Explanation of Crypto Engine on Raspberry Pi 5

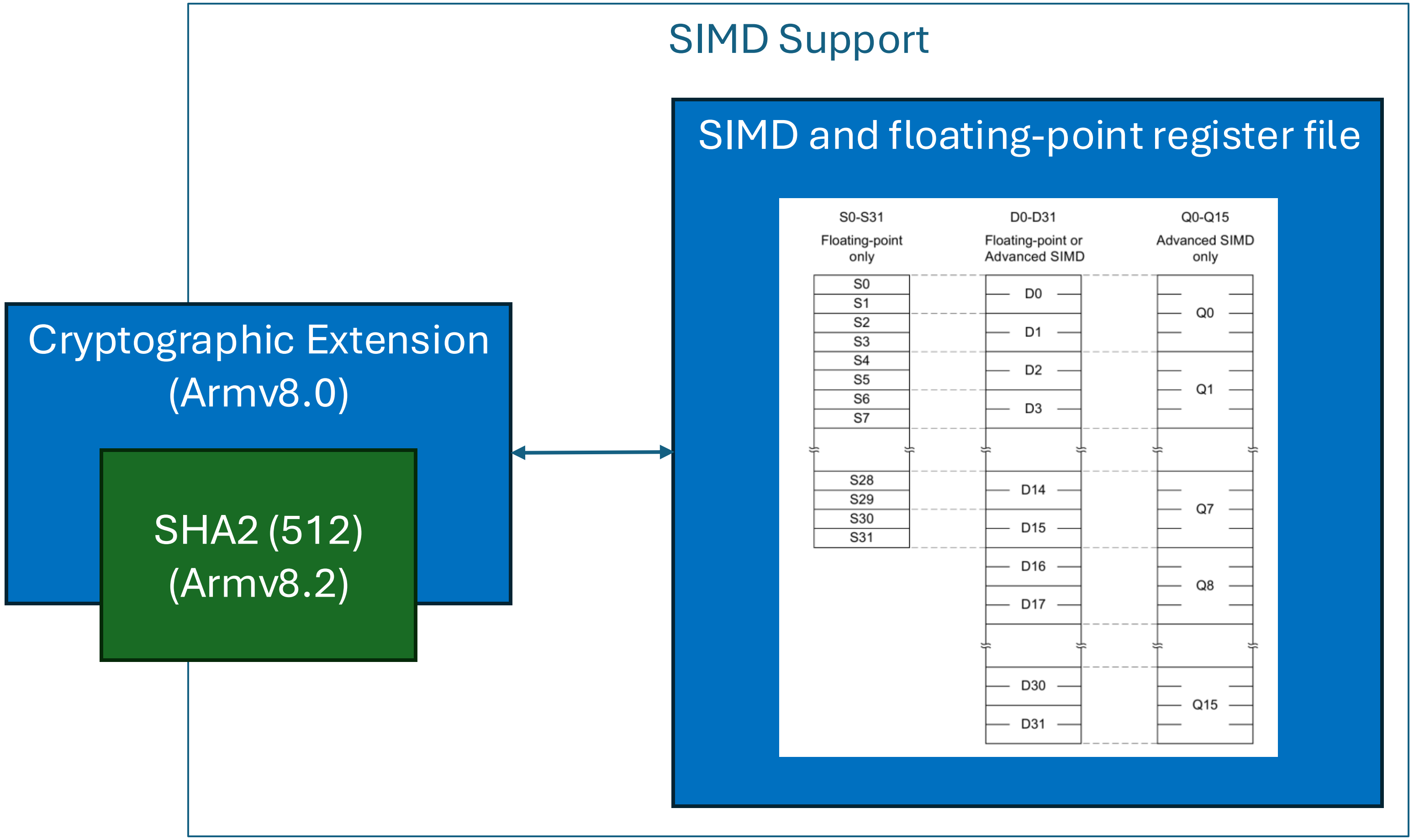
Joshua Deckman and Jack Pearson

Cryptographic Extensions on the Pi 5

The Pi 5’s processor implements the Armv8.0 cryptographic extension to SIMD support.

In addition to the base cryptographic extension, the processor also implements the Armv8.2 SHA2 (SHA512) extension.

The SIMD and floating-point register file is different from the general-purpose registers and can be viewed as single word, doubleword, or quadword registers, depending on the task.



SIMD Support

SIMD and floating-point register file

Cryptographic Extension (Armv8.0)

SHA2 (512) (Armv8.2)

Supported Instructions

Armv8.0 Cryptographic Extension

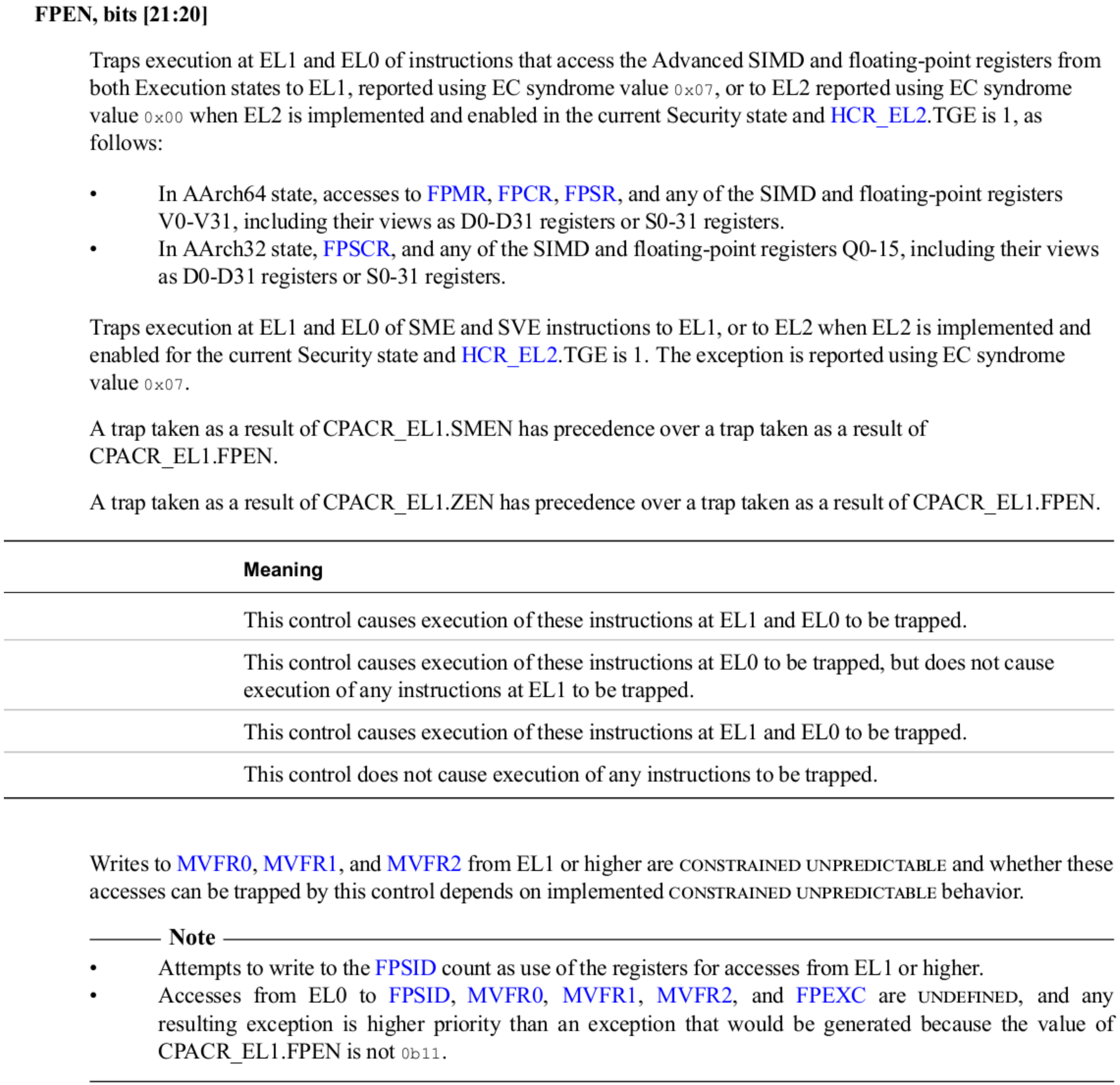
• AESD:	AES single round decryption
• AESE:	AES single round encryption
• AESIMC:	AES inverse mix columns
• AESMC:	AES mix columns
• PMULL:	Polynomial multiply long
• PMULL2:	Polynomial multiply long
• SHA1C:	SHA1 hash update (choose)
• SHA1H:	SHA1 fixed rotate
• SHA1M:	SHA1 hash update (majority)
• SHA1P:	SHA1 hash update (parity)
• SHA1SU0:	SHA1 schedule update 0
• SHA1SU1:	SHA1 schedule update 1
• SHA256H:	SHA256 hash update, part 1
• SHA256H2:	SHA256 hash update, part 2
• SHA256SU0:	SHA256 schedule update 0
• SHA256SU1:	SHA256 schedule update 1

Armv8.2 SHA2 Extension (FEAT_SHA512)

• SHA512H:	SHA512 hash update part 1
• SHA512H2:	SHA hash update part 2
• SHA512SU0:	SHA512 schedule update 0
• SHA512SU1:	SHA512 schedule update 1

Permissions

- The capability to run these SIMD instructions in userspace (EL0) can be set by kernel by setting CPTR_EL2.FPEN or CPACR_EL1.FPEN flags
    - This bit is set in default kernel configuration



FPEN, bits [21:20]

Traps execution at EL1 and EL0 of instructions that access the Advanced SIMD and floating-point registers from both Execution states to EL1, reported using EC syndrome value `0x07`, or to EL2 reported using EC syndrome value `0x00` when EL2 is implemented and enabled in the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to FPMR, FPCR, FPSR, and any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers.
- In AArch32 state, FPSCR, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers.

Traps execution at EL1 and EL0 of SME and SVE instructions to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The exception is reported using EC syndrome value `0x07`.

A trap taken as a result of CPACR_EL1.SMEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

A trap taken as a result of CPACR_EL1.ZEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

	Meaning
	This control causes execution of these instructions at EL1 and EL0 to be trapped.
	This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped.
	This control causes execution of these instructions at EL1 and EL0 to be trapped.
	This control does not cause execution of any instructions to be trapped.

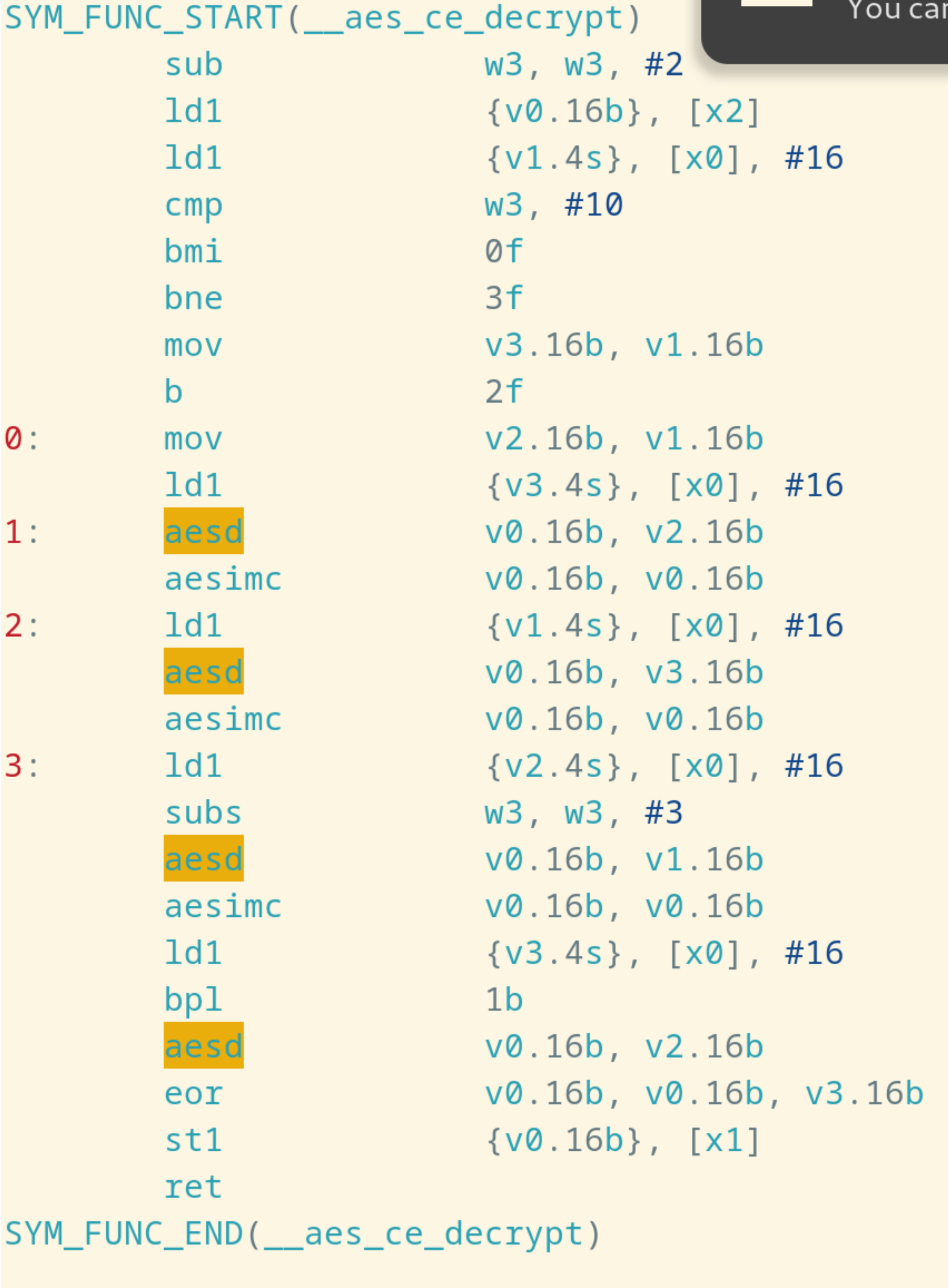
Writes to MVFR0, MVFR1, and MVFR2 from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

Note

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to FPSID, MVFR0, MVFR1, MVFR2, and FPEXC are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.FPEN is not `0b11`.

The Kernel APIs

- Linux kernel provides prewritten "glue" of the extended instructions
- Kernel implementation is accessible on Netlink socket in AF_ALG family or node in /dev/crypto with cryptodev module
- In practice usually AF_ALG option is leveraged via libkcapi library
- In practice cryptodev option is leveraged via cryptodev.h header

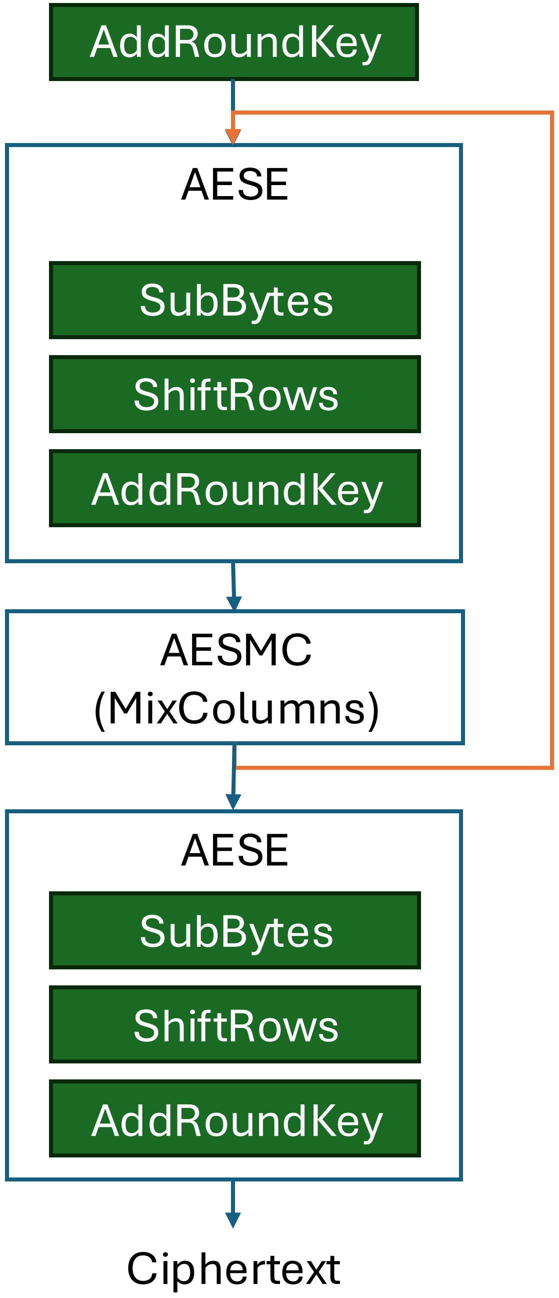


```
SYM_FUNC_START(__aes_ce_decrypt)
        sub             w3, w3, #2
        ld1             {v0.16b}, [x2]
        ld1             {v1.4s}, [x0], #16
        cmp             w3, #10
        bmi             0f
        bne             3f
        mov             v3.16b, v1.16b
        b               2f
0:      mov             v2.16b, v1.16b
        ld1             {v3.4s}, [x0], #16
1:      aesd            v0.16b, v2.16b
        aesimc          v0.16b, v0.16b
2:      ld1             {v1.4s}, [x0], #16
        aesd            v0.16b, v3.16b
        aesimc          v0.16b, v0.16b
3:      ld1             {v2.4s}, [x0], #16
        subs            w3, w3, #3
        aesd            v0.16b, v1.16b
        aesimc          v0.16b, v0.16b
        ld1             {v3.4s}, [x0], #16
        bpl             1b
        aesd            v0.16b, v2.16b
        eor             v0.16b, v0.16b, v3.16b
        st1             {v0.16b}, [x1]
        ret
SYM_FUNC_END(__aes_ce_decrypt)
```

Example Use Case

The AESE instruction can be used with the AESMC instruction to perform AES encryption.

Data is retrieved and stored from the SIMD and floating-point register file.


AddRoundKey
AESE
SubBytes
ShiftRows
AddRoundKey
AESMC
(MixColumns)
AESE
SubBytes
ShiftRows
AddRoundKey
Ciphertext