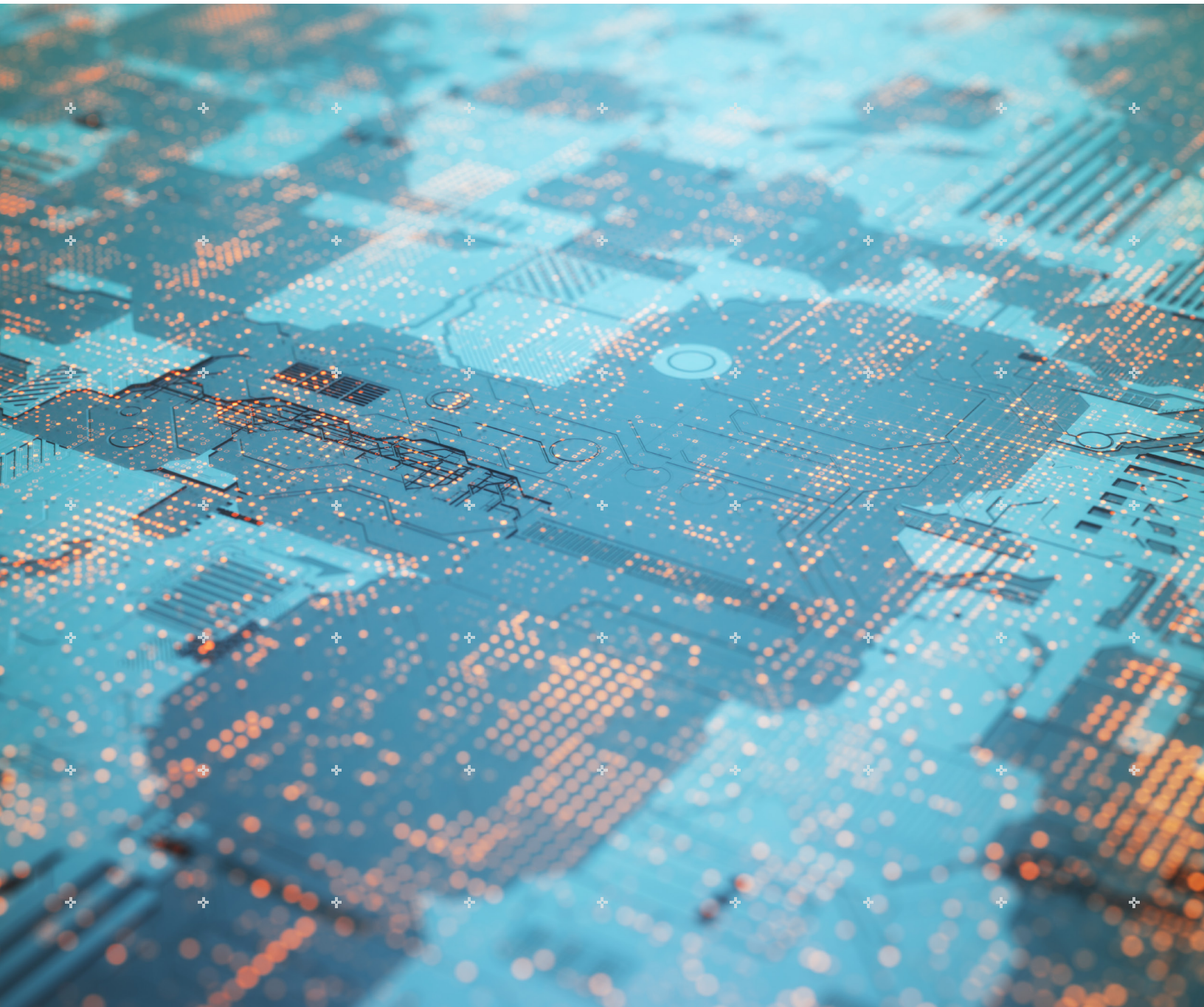
Introduction to Armv8.1-M architecture

arm

By Joseph Yiu, Senior Principal Engineer, Arm

February 2019

White Paper



The Armv8.1-M architecture is an enhancement of the current Armv8-M architecture. It brings many new features, including an M-Profile Vector Extension (MVE) for signal processing and machine learning applications. MVE for the Arm Cortex-M processor series is called Arm Helium technology.

This whitepaper provides an overview on the various enhanced areas in the Armv8.1-M architecture, including Helium.

What is the Armv8.1-M architecture?

Armv8.1-M is an extension of the Armv8-M architecture that includes many new features:

- A new vector instruction set extension called Helium
- Additional instruction set enhancements for loops and branches (Low Overhead Branch Extension)
- Instructions for half precision floating-point support
- Instruction set enhancement for TrustZone management for Floating Point Unit (FPU)
- New memory attribute in the Memory Protection Unit (MPU)
- Enhancements in debug including Performance Monitoring Unit (PMU), Unprivileged Debug Extension, and additional debug support to focus on signal processing application developments
- Reliability, Availability and Serviceability (RAS) extension

Helium:

The M-Profile Vector Extension (MVE) is an optional architectural extension that enables higher signal processing and machine learning capabilities. For Arm Cortex-M processors, MVE is called Helium.

Neon is an architectural extension for A-Profile processors, including Arm Cortex-A and Neoverse, that enables high performance Advanced SIMD technology.

While there are similarities between Helium and Neon, Helium is a new ground-up design that enables efficient signal processing performance in small processors. It offers many new architectural features that enhance the compute performance of embedded use cases, as it is optimized for area and power, bringing Neon capabilities (SIMD instructions for Cortex-A) to the M-Profile architecture.

"The M-Profile Vector Extension (MVE) is an optional architectural extension that enables higher signal processing capabilities. For Arm Cortex-M processors, MVE is called Helium."

Helium and Neon are similar in the following areas:

- 128-bit vector size
- Uses registers in the floating-point unit as vector registers
- Some vector processing instructions are available in both Helium and Neon

However, there are many key differences between Helium and Neon:

- Helium is designed to maximize the use of all available hardware and uses fewer vector registers than Neon
- Some operations in Helium utilize both vector registers and registers in the scalar register bank
- Helium supports more data types than Neon
- Helium supports many new features like loop predication, lane predication, complex maths and scatter-gather memory accesses

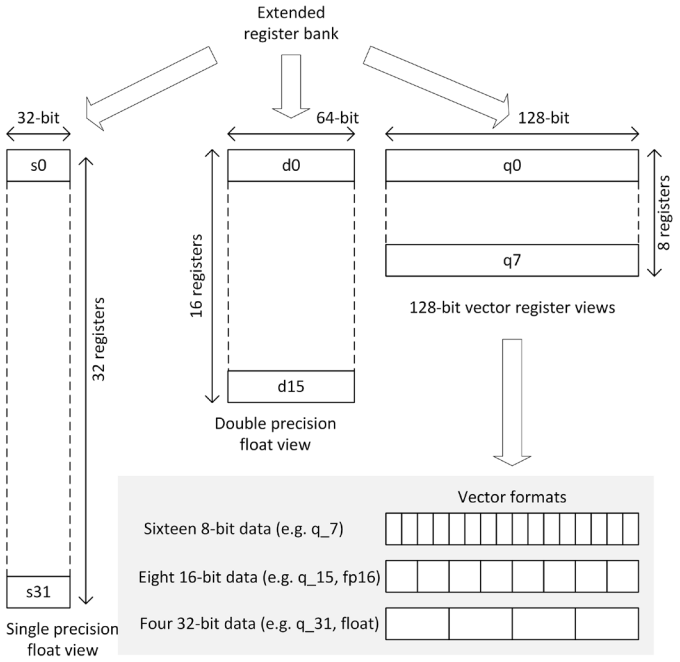
Helium adds over 150 new scalar and vector instructions. Architecturally, there are many implementation options:

- Helium option omitted – Armv8.1-M integer core with optional scalar FPU (double precision support also optional)
- Helium with support for vectored integer only, and with optional scalar FPU (double precision support also optional)
- Helium with vectored Integer + floating point (support vectored single precision and half precision) with scalar FPU (double precision support also optional)

Due to large number of instructions covered by Helium, it is impossible to cover all of them in this document. Here are a number of highlights, in addition to the vector processing capabilities:

- Interleaving and de-interleaving load and store instructions (VLD2/VST2 with strides of 2, and VLD4/VST4 with strides of 4)
- Vector gather load and vector scatter store – memory access of elements in a vector register, with address offset of each element in the vector, defined using elements in another vector register. This allows software to handle arbitrary memory access patterns and can be used to emulate special addressing modes like circular addressing, which are often used in signal processing. This can also help accelerate non-sequential accesses of data elements in arrays in various data processing tasks
- Vector complex value processing supporting integers (8-bit, 16-bit and 32-bit) and float (32-bit) – e.g. VCADD, VCMUL, VCMLA instructions
- Lane predication – this will be covered later
- Big integer support – this will be covered later

To reduce the processor's area and power, the register bank in the FPU is reused for vector processing:



The integer support in Helium enables efficient compute of 8-bit, 16-bit and 32-bit fixed point data. 16-bit and 32-bit fixed point formats are widely used in traditional signal processing applications, such as audio processing. 8-bit fixed point format can be important to machine learning (ML) processing, such as neural network computation, as well as image processing.

Combining Helium with the new Low Overhead Branch Extension, the performance of signal processing with Armv8.1-M processors can be several folds better than using traditional SIMD instruction set in Armv7-M and Armv8.0-M, while maintaining a small processor size with excellent energy-efficiency.

Helium enables Arm Cortex-M processors to address the compute challenges in (but not limited to):

- Audio devices
- Sensor hubs (sensor fusion), context hub (environmental sensing) and wearables
- Keyword spotting and voice command control
- Power electronics and controls (signal processing)
- Communications (e.g. NB-IoT)
- Still image processing (e.g. camera)

"With Helium, many applications that previously used a Cortex-M processor with a dedicated DSP could consolidate the two processor systems into a single processor."

With Helium, many applications that previously used a Cortex-M processor with a dedicated DSP could consolidate the two processor systems into a single processor. This has many advantages:

- Simplify software development – only requires a single toolchain, single architecture and reduces software overhead for communications between the two processor systems
- Reduce silicon design complexity – only requires a single memory system, hence enabling faster system-on-chip (SoC) design cycle and reducing costs

It is also possible to replace some of the legacy standalone DSP products in embedded systems:

- Cortex-M processors are highly versatile and can perform general non-DSP workloads at higher performance than many legacy DSPs
- High code density of Cortex-M processors allows system cost reduction in such DSP replacements

Helium features – lane predication:

Helium includes MVE features for supporting conditional execution for each of the lanes in the vector. This optimization mechanism is called lane predication. To enable this feature, a new special purpose register called Vector Predication Status and Control Register (VPR) is added, to hold the condition for each lane. This predication condition inside this register is updated by vector operations, such as Vector Compare (VCMP). After setting up the condition flags, VPT (Vector Predicate Then) / VPST (Vector Predicate Set Then) instructions can then set up conditional execution of each lane, in sub-sequence vector instructions (up to 4 instructions in vector predication block, similar to the IF-THEN instruction block).

The status of the VPR is saved and restored automatically in exception events, using a reserved word inside the extended exception stack frame.

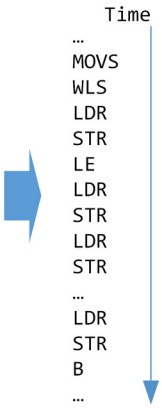
Helium features – big integer support:

To get the most out of the 128-bit vector registers, Helium also introduced big integer processing instructions (VADC, VSBC and VSHLC) as defined in MVE, which can be chained to operate with integer data types of 128 bits or larger.

Low Overhead Branch Extension:

To enable efficient signal processing, Armv8.1-M introduced low overhead loops and additional branch instructions. A simple while loop structure starts with a WLS (while-loop-start) instruction, which specifies the loop count and branch back address, and the loop structure ends with a LE (loop-end) instruction.

```
    MOVS   R0, #10 ; loop 10 times
    WLS    LR, R0, loopend_1 ; while loop start
loopstart_1
    LDR    R1, [R2], #4 ; Memory copy
    STR    R1, [R3], #4 ; Memory copy
    LE     LR, loopstart_1   ; loop end
loopend_1
    B      label_x ; branch to somewhere else
```



The first time the loop is executed, both WLS and LE instructions would be executed, and the loop addresses are cached inside the processor, and subsequent loop iterations do not need to execute these two instructions again. The loop counting is handled by the link register (LR/r14) and the loop exit, when the loop counter reaches zero.

If an interrupt occurred during the low-overhead-loop, the loop address cache would be cleared and the LE instruction would be re-executed after returning from the interrupt service routine (ISR).

In addition to while-loop-start (WLS), there is also a do-loop-start (DLS) instruction, which is similar. The do-loop always executes the first iteration of the loop body, but while-loop will jump to the end of the loop before the first iteration, if the condition is not met.

There is a variant of low-overhead-loop instructions (WLSTP and DLSTP) which enables loop tail predication – if a data processing task needs to be performed on N elements, where N is not a multiple of vector lane width (e.g. if the elements to be processed are 32-bit, Helium can process 4 lanes of elements per vector instruction), then loop tail predication allows the last loop iteration to process just the remaining elements using conditional execution mechanism. When using WLSTP/DLSTP, LETP (Loop-end with Tail Predication) must be used to indicate end of the loop. A suffix is needed for WLSTP and DLSTP (.8/.16/.32) to indicate the size of the vector elements to be processed, and the value in the LR holds the number of elements to process, rather than the number of loops.

"By moving processing from single precision to half precision, the processor can process twice the amount of data over the same duration."

An additional instruction called LCTP (Loop clear with tail predication) allows the low overhead loop with tail predication to be terminated early, if required.

WLS, DLS, LE instructions are available regardless of whether Helium is implemented. Loop tail predication instructions (WLSTP, DLSTP, LETP) require Helium option. There are additional branch hint instructions, introduced in the Armv8.1-M architecture, which take advantage of the hardware introduced by low overhead loop, to enable better branching performance.

Other Armv8.1-M mainline extension enhancements:

Armv8.1-M introduces a number of conditional execution instructions:

- CINC – conditional increment
- CINV – conditional invert
- CNEG – conditional negate
- CSEL – conditional select
- CSET – conditional set register to 1
- CSETM – conditionally set all bits in register to 1
- CSINC – conditional select / increment
- CSINV – conditional select / bitwise invert
- CSNEG – conditional select / negate

Armv8.1-M also adds:

- 64-bit arithmetic and logical shift instructions (ASRL, LSLL, LSRL + saturated / rounded variants), halving variants for some of these instructions are also available
- Signed and unsigned rounding/saturating shift instructions for 32-bit and 64-bit data
- These are available only when Helium is implemented.

Floating point processing enhancements:

Just like the Armv8.0-M architecture, Armv8.1-M supports optional scalar single precision (32-bit) and double precision (64-bit) floating point computation (supports all FPv5 instructions). In addition, Armv8.1-M also supports:

- Optional scalar half precision (16-bit) floating point
- Optional vector half precision (16-bit) floating point (part of Helium)
- Optional vector single precision (32-bit) floating point (part of Helium)

Half precision floating point support can be useful for audio pre-processing for keyword spotting and voice command control applications. In these applications, the audio does not need to have very high resolution, but good dynamic range support is highly desirable. By moving processing from single precision to half precision, the processor can process twice the amount of data over the same duration, using Helium technology. Use of half precision floating point format can also help reduce memory size requirements for data (e.g. filter coefficients).

Security-related enhancements – FPU context saving/restore:

Armv8.0-M introduced TrustZone security extension, which enabled new generations of security solutions to be implemented on low cost, low power embedded systems, using Cortex-M processors. One of the key characteristics of TrustZone for Armv8-M is that it allows efficient direct function calls between Secure software (protected environment) and Non-secure software (Normal environment), and Armv8.1-M continues to add enhancement in this area.

To support TrustZone for Armv8-M, Arm C Language Extension (ACLE) defined various C compiler features required, namely Cortex-M Security Extension (CMSE). One of the requirements is that when Non-secure software calls a Secure API, the Secure function epilogue (code insert by C compiler at the end of the Secure API) needs to clear the contents in the Floating Point Status and Control Register (FPSCR) to avoid information leaking to the Non-secure side. This is good for security but can also cause a problem for Non-secure software, with regard to ABI compatibility – FPU configurations could be changed (e.g. rounding mode configuration).

In Armv8.1-M, the instruction set enhancements are added to enable context saving of Non-secure FPSCR states (FPCXT_NS) in the prologue of the Secure API and context restore in the epilogue. The enhanced instructions include FPU memory access instructions (VLDR, VSTR) and FPU register access instructions (VMRS, VMSR). A C compiler update is required to utilize this new feature. In relation to this, Armv8.1-M also adds new instruction for clearing context in the register bank (CLRM and VSCCLRM).

Security related enhancements – MPU's Privileged eXecute Never (PXN) attribute:

Another security enhancement is inside the Memory Protection Unit (MPU) – a new MPU region attribute called Privileged eXecute Never (PXN). If an MPU region has PXN attribute set and the processor attempts to execute the code inside with privileged level, the Memory Management Fault exception would be triggered, with IACCVIOL bit in MemManage Fault State Register set to 1.

The PXN attribute bit is in bit 4 of MPU_RLAR (Region Limit Address Register) and its alias registers. It is available in both Secure and Non-secure MPU, and this bit was previously fixed to 0 in Armv8.0-M.

The PXN feature allows privileged software to ensure specific application tasks (threads) to execute in unprivileged level only. For example, a hacker cannot use stack corruption in a privileged peripheral handler to branch into unprivileged codes and execute them with privileged level.

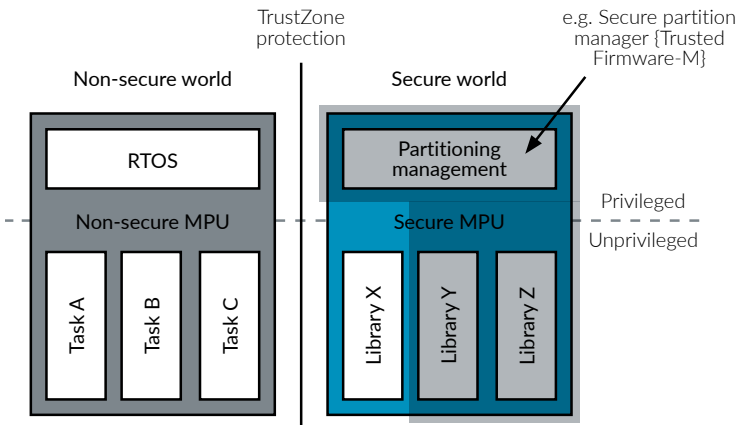
This feature is also particularly useful for TrustZone-enabled systems with secure firmware components from various software vendors. In those cases, some of the security firmware components might not be fully trusted and need to be restricted to unprivileged execution only. If such systems are implemented with Armv8.0-M, the unprivileged software components must not be allowed to have its own secure entry points which are callable from Non-secure state, because the software components would execute in a privileged state if being called directly from Non-secure Handler mode. As a result, the entry points need to be implemented separately with security checking, which increases software overhead. With the PXN attribute available in Armv8.1-M, these unprivileged software components can have their own Secure API entry points, and only if the APIs are called by Non-secure handlers, then the MemManage fault exception handler can intercept and switch the processor to unprivileged state for the Secure APIs to be executed.

Security-related enhancements – Unprivileged debug extension:

For systems with Secure software libraries from third parties, there are situations where a software developer might need to debug the unprivileged software library, but he/she might not be fully trusted by the vendors of other Secure firmware components. In Armv8.0-M, if Secure debug is enabled, then the software developer can have full debug access to both privileged and unprivileged software, which is undesirable in this case. While it is possible to restrict debug access (instead of providing full Secure debug access via debug access port, Secure privileged software can use debug monitor with a communication channel, such as CoreSight SDC-600, to provide restricted debug access to Secure unprivileged world), this requires more software overhead.

Armv8.1-M provides a new mode of debug enabling mechanism through the unprivileged debug extension. When Secure debug is disabled, Secure privileged software can enable the unprivileged debug extension via the Debug Authentication Control Register (DAUTHCTRL).

When dealing with multiple Secure firmware libraries, the Secure privileged software, that deals with context switching in Secure world (e.g. Secure Partition manager in the Arm Platform Security Architecture ‘PSA’), should program DAUTHCTRL registers when switching between different contexts. For example, in the figure below, the software developer has debug access to all Non-secure software and Library X, which is Secure unprivileged. Halting request can be accepted when the processor is running the software, but not possible when running the Library manager or Library Y and Z.



With the unprivileged debug extension, debug access to memory of current security state (except certain debug components inside the processor) is blocked when the processor is running and is allowed when the processor is halted in the unprivileged state, with unprivileged debug enabled (UIDEN bit in DAUTHCTRL set to 1). The debug accesses also check against the permission in the MPU of the current state, when unprivileged debug is used for that state.

Based on the example in the diagram:

- Software developers can access the Non-secure memory and halt the processor when it is in a Non-secure state
- Software developers can halt the processor only when the processor is running Library X (DAUTHCTRL configure by library manager in context switches) and can access memory space that is accessible by Library X (permission based on Secure MPU)

The unprivileged debug extension is available for both the Secure and Non-secure world (the UIDEN and UIDAPEN bits in DAUTHCTRL are banked between security states). To support unprivileged debug, the debugger tools and the library manager need to be updated. However, the unprivileged debug is an extension of the debug architecture and existing debug functionalities (when legacy authentication control signals are enabled) are not affected.

Debug enhancements – Performance Monitoring Unit (PMU):

In Armv8.1-M, the profiling counters in Data Watchpoint and Trace (DWT) are extended to support Performance Monitoring Unit (PMU) features, as found in Cortex-A processors. This enables advanced profiling features, including the capabilities to analyze cache hit/miss. The previous profiling features are still available, but for software developers to make the most out of the architecture, debuggers need to be updated.

"Several debug architecture enhancements are also included in Armv8.1-M that further enhance debug efficiency for signal processing applications."

Please note that the PMU uses a separated address space from DWT, although the profiling counters are physically the same (address aliasing) and hence, debug tools cannot use PMU and legacy DWT profiling at the same time.

Debug enhancements – features for signal processing applications:

Several debug architecture enhancements are also included in Armv8.1-M that further enhance debug efficiency for signal processing applications. These include:

- Data watchpoint with bit mask for value matching. Useful for detecting signal ranges
- Breakpoint with counter – breakpoint triggers (halt) only when certain count value is reached. This can be useful to halt a processor when a digital filter is stabilized

Reliability, Availability and Serviceability (RAS) Extension:

The RAS extension was introduced in Cortex-A processors in Armv8-A architecture. RAS are three aspects of the dependability of a system:

- Reliability, continuity of correct service
- Availability, readiness for correct service
- Serviceability, ability to undergo modifications and repairs

The need for RAS extension was originated from server/enterprise applications – RAS techniques reduce unplanned outages because:

- Transient errors can be detected and corrected before they cause application or system failure
- Failing components can be identified and replaced
- Failure can be predicted ahead of time, to allow replacement during planned maintenance

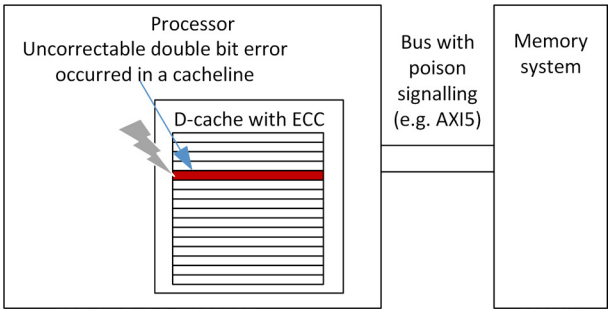
While Cortex-M processors are mostly deployed in embedded and automotive applications, the functional safety requirements for automotive and industrial usages make RAS a natural choice for the next generation Cortex-M processors, as many techniques for RAS also help functional safety aspects.

In processors designed with Armv8.1-M, the RAS extension covers:

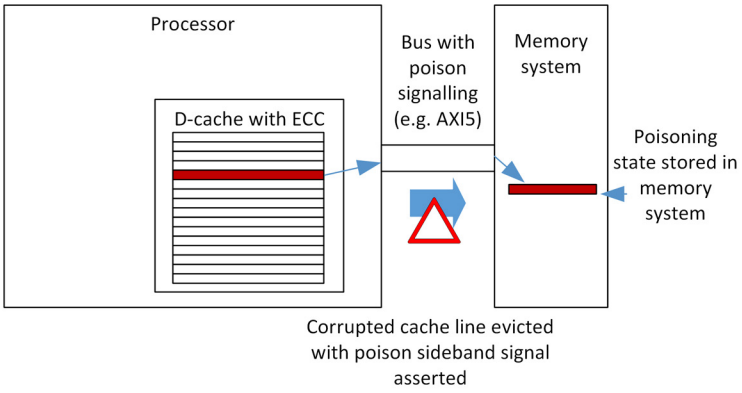
- Adding of Error Synchronization Barrier (ESB) instruction
- Error reporting registers (e.g. for Error Corrected Code errors in cache)
- Bus interface level enhancements, such as parity or Error Correction Code (ECC) signals for bus transfer integrity checks, and poison signalling in bus interface

The poison signalling feature is new to embedded processors. One common usage example is for handling of corrupted data in data cache:

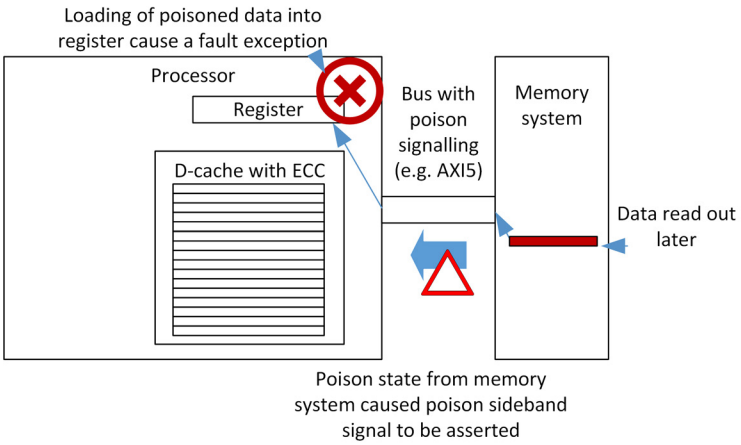
1) A cache line is corrupted (e.g. due to voltage instability). If the error is more than one bit, ECC will not be able to correct the error when the data in the corrupted cache line is used.



2) Later, a cache line eviction occurred, and this flushed the corrupted data. ECC error is detected at this stage and triggers the poisoning side band signal on the bus to be asserted. The poisoning state is stored in the memory system. No fault exception occurred at this stage.



3) When the data (with poison state) is read by a processor (could be a different processor or a different bus master in the system), the poison information is passed along and triggers a fault exception at the processor.



This approach has several advantages:

- If the corrupted data is not used by any bus master (e.g. gets overwritten by other data later), there is no need to trigger a fault exception
- The fault exception is triggered when the data is being used and is synchronous to the applications being affected. If poisoning signaling is not available, the exception needs to be triggered at cache line eviction and the processor can be running a completely unrelated application, making it hard to decide which applications are being affected

The use of RAS extension also enables better consistency between Cortex-M and Cortex-A processors, which is an advantage for software that handles error conditions in heterogeneous multi-processor systems with Cortex-A and Cortex-M processors.

Migration of software to Armv8.1-M:

Armv8.1-M processors retain the previous key characteristics of Arm Cortex-M processors including:

- Ease-of-use
- Fast and deterministic interrupt response time
- Optimized for low power embedded applications

Existing Armv8-M software can run on Armv8.1-M, to enable easy software migration. Just like Armv8.0-M, Armv8.1-M supports the TrustZone security extension, which addresses security requirements for connected products, such as IoT devices.

"The Helium ecosystem has a wide range of tools and software libraries available, enabling rapid and low risk development."

To take advantage of new features, various software related updates would be needed:

- C compilers – compiler (and binutils) need to be updated to support new instructions. The Arm C language extension (ACLE) needs to introduce new intrinsic functions for new instructions. The prologue and epilogue of Secure APIs should also be enhanced to enable saving and restoring of Non-secure FPCXT. Compiler vendors can also add advanced optimizations, to take advantage of the low overhead loops, new instructions, and potentially support auto-vectorization on Helium capable systems
- Debug tools – Debuggers require updates to support new features and new architectural registers (e.g. VPR)
- RTOS – To take advantage of new security features, like new MPU attributes, OS needs to be updated. Additional updates are also needed to support unprivileged debug, if this feature is required for the device
- Arm Trusted Firmware for Armv8-M – this needs to be updated to support new PXN MPU attribute and unprivileged debug extension
- Application code – New CMSIS-CORE headers and CMSIS-DSP libraries will be available. The enhanced CMSIS-DSP library will take advantage of the Helium instruction set, to provide significant performance uplift

Summary

Armv8.1-M includes many new enhancements and advantages. A major addition to Armv8.1-M is the M-Profile Vector Extension (MVE). Helium is the M-Profile Vector Extension for the Arm Cortex-M processor series, which enables efficient signal processing and machine learning in small, embedded applications. It allows consolidation of what would previously require two processors onto one SoC, reducing complexity and cost.

Armv8.1-M includes other enhancements to the instruction set, debug features and improves the dependability of embedded systems. Also, the architecture has built-in security with TrustZone and is designed to Platform Security Architecture (PSA) specifications.

The Helium ecosystem has a wide range of tools and software libraries available to enable performance increases for DSP and ML applications. Developers can get started today with pre-silicon software development and migrate libraries and other code to Helium. Tools with support include the Arm Fast Model virtual platform, Arm Compiler 6, software libraries, and toolchains such as Arm Development Studio and Keil MDK.

Get in touch to get access to the Arm tools supporting Helium.

arm