SPECT

Programmer Guide ISAv0.2

Version: 0.6

Git tag:

Tropic Square

April 3, 2024

Version history

Version Tag	Date	Author	Description
0.1	12.10.2022	Ondrej Ille	Initial version
0.2	7.11.2022	Ondrej Ille	Fix semantics of LSR instruction.
0.3	8.11.2022	Ondrej Ille	Fix SCB semantics.
0.4	14.11.2022	Ondrej Ille	Fix semantics of ST instruction (op1 instead of op2). Add note about modular instruction operands.
0.5	23.11.2022	Ondrej Ille	Add description of SW toolchain.
0.6	9.8.2023	Vit Masek	Move ISA descriptions to separate documents.

Bibliography

References

[1] FIPS 180-4

`https://csrc.nist.gov/pubs/fips/180-4/upd1/final`

[2] TROPIC01 Repository

`https://tropic-gitlab.corp.sldev.cz/internal/tropic01/tassic`

[3] ts-crypto-blocks

`https://tropic-gitlab.corp.sldev.cz/internal/development-environment/ts-crypto-blocks`

[4] ts-spect-fw

`https://tropic-gitlab.corp.sldev.cz/internal/sw-design/ts-spect-fw`

[5] Danger, Jean-Luc et al. “A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards.” Journal of Cryptographic Engineering 3 (2013): 241 - 265.

Contents

1 Glossary

- **CPU** - Central Processing Unit
- **ECC** - Elliptic Curve Cryptography
- **SPECT** - Secure Processor of Elliptic Curves for Tropic
- $P_{25519} = 2^{255} - 19$
- $P_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

2 Register field types

Meaning of Register field types is following:

- **RW** - Read-Write field
- **RO** - Read-only field
- **WO** - Write-only field
- **RW W1C** - Read-Write field, Write 1 to clear
- **RW W0C** - Read-Write field, Write 0 to clear
- **RW W1S** - Read-Write field, Write 1 to set
- **RW W0S** - Read-Write field, Write 0 to set
- **RW W1T** - Read-Write field, Write 1 to toggle
- **RW W0T** - Read-Write field, Write 0 to toggle

3 Introduction

This document provides a programmer's guide for SPECT. SPECT is a domain specific processing unit targeted for calculations of Elliptic Curve Cryptography (ECC). SPECT provides instructions for calculation with 256 bit numbers and modular arithmetics. SPECT is useful to implement operations/algorithms such as:

- ECDSA – Elliptic Curve Digital Signature Algorithm
- ECDH – Elliptic Curve Diffe-Hellman

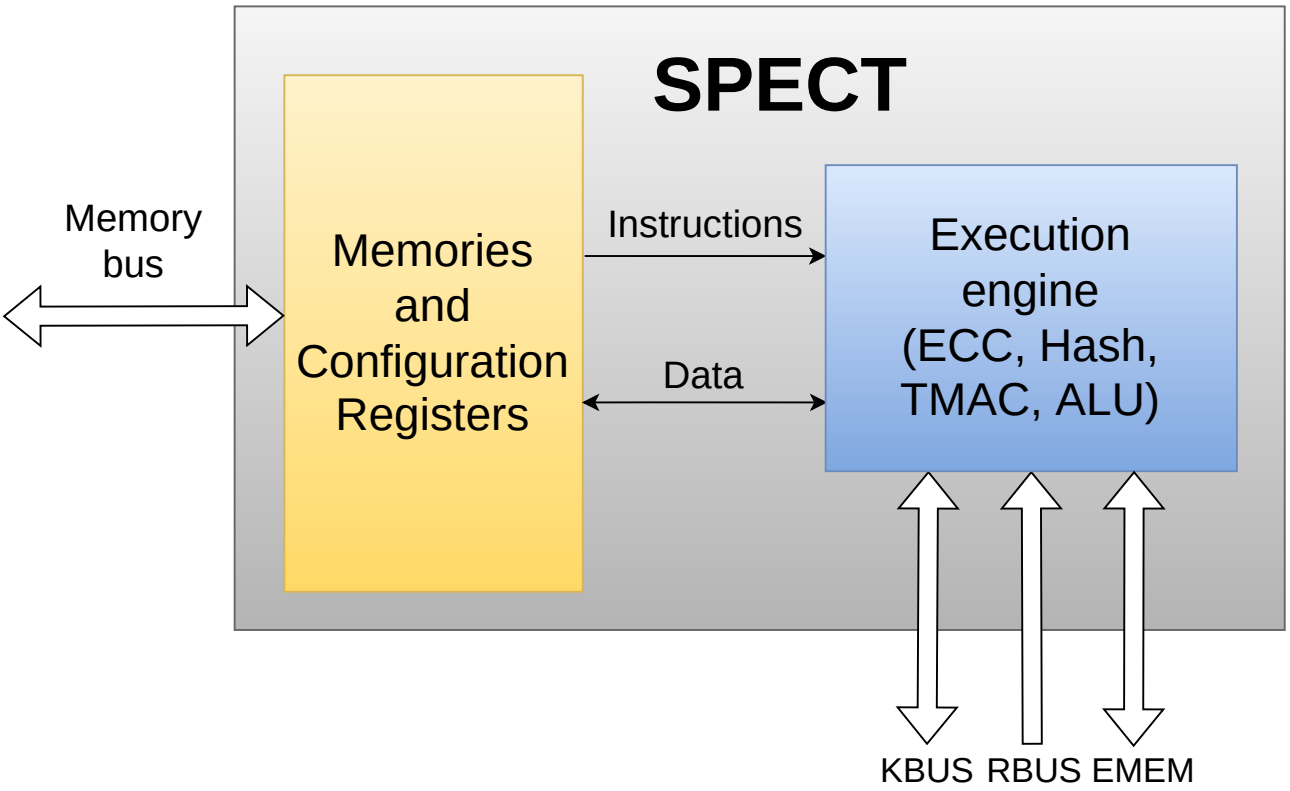

Figure 1: SPECT – Block diagram

4 Programmer's model

SPECT programmer's model consists of:

- 32 x 256 bit general purpose registers (**R0** - **R31**).
- **PC** - Program counter.
- Zero (**Z**), Carry (**C**) and Error (**E**) flag.
- HW **RAR** (Return Address Register) stack for nested procedure calls.
- 2048 B read-write memory space in address range 0x0000 – 0x07FC.
- 512 B write-only memory space in address range 0x1000 – 0x11FC.
- 2048 B read-only memory space in address range 0x3000 – 0x37FC.
- 144 B read-only memory space in address range 0x4000 – 0x408C. (from ISA v0.2)
- 50 B write-only memory space in address range 0x5000 – 0x504C. (from ISA v0.2)

Note

SPECTs address space is 32 bit word organized. Load and store instructions works with 256 bit values and it always uses 8 consecutive words in the memory. E.g. 0x0020 - 0x003C.

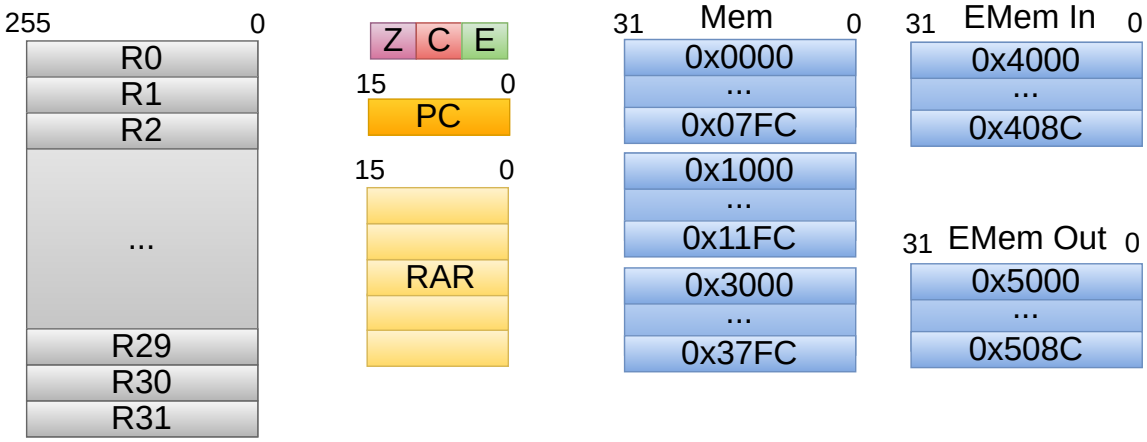

Figure 2: SPECT – Programmer model

4.1 Subroutine calls

SPECT contains HW **RAR** stack, and it pushes return address from subroutine to RAR stack each time when it executes CALL instruction. When SPECT executes RET instruction, it pops value from RAR stack and updates **PC**. HW **RAR** stack supports up to 5nested subroutine calls.

Note

Behavior of SPECT when number of nested subroutine calls is exceeded is undefined.

4.2 KBUS

From SPECT ISA v0.2, SPECT HW implements special 32 bit BUS interface (KBUS) to store and load cryptographic keys. Particular key within the system is identified by following parameters:

- Type – Type of the key (ECC, SHPUB, STPRIV etc.). It specifies the location of tke key in the system.
- Slot – Particular key slot of the specified type. One slot can contain multiple related keys (E.g. scalar and prefix in case of EdDSA).
- Offset – Offset within the slot. Specifies position of the particular key.

SPECT ISA provides three instruction for KBUS – LDK, STK, KBO. LDK is used to load keys, STK to store keys and KBO for further control of the KBUS. Because SPECT operates with 256 bit values, both LDK and STK execute 8 consecutive KBUS transactions incrementing the offset.
For purpose of this document, notation KBUS_READ[type,slot,offset] indicates result of 8 consecutive KBUS read transactions (increasing offset). Data from the first transaction are considered as LSBs, data from the last transaction are considered as MSBs.
Further, notation KBUS_WRITE[key,type,slot,offset] indicates 8 consecutive KBUS write transactions (increasing offset) with wdata = key. In first transaction, wdata = key[31:0]. In the last transaction, wdata = key[255:223]. KBUS_OP[type,slot,op] indicates one KBUS transaction of specific OP (e.g. "program slot").
Fore more information about KBUS, see TROPIC01 Functional Specification, Section 19 [2].

4.3 RBUS

SPECT HW implements special 32 bit interface for requesting random numbers from the external systems RNG. SPECT ISA provides possibility to fetch 256 bit random number with GRV instruction.

4.4 Modular arithmetics

SPECT provides instructions for finite field arithmetic such as addition, subtraction and multiplication with 256 bit operands stored in general purpose registers. SPECT supports fast multiplication in Ed25519 and P-256 curves finite fields via dedicated instructions – MUL25519 and MUL256. Modular arithmetics with generic modulus specified by value in **R31** is supported by instructions ADDP, SUBP, MULP. SPECT also supports modular reduction of 512 bit number with REDP instruction.
When programming with modular instructions, one needs to be careful about input operands of such instructions. Following conditions must be met:

- op2 < P_{25519} and op3 < P_{25519} for MUL25519 instruction.
- op2 < P_{256} and op3 < P_{256} for MUL256 instruction.
- op2 < R31 and op3 < R31 for ADDP, SUBP instructions.
- R31 != 0 and R31 != 1 for ADDP, SUBP, MULP, REDP instructions.

if these conditions are not met when invoking such a instruction, result of the instruction calculation is undefined (value in op1).

Note

Performance of MULP when **R31** = P_{25519} / P_{256} is lower than performance of MUL25519 / MUL256.

4.5 SHA512

SPECT HW supports SHA512 Hash calculation as specified in [1]. SPECT can calculate SHA512 hash from arbitrarily long data stream. When SPECT executes HASH_IT instruction, it resets context in its execution engine to initialization vector as specified in [1]. Each execution of HASH instruction processes 1024 bit block, and executes next round of SHA512 calculation.

Note

SPECT HW does not add any padding of input data. It is responsibility of the firmware or external system to add such padding.

4.6 TMAC

From SPECT ISA v0.2, SPECT HW supports TMAC calculation as specified in TMAC documentation as part of ts-crypto-blocks repository [3]. TMAC stands for Tropic Message Authentication Code. It is a custom MAC function inspired by KMAC function. It uses masked implementation of KECCAK permutation with 400 bits internal state and rate of 18 bytes.
SPECT ISA provides four instructions for TMAC calculation.

- TMAC_IT – initialize underlying KECCAK core with 800 bits of mask and a guard.
- TMAC_IS – initialize TMAC with initialization string as defined in TMAC specification.
- TMAC_UP – updates internal state with another 18 bytes of data.
- TMAC_RD – Squeeze 256 bits from the underlying KECCAK core as an output of the TMAC function.

4.7 Group Scalar Blinding

SPECT HW supports scalar blinding by a random number as a side-channel countermeasure with SCB instruction. It blinds the scalar sc using group scalar randomization method as defined in [5] with 256 bit random number. The random number rng shall be obtained in advance by GRV instruction as described above. The group order q shall be present in **R31**.
SCB performs this exact function:

$$Blind(sc, rng, q) = q \times (rng|(2^{255} + 2^{223})) + sc$$

4.8 SPECT invocation

SPECT firmware execution is invoked by external system that has access to its memory space via memory bus as shown in following figure:

Note

Address of the first instruction executed by SPECT after **COMMAND[START]** = 1 is written, is fixed and defined by a system that integrates SPECT.

4.9 Invalid instructions

When SPECT attempts to execute invalid instruction, it aborts firmware execution and sets **STATUS[ERR]** = 1.

Note

Invalid instruction means invalid opcode or not matching parity bit in the instruction code. Unless a fault, usual cause of this is e.g. missing RET instruction in subroutine or END instruction at the end of the firmware execution.

4.10 Soft Reset

SPECT can be reset by external system by writing **COMMAND[SOFT_RESET]** = 1. When SPECT is reset, it aborts any firmware execution and resets its internal state.

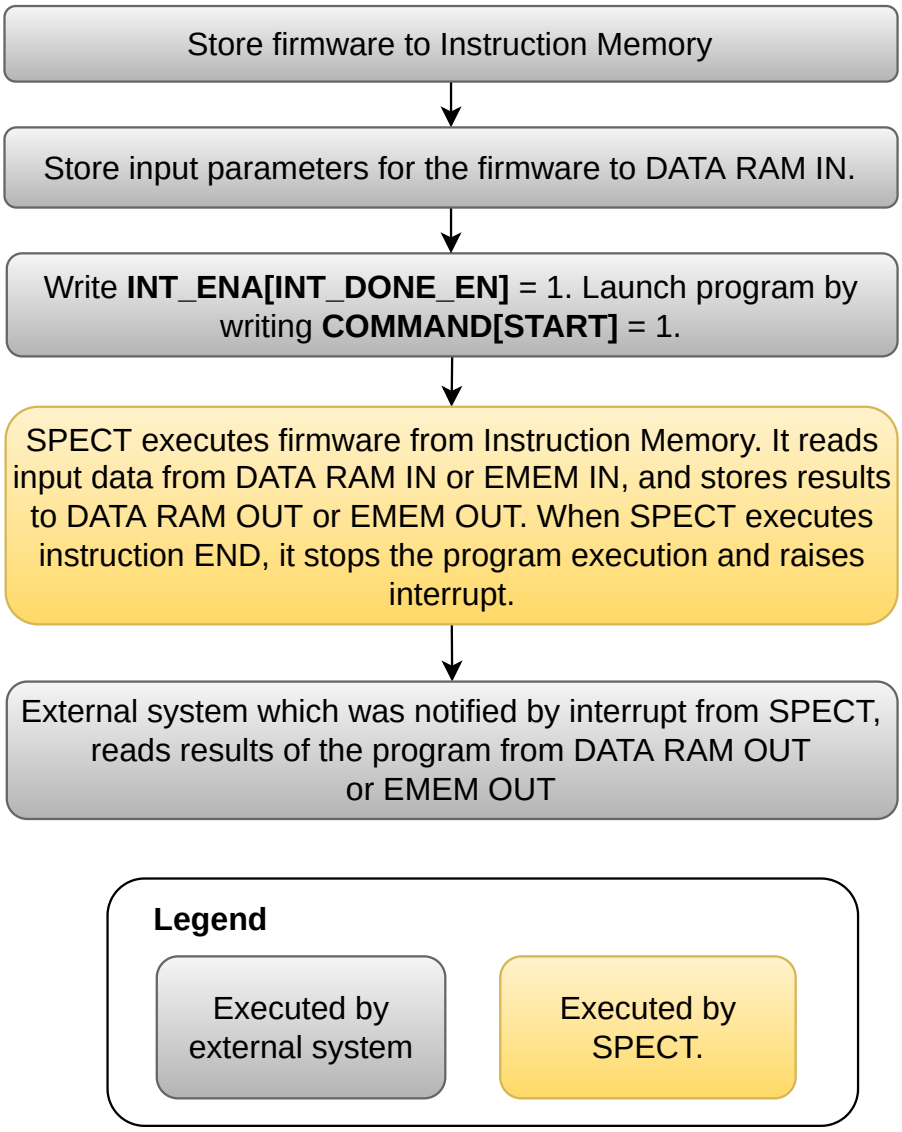

Figure 3: SPECT – Invocation

Note

Because GPRs are implemented as RAM, Soft Reset does not effect them in any way. GPRs can be cleared only by firmware.

4.11 Interrupts

SPECT firmware execution can not be interrupted by an external event (other than Soft Reset). SPECT itself can generate following interrupts for external system:

- Done – Enabled when **INT_ENA[INT_DONE_EN]** = 1. Generated when SPECT firmware executes END instruction

- Error – Enabled when **INT_ENA[INT_DONE_EN]** = 1. Generated when SPECT experience internal error (invalid instruction, bit-flip in SHA512 or TMAC core etc.)

5 SPECT Memory Map

Base Address: 0x0000 0000
End Address: 0x0000 9FFF

Memory region	Address offset range	Size
Data RAM IN	0x0000 0000 0x0000 07FF	2 KB
Data RAM OUT	0x0000 1000 0x0000 11FF	512 bytes
Configuration registers	0x0000 2000 0x0000 200F	16 bytes
Constants ROM	0x0000 3000 0x0000 37FF	2 KB
External Memory In	0x0000 4000 0x0000 403F	64 bytes
External Memory Out	0x0000 5000 0x0000 504F	80 bytes
Instruction Memory	0x0000 8000 0x0000 9FFF	8 KB

5.1 Configuration registers

Base Address: 0x0000 2000
End Address: 0x0000 200F

Address Offset	Register Name	Reset Value
0x0	BLOCK_ID	0x000-0030
0x4	COMMAND	0x00000000
0x8	STATUS	0x00000001
0xc	INT_ENA	0x00000000

Register name:	BLOCK_ID
Address:	0x2000

Field	Type	Reset value	Bits	Description
ID_CODE	RO	0x30	15:0	Identification code
REV_CODE	RO	-	19:16	Revision code

Register name:	COMMAND
Address:	0x2004

Field	Type	Reset value	Bits	Description
START	WO W1S;	0x0	0:0	Starts SPECT FW operation
SOFT_RESET	WO	0x0	1:1	Stops FW execution and resets SPECT

Register name:	STATUS
Address:	0x2008

Field	Type	Reset value	Bits	Description
IDLE	RO	0x1	0:0	SPECT is in IDLE mode
DONE	RW W1C	0x0	1:1	Active when SPECT successfully completes the calculation
ERR	RW W1C	0x0	2:2	Active when SPECT ends the calculation with error

Register name:		INT_ENA		
Address:		0x200c		
Field	**Type**	**Reset value**	**Bits**	**Description**
INT_DONE_EN	RW	0x0	0:0	Enables DONE interrupt
INT_ERR_EN	RW	0x0	1:1	Enables ERROR interrupt

5.2 Data RAM IN

Data RAM IN is a memory where external system stores parameters for SPECT firmware before it starts its execution. SPECT firmware sees it as read-write memory.

5.3 Data RAM OUT

Data RAM OUT is a memory where SPECT firmware stores results of its calculation, and external system reads such results after SPECT firmware execution ends. SPECT firmware sees it as write-only memory.

5.4 Instruction Memory

Instruction memory contains the firmware executed by SPECT. External system preloads the SPECT firmware to this memory in its boot up sequence. SPECT firmware do not have access to this memory via load and store instructions.

5.5 Constant ROM

Constant ROM contains a ROM image with constants used by SPECT firmware (e.g. P_{25519}). SPECT firmware sees it as read-only memory.
Content of such ROM is part of SPECT firmware repository. See [4].

5.6 External Memory

From SPECT ISA v0.2, SPECT HW implements special BUS interface (EMEM) to access different memory space within the external system. There are two memories:

- External Memory IN – SPECT firmware sees it as read-only memory.
- External Memory OUT – SPECT firmware sees it as write-only memory.

These memories are mapped in to SPECT memory space. Load and store instruction directs read / write transactions to SPECTs memory subsystem or on to EMEM interface depending on the address in Addr field os the instruction.

6 SPECT Assembler

SPECT assembler has support for following assembly language features:

- Function labels
- Constant definitions
- Include other assembly file
- Conditional compilation

6.1 Tool requirements

SPECT SW toolchain requires following tools:

- CMAKE 3.18.2 or higher

6.2 Function labels

SPECT compiler allows definition of function labels, and passing them as NewPc of J instructions, e.g like so:

```
_start:
    CALL my_func
    END

my_func:
    ADD r0, r1, r2
    RET
```

6.3 Constant definitions

SPECT compiler allows definition of constants, and passing them as Addr of M instructions or Immediate operand of I type instructions like so:

```
threshold .eq 0x12

_start:
    ADDI r0, r0, threshold
```

```
p25519_addr .eq 0x3020

_start:
    LD r31, p25519_addr
```

Note

Currently, SPECT compiler does not support expression parsing. It only supports simple decimal, hexadecimal or binary value when defining constants.

6.4 Include other assembly file

Multiple .s assembly files can be connected together in SPECT source code via "`.include`" directive, e.g. like so:

```
_start:
    NOP

.include <other_s_file>
    END
```

6.5 Conditional compilation

SPECT compiler supports conditional compilation using "`.ifdef`" directive.

```
    .define MY_DEFINE
    .ifdef MY_DEFINE
        <some code>
    .else
        <some other code>
    .endif
```

By using "`--isa-version=X`" switch of SPECT compiler or ISS, symbol "`SPECT_ISA_VERSION_<X>`" is defined automatically. The default value is the always newest ISA version.

7 SW Toolchain

SW toolchain intended for SW development and debugging SPECT firmware is available. The toolchain has following applications available:

- `spect_compiler` – A compiler/assembler which creates .hex file from .s assembly file.
- `spect_iss` – Instruction set simulator with simple command line debugger. It can simulate .s file as well as .hex file.

Options for each of the applications are described when using `--help` command line option. Options available inside interactive shell of `spect_iss` are available with `--help` command line option or `help` command.

8 Open Issues

Document contains following open issues: