Appendix

Cortex-M Architecture in a Nutshell

1. General Properties

General-purpose registers. A Cortex-M MCU has 16 general-purpose registers, i.e., `R0` to `R12`, `R13/SP` (stack pointer), `R14/LR` (link register), and `R15/PC` (program counter). Depending on the *execution mode* and `CONTROL` register, `SP` shadows the main stack pointer (`MSP`) or the process stack pointer (`PSP`). In Armv8-M, `MSPLIM` and `PSPLIM` hold the lower limit of the corresponding stack pointer. `LR` holds the return address for a subroutine or a special value `EXC_RETURN` when an interrupt occurs.

Execution mode, privilege level, and special registers. Cortex-M MCUs have thread and handler execution modes. They, with the exception of M0/M0+/M1, also have privileged and unprivileged levels. Execution in the thread mode can be either privileged or unprivileged, whereas the handler mode is always privileged. The current execution mode and privilege level are determined by the combination of the interrupt program status register (`IPSR`) and the `CONTROL` register. To switch from privileged mode to unprivileged, one can simply set `CONTROL[0]` to 1 using the move-to-system-register (`MSR`) instruction. To switch from unprivileged to privileged, the software uses the supervisor call (`SVC`) instruction.

Physical memory map. The physical memory space of a Cortex-M MCU is 32-bit, which is equally divided into eight regions. The first region is reserved for *code*. The second region is mapped to on-chip *SRAM*. The *peripheral* and two *device* regions map I/O device registers. The *system* region maps system control registers, including *private peripheral bus* (PPB) for memory protection unit (MPU), security attribution unit (SAU) in Armv8-M, SysTick timer, etc.

Automatic stacking and unstacking. When a higher-priority interrupt or exception occurs, the MCU automatically pushes eight registers, i.e., `xPSR`, `PC`, `LR`, `R12`, `R3`, `R2`, `R1`, and `R0`, to the current stack. Then, the MCU stores a special value `EXC_RETURN` to the `LR` register, and executes the interrupt service routine (ISR). When the ISR returns, the MCU automatically performs unstacking to restore the context. This is triggered by copying `EXC_RETURN` to the `PC` register.

Memory protection. The memory protection unit (MPU) regulates access permissions to software-designated physical regions based on the privilege level. Cortex-M55 MPU additionally introduces privileged execute never (`PXN`). Most processors only support up to 16 MPU regions.

Unprivileged memory access instructions. The v7-M instruction set introduces several special store and load instructions (`STR*T/LDR*T`). Even when these instructions are executed at the privileged level, they are treated as if they were executed at the unprivileged level.

Debug units. Cortex-M comes with many debug and trace units such as flash patch and breakpoint unit (FPB), data watchpoint and trace unit (DWT), micro trace buffer (MTB), and embedded trace macrocell (ETM). These units are restricted to the software at the privileged level and external hardware debug adapters. They can perform instruction and data tracing, system events profiling, etc.

TABLE 6. CORTEX-M MCUS AND THEIR SECURITY-RELATED FEATURES

CPU	ISA	Max MPU Regions	TrustZone-M	Max SAU Regions	PACBTI	PXN	XOM	MSPLIM/PSPLIM	UP Store/Load Ins.	Branch Prediction	ETM	MTB	DWT	FPB	Year Introduced
M0 [175]	v6-M [5]												●	⦿	2009
M1 [176]	v6-M												⦿	⦿	2007
M0+ [177]	v6-M	8										●	●	⦿	2012
M3 [178]	v7-M [6]	8							●	●	⦿		●	●	2004
M4 [179]	v7-M	8							●		⦿		●	●	2010
M7 [180]	v7-M	16							●	●	●		●	●	2014
M23 [10]	v8-M Baseline [7]	16	●	8				●	●		⦿	●	●	●	2016
M33 [11]	v8-M Mainline [7]	16	●	8				●	●		⦿	●	●	⦿	2016
M35P [181]	v8-M Mainline	16	●	8				●	●		⦿	●	●	●	2018
M55 [12]	v8.1-M Mainline [7]	16	●	8	●	●	●	●	●	●	⦿		●	⦿	2020
M85 [182]	v8.1-M Mainline	16	●	8	●	●	●	●	●	●	⦿		●	⦿	2022

Whitespace: Not support, ●: Support, ⦿: Partially Support. On M23, the ETM and the MTB are exclusive to each other.

2. Security Extensions

Secure and non-secure state. With the Armv8-M TrustZone-M extension (TrustZone-M), the MCU runs either in the secure state or non-secure state. In the non-secure state, the processor can only access non-secure resources, whereas in the secure state, the processor can access all resources. Different from Cortex-A TrustZone (TrustZone-A) [183] where the selection of the effective execution state is configured in the secure configuration register, the division of states in Cortex-M is based on memory regions and the transition between them is regulated by newly introduced instructions (e.g., `SG`). Some special registers and system peripherals are banked for different security states, e.g., `MSP`, `PSP`, `CONTROL[1-0]`.

Secure memory partitioning. A memory region can be secure, non-secure callable (NSC), or non-secure. An NSC region is a special region that serves as a springboard from non-secure regions to secure regions using the `SG` instruction. The security assignment of a memory region is controlled by the combination of the internal secure attribution unit (SAU) and a vendor-specific implementation-defined attribution unit (IDAU). SAU can configure up to 8 regions, whereas IDAU can't be configured after boot.

Memory aliasing, memory protection controller (MPC), and peripheral protection controller (PPC). Memory aliasing, enabled by the MPC, maps shared resources between the secure and non-secure states. Configuring the MPC at run-time can dynamically control which state a memory block is assigned to. Similar to an MPC, a PPC can also be added on paths to peripherals to provide security gating.

Synchronous state switching. In contrast with Cortex-A processors, where there is a single entry point, i.e., secure monitor call, for entering or leaving the secure state, Cortex-M supports fast state switches with an unlimited number of entries. To switch from the non-secure state to the secure state, non-secure software uses the normal branch with link (`BL`) instruction to call into a secure gateway (`SG`) instruction in an NSC region. Afterwards, the processor state is changed to secure. To switch from the secure state to the non-secure state, the software uses the branch with link and exchange non-secure (`BLXNS`) instruction or branch and exchange non-secure (`BXNS`) instruction.

Pointer authentication and branch target identification (PACBTI). The v8.1-M instruction set introduces the PACBTI extension [14]. Pointer authentication code (PAC) authenticates pointers at run-time, and branch target identification (BTI) prevents code reuse attacks by constraining indirect branch targets to BTI clearing instructions. While Cortex-A adopts the unused most significant bits of a 64-bit pointer to embed the PAC, Cortex-M uses a 32-bit PAC and stores it in a general-purpose register. Cortex-M has four PAC keys, one for each security state and privilege level.

A Suite of Example Cortex-M Code

We developed a suite of example Cortex-M code to help researchers easily access Cortex-M research. The suite is tested on the Cortex-M33 based Armv8 IoT Kit FVP [184], which can be used on the Cortex-M Prototyping System (MPS2+)-based Field Programmable Gate Array (FPGA) [185] prototype board. We will open-source and keep developing the suite after the paper is accepted. To show the good faith of open-source, we put an anonymized copy of the source code online for artifact evaluations[3]. The example code suite includes the following components:

1) Helper and configuration functions. These functions provide a C language interface to configure system and peripheral registers. The current version include: i) exception handlers that retrieve detailed fault information; ii) functions to enable and disable data and instruction cache; iii) stack dump function that prints out regular stack and exception stack contents; helper functions to configure and control iv) MPU; v) SAU; vi) MPC; vii) DWT; viii) MTB.

2) Test cases to demonstrate how to use hardware features. The current version include the following test cases: i) change `xPSR`; ii) legal and illegal memory accesses; iii) `TT` instruction examples; iv) stacking and unstacking with and without TrustZone; v) instruction tracing with MTB; vi) performance measurement and data access monitor with DWT; vii) state transition tests and performance measurement.

3. https://anonymous.4open.science/r/code-SoK-Cortex-M