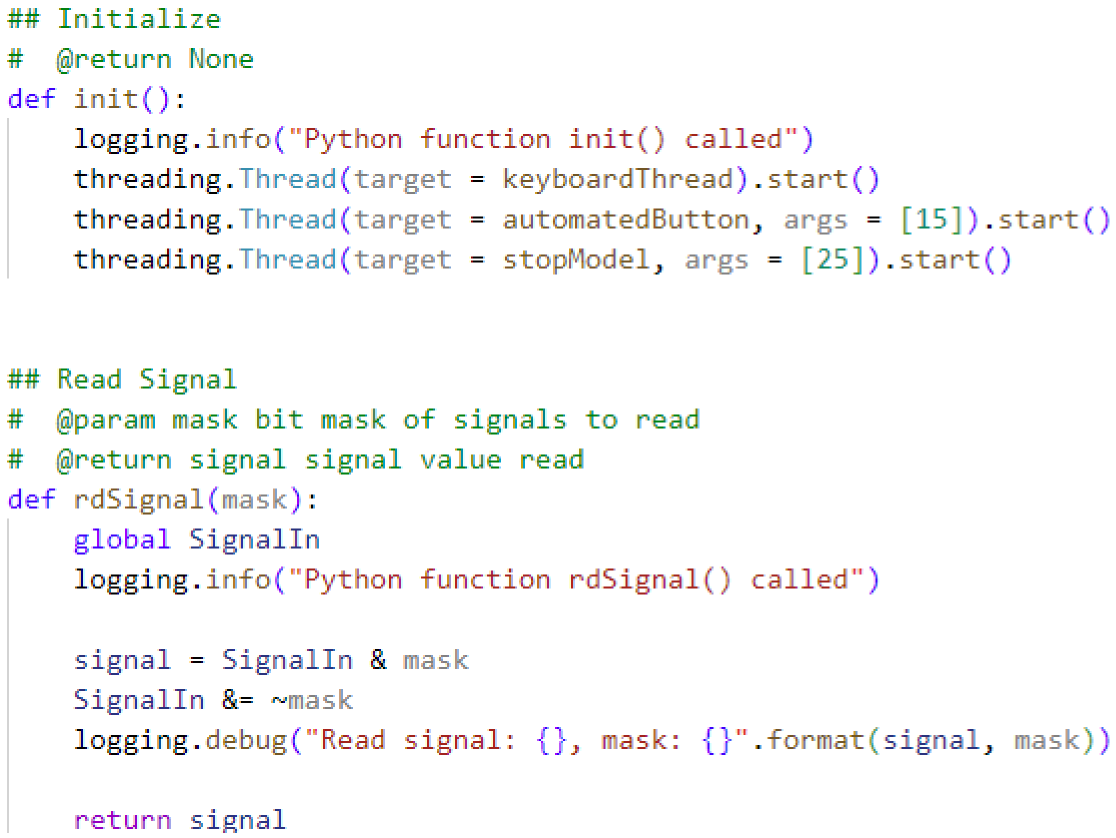
vio_fvp.c (Source Code)



```c
// Get signal input.
uint32_t vioGetSignal (uint32_t mask) {
  uint32_t signal;

  ARM_VIO->SignalIn.mask = mask;
  signal = ARM_VIO->SignalIn.signal;

  vioSignalIn &= ~mask;
  vioSignalIn |=  signal;

  return signal;
}
```

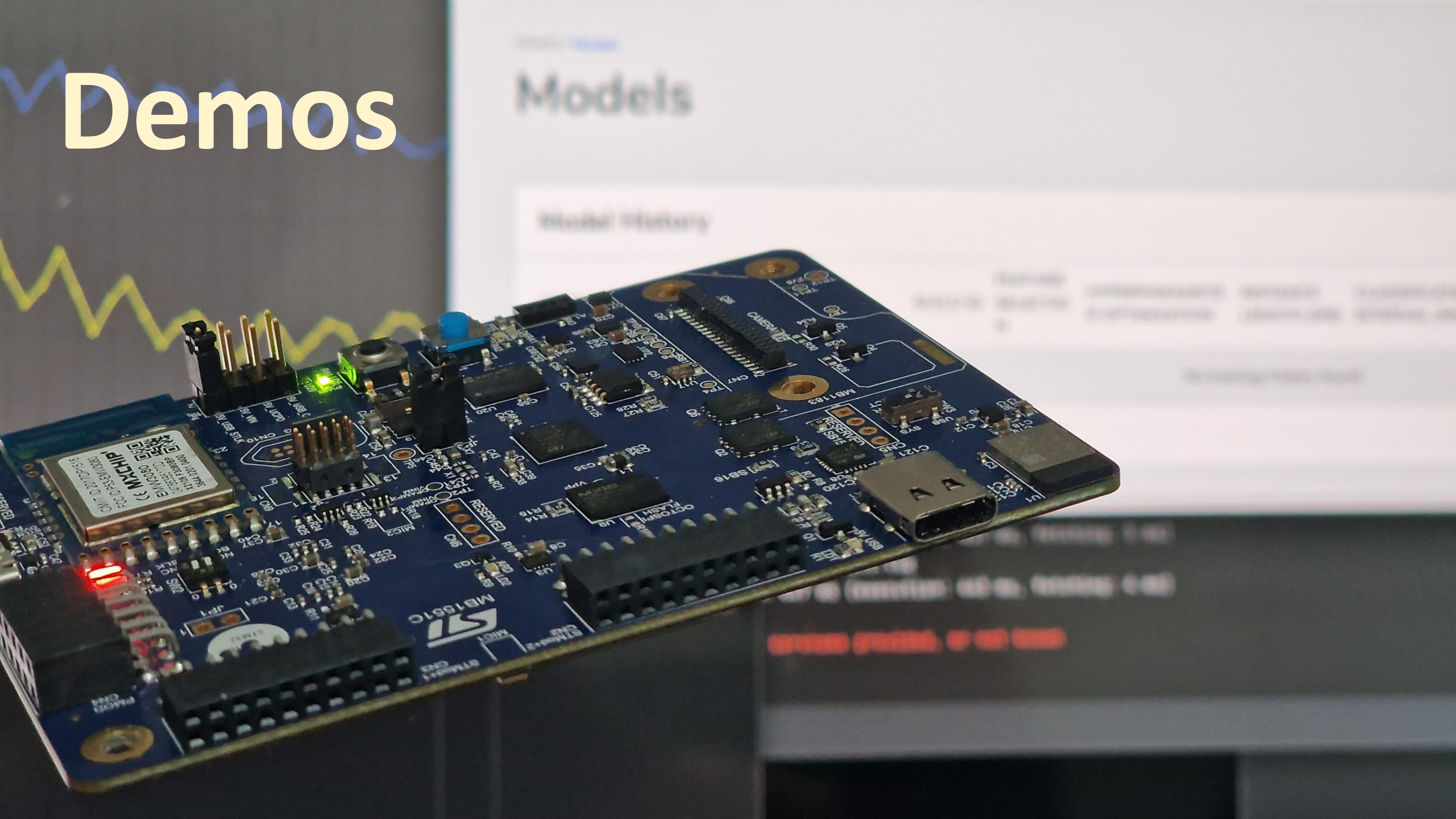
Note: essentially the same blinky example as in BSP-Pack-HandsOn

arm_vio.py (Test Script for AVH)



```python
## Initialize
#  @return None
def init():
    logging.info("Python function init() called")
    threading.Thread(target = keyboardThread).start()
    threading.Thread(target = automatedButton, args = [15]).start()
    threading.Thread(target = stopModel, args = [25]).start()


## Read Signal
#  @param mask bit mask of signals to read
#  @return signal signal value read
def rdSignal(mask):
    global SignalIn
    logging.info("Python function rdSignal() called")

    signal = SignalIn & mask
    SignalIn &= ~mask
    logging.debug("Read signal: {}, mask: {}".format(signal, mask))

    return signal
```

Demos

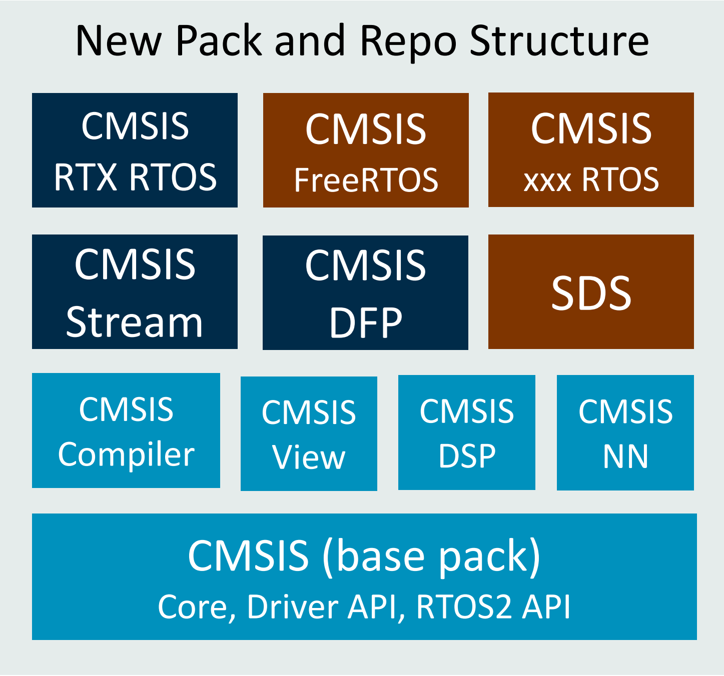

Models
Model History
MXCHIP
MB1551C
CAMERA
OCTOSPI FLASH
RESERVED
MB1183

CMSIS Version 6 - enhancements (compared to 5.9) + timeline

Overall goal: simplify software re-use across supported processors and toolchains

- **Core**: C Startup, new linker scripts (using C header files), fault exception template.
- **Driver**: GPIO for I/O pin control, simplified VIO for LEDs and switches/buttons.
- **RTOS2**: add FuSa RTS API extensions, deprecate TZ handling.
- **Compiler:** I/O retargeting (currently for GCC / AC6)
- **View**: complete initial release.
- **DSP**: incremental improvements in a separate pack.
- **NN**: incremental improvements in a separate pack.
- **Stream**: new component, derived from ComputeGraph (relates to SDS-Framework)
- **DFP:** Generic Device Family Pack for all Cortex-M processors.
- **CMSIS-Toolbox v2.0** feature complete (i.e. with linker script support).

NOTE: **RTOS**: version 1 deprecate and remove.


New Pack and Repo Structure

🟦 NOTE: Tools are no longer included in the CMSIS base pack

May’23 (Beta)

July’23 (Release)

Setup of new structure

Integration tests

Continues development

- CMSIS-DSP and CMSIS-NN already separate
- CMSIS (base pack) is just called CMSIS pack

- Validation with various tools and software integrations

- Further improvements depending on feedback

Actions and Discussion

Closing gaps for seamless operation

- AVH Examples
  - CMSIS-RTOS2 Validation using a build test matrix
  - RTX_Blinky with simplified CMSIS-VIO and native GitHub action workflows.
  - Native GitHub integration

- AVH VSI – Virtual Streaming Interface
  - Audio: github.com/ARM-software/AVH-TFLmicrospeech – shows eventlist tool
  - Sensor: github.com/ARM-software/SDS-Framework - (introduction video here)
  - Video: coming soon
  - VSI is flexible with DMA, IRQ and timer capablities; what other use-cases would be important?

- Please provide feedback so that we can close gaps

Thank You

Danke

Gracias

Grazie

谢谢

ありがとう

Asante

Merci

감사합니다

धन्यवाद

Kiitos

شكرًا

ধন্যবাদ

תודה