OPEN

Compute Project

Caliptra Integration Specification

V0.5

CONTRIBUTORS

Caliptra Consortium

REVISION HISTORY

Date	Revision #	Description
6/20/2022	V0.1	Initial draft
	V0.5	**TARGET DATE:** End of July'22 You will see TODOs that will be addressed by this time

Table of Contents

Table of Figures

Table of Tables

1 Scope

The objective of this document is to describe the Caliptra hardware implementation requirements and details, and any pertinent release notes. This document is intended for a high-level overview of the IP used in Caliptra.

This document is not intended for any micro-architectural design specifications. Detailed information on each of the IP components are shared in individual documents, where applicable.

2 Overview

This document contains high level information on the Caliptra HW design. The details include open-source IP information, configuration settings for open-source IP (if applicable) and IP written specifically for Caliptra.

2.1 Acronyms and Abbreviations

For the purposes of this document, the following abbreviations apply:

Table 1: Acronyms and Abbreviations

Abbreviation	Description
AHB	AMBA Advanced High-Performance Bus
APB	AMBA Advanced Peripheral Bus
AES	Advanced Encryption Standard
ECC	Elliptic Curve Cryptography
RISC	Reduced Instruction Set Computer
SHA	Secure Hashing Algorithm
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter

2.2 Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14] [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.3 References / Related Specifications

The blocks described in this document are either obtained from open-source GitHub repositories, developed from scratch, or modification of open-source implementations. Links to relevant documentation and GitHub sources will be shared in this section.

Table 2: Related Specifications

IP/Block	GitHub URL	Documentation	Link
Cores-VeeR	GitHub - chipsalliance/Cores-VeeR-EL2	VeeR EL2 Programmer's	chipsalliance/Cores-VeeR-EL2 · GitHubPDF

		Reference Manual	
AHB Lite Bus	aignacio/ahb_lite_bus: AHB Bus lite v3.0 (github.com)	AHB Lite Protocol	ahb_lite_bus/docs at master · aignacio/ahb_lite_bus (github.com)
		Figure 2	ahb_lite_bus/diagram_ahb_bus.png at master · aignacio/ahb_lite_bus (github.com)
SHA 256	secworks/sha256: Hardware implementation of the SHA-256 cryptographic hash function (github.com)		
SHA 512			
SPI Controller	https://github.com/pulp-platform/axi_spi_master		

3 Synopsys DC Data

4 DFT

5 Clock Gating

6 Block Diagram

Caliptra top-level block diagram is shown in the figure below.

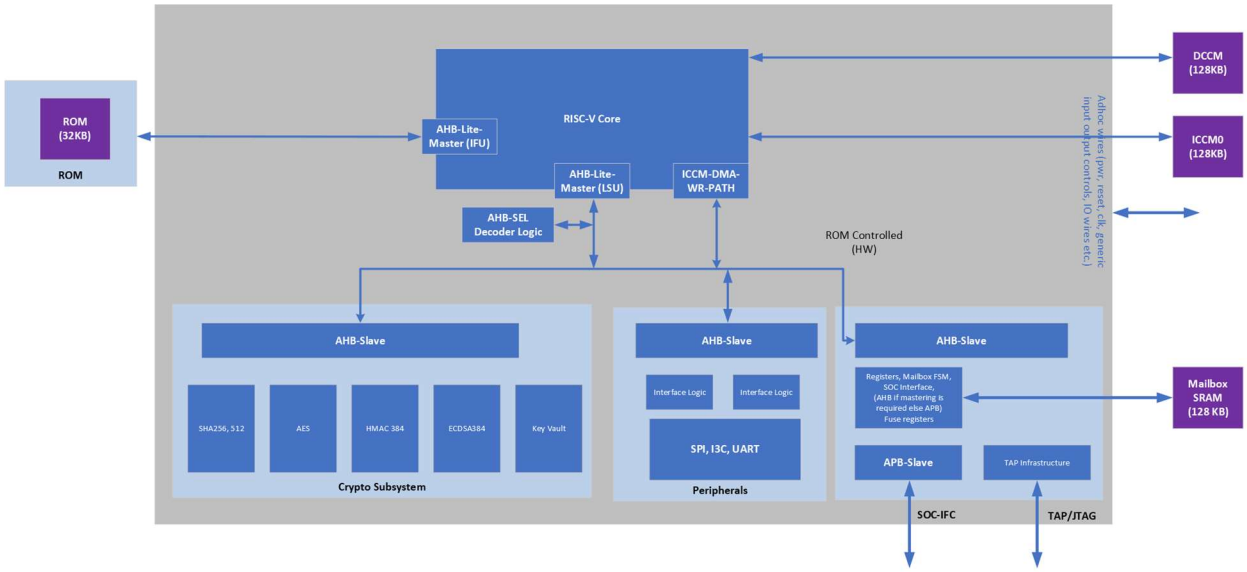

Figure 1: Caliptra Block Diagram

6.1 Passive vs Active Profile

In passive profile, none of the IOs in the peripherals are active. This will be an integration time parameter passed to the HW which is exposed to ROM. Please see boot flows to see the difference in the HW/ROM behavior for passive profile vs active profile.

From SOC integration POV, peripheral IOs can be tied off appropriately for passive profile at SOC integration time.

7 SOC Interface

TODO 0p5: This is a WIP list

7.1 Block Diagram

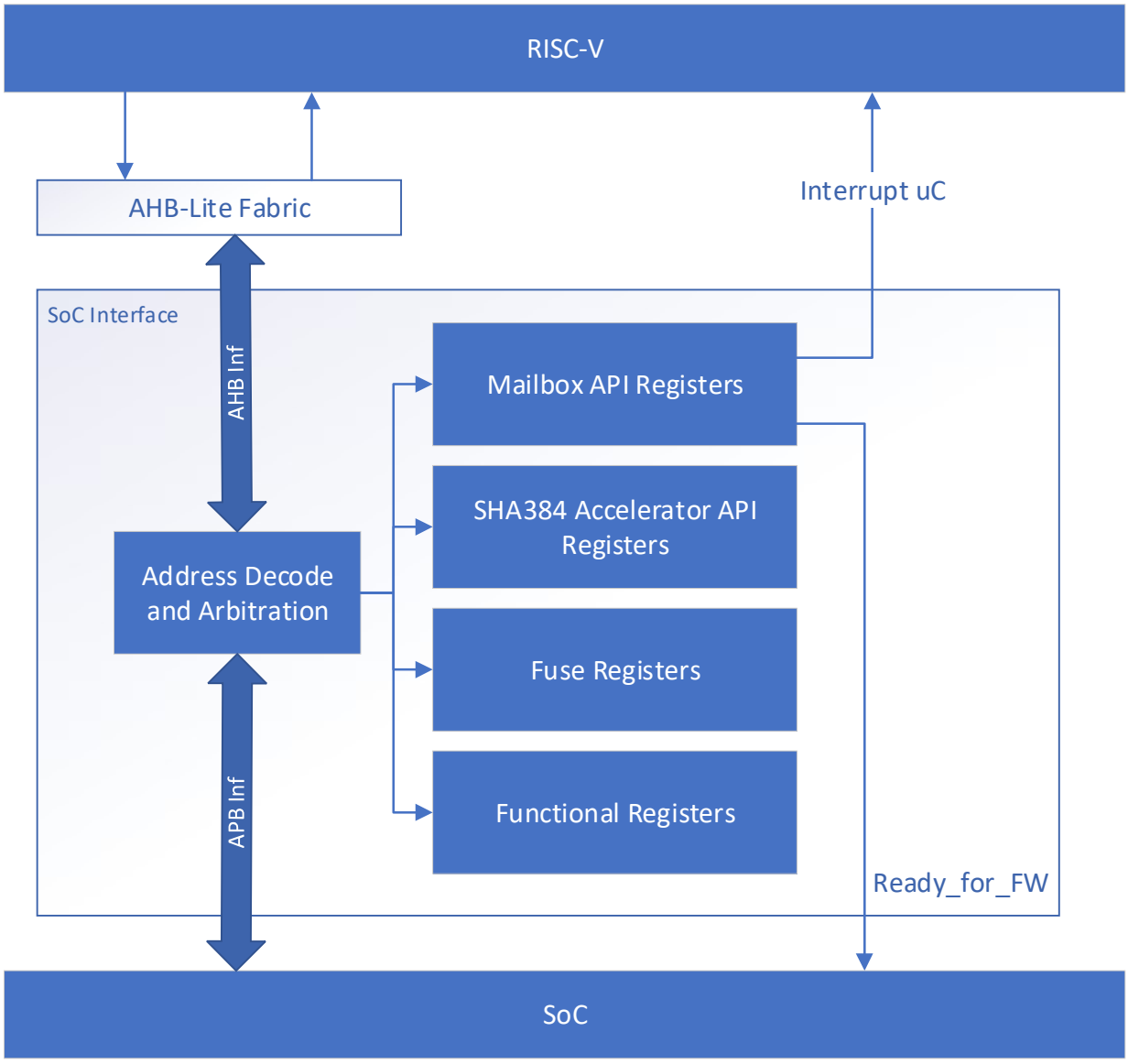

Figure 3: SoC Interface Block Diagram

7.2 Integration Parameters

Table 3: Integration Parameters

Parameter Name	Width	Description
APB_ADDR_WIDTH	32	Width of the APB Address field. Default to 32.
APB_DATA_WIDTH	32	Width of the APB Data field Default to 32.
APB_USER_REQ_WIDTH	<TODO>	Width of the APB PAUSER field
ENABLE_INTERNAL_TRNG	1	Enable Internal TRNG. Default to 0. 1'b0: External TRNG Source 1'b1: Internal TRNG Source
TODO		

7.3 Interface

Table 4: Interface Signals

Signal Name	Width	Driver	Synchronous (as viewed from Caliptra's boundary)	Description
Clocks and Resets				
cptra_pwrgood	1	Input	Asynchronous Assertion Synchronous deassertion to clk	Active high power good indicator de-assertion will hard reset Caliptra
cptra_rst_b	1	Input	Asynchronous Assertion Synchronous deassertion to clk	Active low asynchronous reset
clk	1	Input		Covergence & Validation done at 400MHz. All other frequencies are upto user.
APB Interface				

PADDR	32	Input	Synchronous to clk	Address bus
PPROT	3	Input	Synchronous to clk	Protection level
PSEL	1	Input	Synchronous to clk	Select line
PENABLE	1	Input	Synchronous to clk	Indicates the second and subsequent cycles
PWRITE	1	Input	Synchronous to clk	Indicates write access when high read when low
PWDATA	32	Input	Synchronous to clk	Write data bus
PAUSER	APB_USER_REQ_WIDTH	Input	Synchronous to clk	User request attributes
PREADY	1	Output	Synchronous to clk	Used to extend an APB transfer by completer
PRDATA	32	Output	Synchronous to clk	Read data bus
PSLVERR	1	Output	Synchronous to clk	Transfer error
QSPI Interface				
qspi_clk_o	1	Output		QSPI clock
qspi_cs_no	2	Output	Synchronous to qspi_clk_o	QSPI chip select
qspi_d_io	4	IO	Synchronous to qspi_clk_o	QSPI data lanes for transmitting opcode, address and receiving data
Mailbox Notifications				
ready_for_fuses	1	Output	Synchronous to clk	Indicates that Caliptra is ready for fuse programming
ready_for_fw_push	1	Output	Synchronous to clk	Indicates that Caliptra is ready for firmware
ready_for_runtime	1	Output	Synchronous to clk	Indicates that Caliptra FW is ready for RT flows
mailbox_data_avail	1	Output	Synchronous to clk	Indicates that the mailbox has data for SoC to read (reflects the value of the register)
mailbox_flow_done	1	Output	Synchronous to clk	Indicates that the mailbox flow is complete (reflects the value of the register)
SRAM Interface				

mbox_sram_cs	1	Output	Synchronous to clk	Chip select for mbox SRAM
mbox_sram_we	1	Output	Synchronous to clk	Write enable for mbox SRAM
mbox_sram_addr	MBOX_ADDR_W	Output	Synchronous to clk	Addr lines for mbox SRAM
mbox_sram_wdata	MBOX_DATA_W	Output	Synchronous to clk	Write data for mbox SRAM
mbox_sram_rdata	MBOX_DATA_W	Input	Synchronous to clk	Read data for mbox SRAM
imem_cs	1	Output	Synchronous to clk	Chip select for imem SROM
imem_addr	IMEM_ADDR_WIDTH	Output	Synchronous to clk	Addr lines for imem SROM
imem_rdata	IMEM_DATA_WIDTH	Input	Synchronous to clk	Read data for imem SROM
iccm_clken	ICCM_NUM_BANKS	Input	Synchronous to clk	Per-bank clock enable
iccm_wren_bank	ICCM_NUM_BANKS	Input	Synchronous to clk	Per-bank write enable
iccm_addr_bank	ICCM_NUM_BANKS x (ICCM_BITS-4)	Input	Synchronous to clk	Per-bank address
iccm_bank_wr_data	ICCM_NUM_BANKS x 39	Input	Synchronous to clk	Per-bank input data
iccm_bank_dout	ICCM_NUM_BANKS x 39	Output	Synchronous to clk	Per-bank output data
			Synchronous to clk	
dccm_clken	DCCM_NUM_BANKS	Input	Synchronous to clk	Per-bank clock enable
dccm_wren_bank	DCCM_NUM_BANKS	Input	Synchronous to clk	Per-bank write enable
dccm_addr_bank	DCCM_NUM_BANKS x (DCCM_BITS-4)	Input	Synchronous to clk	Per-bank address
dccm_wr_data_bank	DCCM_NUM_BANKS x DCCM_FDATA_WIDTH	Input	Synchronous to clk	Per-bank input data

dccm_bank_dout	DCCM_NUM_BANKS x DCCM_FDATA_WIDTH	Output	Synchronous to clk	Per-bank output data
JTag Interface				
jtag_tck	1	input		
jtag_tms	1	input	Synchronous to tck	
jtag_tdi	1	input	Synchronous to tck	
jtag_trst_n	1	input	Async Deassertion Assertion Synchronous to tck	
jtag_tdo	1	output	Synchronous to tck	
Security and Misc Signals				
CPTRA_OBF_KEY	256	Input strap	Asynchronous	Obfuscation key to be driven by SOC at integration time (ideally just before tape-in and the knowledge of this key must be protected unless PUF is driving this). The key will be latched by Caliptra on caliptra powergood deassertion. It is cleared after its use and can only re-latched on a power cycle (powergood deassertion to assertion)
SECURITY_STATE	3	Input	Synchronous to clk	Security state that Caliptra should take (eg. Manufacturing, Secure, Unsecure etc.); Latched by Caliptra on powergood deassertion. Any time the state changes to debug mode, all keys/assets/secrets stored in fuses or key vault are cleared and cryptos are also flushed if they were being used.
scan_mode	1	Input	Synchronous to clk	Needs to be set before entering scan mode. This allows Caliptra to flush any assets/secrets present in key

				vault & flops if the transition is happening from secure state.
GENERIC_INPUT_WIRES	64	Input	Synchronous to clk	Placeholder of input wires for late binding features. These values are reflected into registers that are exposed to FW
GENERIC_OUTPUT_WIRES	64	Output	Synchronous to clk	Placeholder of output wires for late binding features. FW will be able to set the wires appropriately.
CALIPTRA_ERROR_FATAL	1	Output	Synchronous to clk	Indicates a fatal error from caliptra
CALIPTRA_ERROR_NON_FATAL	1	Output	Synchronous to clk	Indicates a non fatal error from caliptra
BootFSM_BrkPoint	1	Input Strap	Asynchronous	Stops the BootFSM to allow TAP writes set up behavior such as skip or run ROM flows or stepping though BootFSM
eTRNG_REQ	1	Output	Synchronous to clk	External Source Mode: TRNG_REQ to SOC. SOC will write to TRNG architectural registers with a NIST compliant entropy. Internal Source Mode: TRNG_REQ to SOC. SOC will enable external RNG digital bitstream input into TRNG_DATA/TRNG_VALID
iTRNG_DATA	4	Input	Synchronous to clk	External Source Mode: Not used Internal Source Mode Only: RNG digital bit stream from SOC which is sampled when TRNG_VALID is high.
iTRNG_VALID	1	Input	Synchronous to clk	External Source Mode: Not used Internal Source Mode Only: RNG bit valid. This is a per-transaction valid. TRNG_DATA can be sampled whenever this bit is high. The expected TRNG_VALID output rate is about 50KHz.

7.4 Architectural Registers and Fuses

Control registers and fuses are documented on github.

https://github.com/Project-Caliptra/rtl-caliptra/tree/main/src/integration/docs

7.5 Fuses

Fuses are writable only one time, and require a hard reset to be written again.

Once all fuses are written, the fuse done register at the end of the fuse address space needs to be set to 1 in order to proceed with the boot flow.

7.6 Interface Rules

7.6.1 APB arbitration

Caliptra is a “slave” on the APB bus. If SOCs have multiple APBs or other proprietary-fabric protocols, that require any special fabric arbitration, it is done at SOC level.

7.6.2 Undefined address accesses

All accesses that are outside of the defined address space of Caliptra will be responded by Caliptra’s SOC interface

- All reads to undefined addresses get completions with zero data
- All writes are dropped
- All other undefined opcodes will be silently dropped

All accesses MUST be 32-bit aligned. Misaligned accesses will be treated as 32-bit aligned.

7.6.3 Undefined mailbox usages

A trusted/valid requester that locks the mailbox and never releases will cause the mailbox to be locked indefinitely.

Caliptra FW internally has the capability to force release the mailbox based on various timers but there is no architectural requirement to use it.

7.6.4 Straps

All straps are sampled on caliptra pwrgood signal – refer to interface table for list of straps.

7.6.5 Deobfuscation Key

SOC ECO’s the key at the tape-in time of the SOC and must be protected from common knowledge.

It must follow the security rules defined in the arch spec

SOC must ensure that there are no SCAN cells on the flops that latch this key “internal” to caliptra.

8 Boot

8.1.1 Boot FSM

The Boot FSM is responsible for detecting the SoC bringing Caliptra out of reset. Part of this flow involves signaling to the SoC that we are awake and ready for fuses. Once fuses have been populated and the SoC has indicated that they are done downloading fuses, we can wake up the rest of the IP by de-asserting the internal reset.

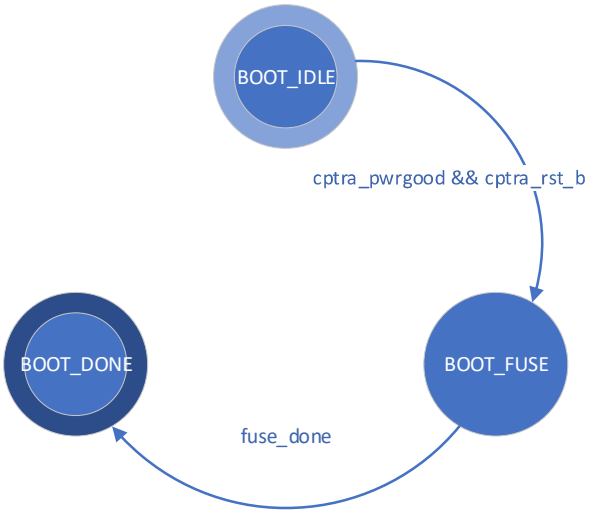

Figure 2: Mailbox Boot FSM State Diagram

The boot FSM first looks for the SoC to assert cptra_pwrgood and de-assert cptra_rst_b. In the BOOT_FUSE state, Caliptra will signal to the SoC that it is ready for fuses. Once the SoC is done writing fuses, it will set the fuse done register and the FSM will advance to BOOT_DONE.

BOOT_DONE enables Caliptra reset de-assertion through a two flip-flop synchronizer.

9 SOC Mailbox

The Caliptra Mailbox is the primary communication method between Caliptra and the SoC it is integrated into.

The Caliptra Mailbox uses an APB interface to communicate with the SoC. The SoC can write to and read from various memory mapped register locations over the APB interface in order to pass information to Caliptra.

Caliptra in turn also uses the mailbox to pass information back to the SoC. The interface does not author any transaction on the APB interface, it will only signal to the SoC that data is available in the mailbox and it is the responsibility of the SoC to read that data from the mailbox.

9.1 SoC Interface

The SoC will communicate with the mailbox through an APB Interface. The SoC acts as the requester with the Caliptra mailbox as the receiver.

The PAUSER bits will be used for the SoC to identify which device is accessing the mailbox.

9.2 Mailbox

The Caliptra Mailbox is a 128KB buffer used for exchanging data between the SoC and the Caliptra microcontroller (uC).

When a mailbox is populated by the SoC, we will send an interrupt to the uC to indicate that a command is available in the mailbox. The uC will be responsible for reading from and responding to the command.

When a mailbox is populated by the uC, we will send a wire indication to the SoC that a command is available in the mailbox. The SoC will be responsible for reading from and responding to the command.

Mailboxes are generic data passing structures, we will only enforce the protocol for writing to and reading from the mailbox. How the command and data is interpreted by the uC and SoC are not enforced in this document.

9.3 Sender Protocol

Sending data to the mailbox:

1. Requester queries the mailbox by reading the LOCK control register.
   - If LOCK returns 0, LOCK is granted and will be set to 1.
   - If LOCK returns 1, MBOX is locked for another device.
2. Requester writes the command to the COMMAND register.
3. Requester writes the data length in bytes to the DLEN register.
4. Requester writes data packets to the MBOX DATAIN register.
5. Requester writes to the EXECUTE register.
6. Requester reads the STATUS register.

   Status can return:

   CMD_BUSY - 2'b00 – Indicates the requested command is still in progress

   DATA_READY - 2'b01 – Indicates the return data is in the mailbox for requested command

   CMD_COMPLETE- 2'b10 – Indicates the successful completion of the requested command

   CMD_FAILURE- 2'b11 – Indicates the requested command failed

7. Requester reads the response if DATA_READY was the status.
8. Requester resets the EXECUTE register to release the lock.

Notes on behavior:

Once LOCK is granted, the mailbox is locked until that device has concluded its operation. We should have a mechanism to terminate a lock early or release the lock if the device does not proceed to use it.

Mailbox is responsible for only accepting writes from the device that requested and locked the mailbox.

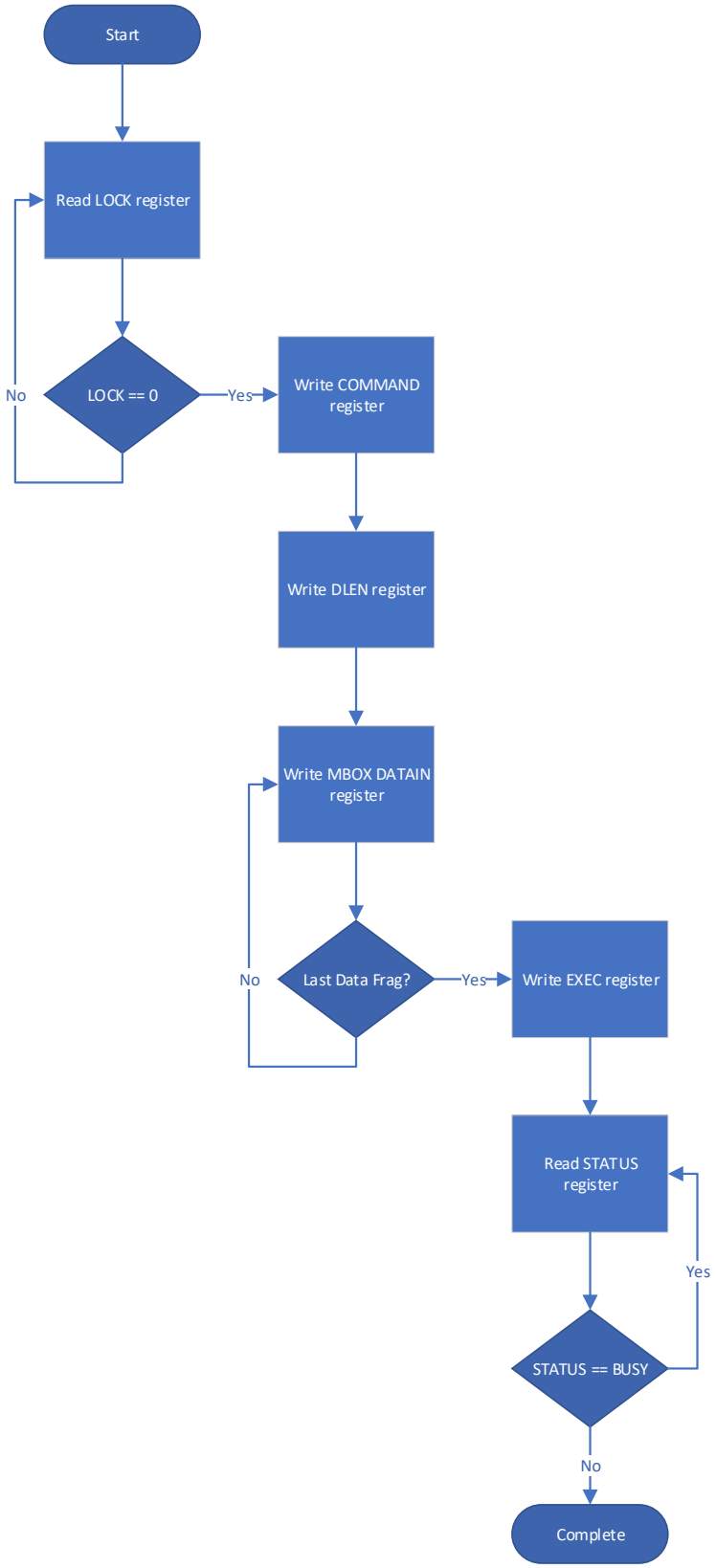

Figure 4: Sender protocol flow chart

9.4 Receiver Protocol

Upon receiving indication that mailbox has been populated, the appropriate device can read the mailbox. This is indicated by a dedicated wire that is asserted when Caliptra populates the mailbox for SoC consumption.

Receiving data from the mailbox:

1. On mailbox_data_avail assertion, the receiver reads the COMMAND register.
2. Receiver reads the DLEN register.
3. Receiver reads the MBOX DATAOUT register.
   - Continue reading MBOX DATAOUT register until DLEN bytes are read.
4. If a response is required, receiver can populate the mailbox with the response by writing DATAIN
5. Set the mailbox status register to hand control back to the sender to read the response
6. The sender will reset the EXECUTE register after reading the response.
   - This releases the LOCK on the mailbox.

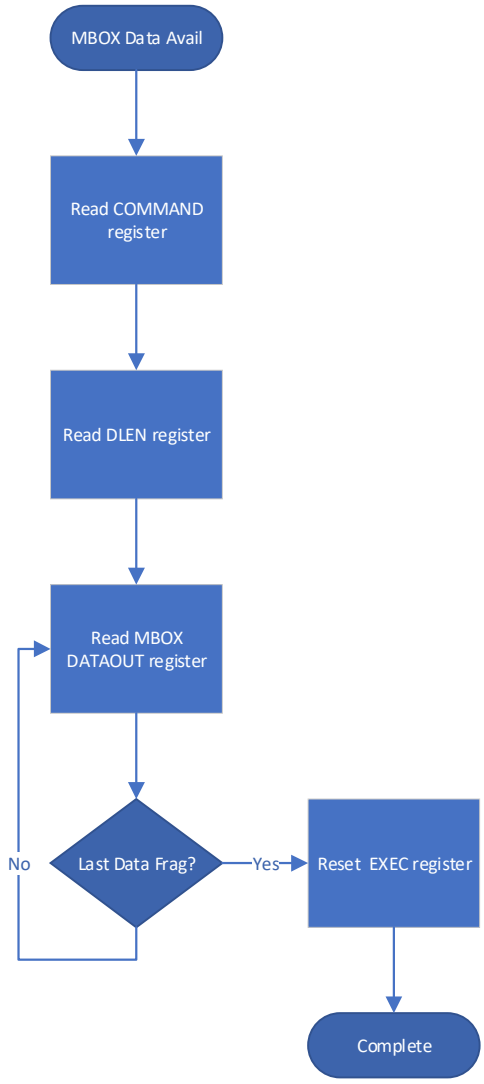

Figure 5: Receiver protocol flowchart

9.5 Mailbox Arbitration

From a mailbox protocol point of view, as long as PAUSER_ATTRIBUTE_REGISTERS carry valid requestors, mailbox lock can be obtained by any of those valid requestors but only one of them at any given time. While the mailbox flow is happening, all other requestors will not get a grant.

A request for lock that is denied due to Firmware having the lock will result in an interrupt to the Firmware. Firmware can optionally use this interrupt to release the lock.

9.6 PAUSER Attribute Register

- 5 PAUSER attribute registers are implemented at SOC interface
- By default any request with PAUSER value of '1 will be considered valid
- Integrators can also set the values of these register at integration time instead.
- These registers MUST be locked by the SOC to prevent unauthorized access.
- These registers are also programmable by trusted SOC logic (ROM, RTL, FW) as long as the lock hasn't been set.

Register	Address	Description
VALID_PAUSER[4:0][31:0]	0x30030020	5 registers for programming PAUSER values that will be considered valid for accessing the mailbox protocol. Requests with PAUSER attributes that are not in this list will be ignored.
PAUSER_LOCK[4:0]	0x30030034	5 registers, bit 0 of each locks the associated VALID_PAUSER register

9.7 Caliptra Mailbox Protocol

Once the SoC side has written the EXECUTE register, the mailbox will send an interrupt to the uC.

The uC will read the COMMAND and DLEN registers, as well as the data populated in the mailbox.

The uC can signal back to SoC through functional registers, and populate COMMAND, DLEN, and MAILBOX as well.

10 SOC SHA Acceleration Block

10.1 Overview

The SHA Acceleration Block sits in the SoC interface. The SoC can access the accelerator through its hardware API and stream data to be hashed over the APB interface.

SHA Acceleration Block utilizes a similar protocol to the mailbox, but has its own dedicated registers.

SHA_LOCK register is set on read. A read of 0 indicates the SHA was unlocked and will now be locked for the requesting user.

SHA_MODE register sets the mode of operation for the SHA.

2’b00 - SHA384 streaming mode
2’b01 - SHA512 streaming mode
2’b10 - SHA384 mailbox mode (Caliptra only, invalid for SoC requests)
2’b11 - SHA512 mailbox mode (Caliptra only, invalid for SoC requests)

10.2 SoC Sender Protocol

Sending data to the SHA Accelerator:

1. Requester queries the accelerator by reading the SHA_LOCK control register.
   - If SHA_LOCK returns 0, SHA_LOCK is granted and will be set to 1.
   - If SHA_LOCK returns 1, it is locked for another device.
2. Requester writes the SHA_MODE register to the appropriate mode of operation.
3. Requester writes the data length in bytes to the SHA_DLEN register.
4. Requester writes data packets to the SHA_DATAIN register until SHA_DLEN bytes are written.
5. Requester writes the SHA_EXECUTE register, this indicates that it is done streaming data.
6. Requesters can poll the SHA_STATUS register for the VALID field to be asserted.
7. Once VALID is asserted, the completed hash can be read from the SHA_DIGEST register.
8. Requester must write 1 to the LOCK register to release the lock.

11 TRNG REQ HW API

For SOCs that choose to not instantiate Caliptra’s embedded TRNG, we provide a TRNQ REQ HW API.

1. Caliptra asserts TRNG_REQ wire (this may be because Caliptra’s internal HW or FW made the request for a TRNG)
2. SOC will write the TRNG architectural registers
3. SOC will write a done bit in the TRNG architectural registers
4. Caliptra desserts TRNG_REQ

Reason to have a separate interface (than using SOC mailbox) is to ensure that this request is not intercepted by any SOC FW agents [which communicate with SOC mailbox]. It is a requirement that this TRNG HW API is always handled by a SOC HW gasket logic (and not some SOC ROM/FW code) for FIPS compliance.

12 SRAM Implementation

12.1 Overview

SRAMs are instantiated at the SOC level. Caliptra provides the interface to export SRAMs from internal components.

SRAM repair logic (eg. BIST) and its associated fuses which are proprietary to companies/their methodologies are done external to the caliptra boundary.

SRAMs must NOT go through BIST/repair flows across a “warm reset”

<FIXME: Any SRAM ECC bit width requirements?>

12.2 RISC-V Internal Memory Export

To support synthesis flexibility and ease Memory integration to various fabrication processes, all SRAM blocks inside the RISC-V core are exported to an external location in the testbench. A single unified interface connects these memories to their parent logic within the RISC-V core. Any memory implementation may be used to provide SRAM functionality in the external location in the testbench, provided it adheres to the interface requirements connected to control logic inside the processor. Memories behind the interface are expected to be implemented as multiple banks of SRAM, from which the RISC-V processor selects the target using an enable vector. The I-Cache has multiple Ways, each containing multiple banks of memory, but I-Cache is disabled in Caliptra and this may be removed for synthesis.

The following memories are exported:

- ICCM
- DCCM

Table 4 indicates the signals contained in the memory interface. Direction is relative to the exported memory wrapper that is instantiated outside the Caliptra subsystem (i.e., from testbench perspective).

12.3 SRAM timing behavior

- [Writes] Input wren signal is asserted simultaneously with input data and address. Input data is stored at the input address 1 clock cycle later.
- [Reads] Input clock enable signal is asserted simultaneously with input address. Output data is available 1 clock cycle later from a flip flop register stage.
- [Writes] Input wren signal is asserted simultaneously with input data and address. Data is stored at the input address 1 clock cycle later.

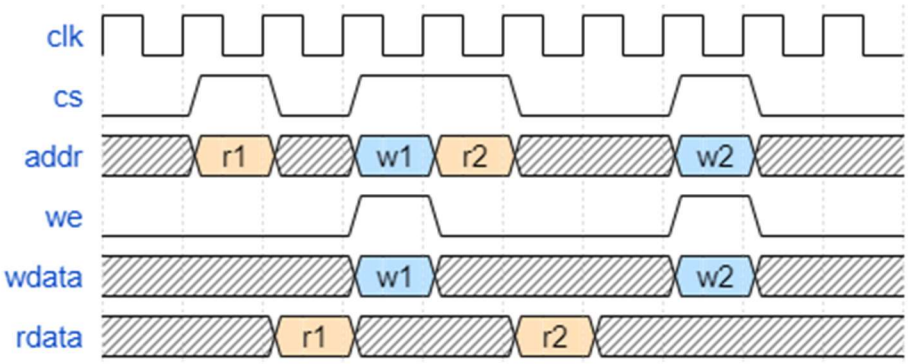

Figure 6: SRAM Interface Timing

12.4 SRAM parameterization

<TODO>: describe dependencies for number of banks/ways

13 Caliptra Subsystem

In order to enable a SOC reusable security solution, we also provide a security subsystem that is a combination of open source & licensable IPs (eg. Analog components such as TRNG, fuse controller).

- SOC Controller has configurable SRAMs to allow per SOC FW.
- RHL (Resource handling logic) is used to bring up subsystem components such as PUF, PLL, Fuse controller, [P]TRNG etc. all and process requests from Caliptra to the outside components (eg. P-TRNG request input to Caliptra's internal TRNG)
- Some of the analog components are licensable IPs that are used to build the subsystem (eg. PUF, PLL)

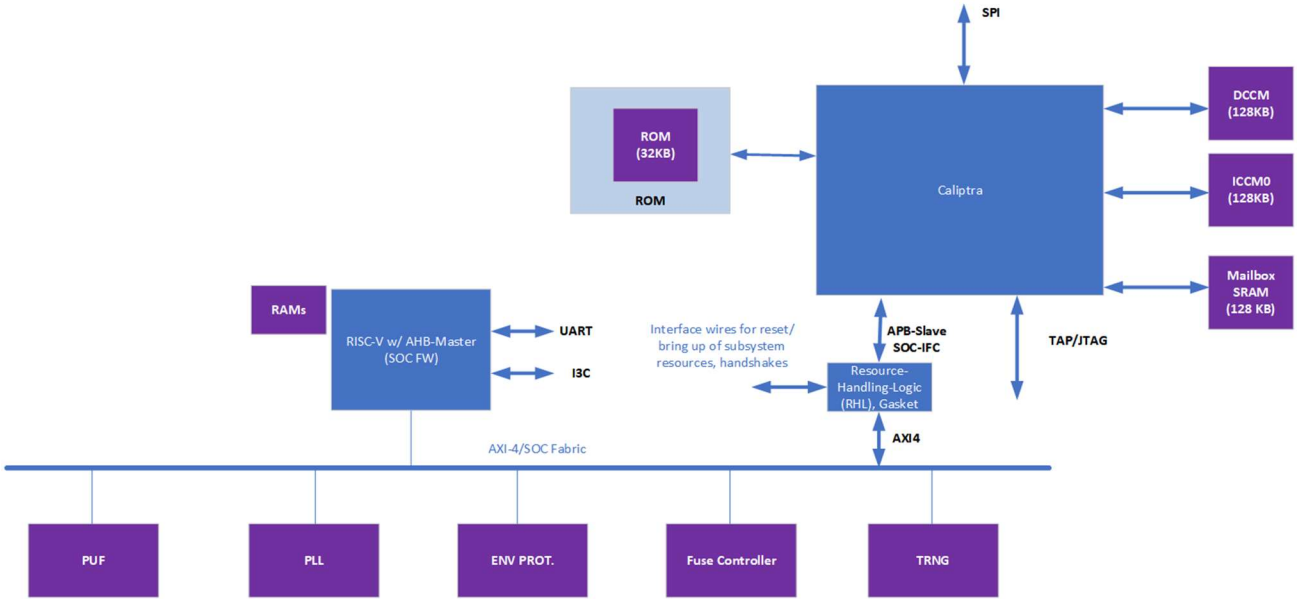

Figure 7: Caliptra Subsystem Block Diagram

14 SOC Integration Requirements

Category	Requirement	Definition of Done	Rationale
Deobfuscation Key	SoC backend flows shall generate Deobfuscation key with appropriate NIST compliance as dictated in the Caliptra ROT specification.	Statement of conformance	Required by UDS & Field Entropy threat model
	If not driven through PUF, SoC backend flows shall ECO the Deobfuscation key before tapeout.	Statement of conformance	Required by UDS & Field Entropy threat model
	Rotation of the deobfuscation key (if not driven through PUF) between silicon steppings of a given product (eg. A0 vs B0 vs PRQ stepping) is dependent on the company specific policies.	Statement of conformance	Required by UDS & Field Entropy threat model

	SoC backend flows should not insert Deobfuscation key flops into the scan chain.	Synthesis report	Required by UDS & Field Entropy threat model
	For defense in depth, it is strongly recommended that debofuscation key flops are not on the scan chain.		Caliptra HW Threat model
CSR Signing Key	SoC backend flows shall generate CSR signing key with appropriate NIST compliance as dictated in the Caliptra ROT specification.	Statement of conformance	Required by IDevID threat model
	SoC backend flows shall ECO the CSR signing key before tapeout.	Statement of conformance	Required by IDevID threat model
	Rotation of the CSR private key between silicon steppings of a given product (eg. A0 vs B0 vs PRQ stepping) is dependent on the company specific policies.	Statement of conformance	
	SoC backend flows should not insert CSR signing key flops into the scan chain.	Synthesis report	Required by IDevID threat model
DFT	If scan is enabled, SoC shall set Caliptra's scan_mode indication to '1.	Statement of conformance	Required by Caliptra threat model
	Caliptra’s TAP should be a TAP endpoint	Statement of conformance	Functional requirement
Mailbox	SoC shall provide an access path between the Mailbox and the application CPU complex on SoCs with such complexes (e.g., Host CPUs, Smart NICs)	Statement of conformance	Required for Project Kirkland and TDISP TSM
Fuses	SoC shall burn non-field fuses during manufacturing. Required vs optional fuses are listed in the architectural specification.	Test on silicon	Required for UDS threat model
	SoC shall expose an interface for burning field fuses. Protection of this interface is up to SoC vendor.	Test on silicon	Required for Field Entropy
	SoC shall write fuse registers and fuse done via immutable logic or ROM code.	Statement of conformance	Required for Caliptra threat model
Security State	SoC shall drive security state wires in accordance with the SoC's security state.	Statement of conforman	Required for Caliptra threat model

		ce	
	If SoC is under debug, then SoC shall drive debug security state to Caliptra.	Statement of conformance	Required for Caliptra threat model
Resets & Clocks	SoC shall start input clock before caliptra_pwrgood assertion.	Statement of conformance	Functional
	SoC reset logic shall assume reset assertions are asynchronous and deassertions are synchronous.	Statement of conformance	Functional
	SoC shall ensure Caliptra's powergood is the SoC's own powergood.	Statement of conformance	Required for Caliptra threat model
TRNG	SoC shall either provision Caliptra with a dedicated TRNG or shared TRNG.	Statement of conformance	Required for Caliptra threat model & Functional
	SoC shall provision the Caliptra embedded TRNG with an entropy source if that is used (vs SOC shared TRNG API support).	Statement of conformance	Functional
	If the TRNG is shared, then upon TRNG_REQ, SoC shall use immutable logic/code to program Caliptra's TRNG registers.	Statement of conformance	Required for Caliptra threat model & Functional
SRAMs	SoC shall ensure timing convergence with 1-cycle read path for SRAMs.	Synthesis report	Functional
	SoC shall size SRAMs to account for SECDED.	Statement of conformance	Functional
	SoC shall write-protect fuses that characterize the SRAM.	Statement of conformance	Required for Caliptra threat model
	SoC shall ensure SRAM content is only destroyed on pwrgood cycling.	Statement of conformance	Functional (Warm Reset, Hitless Update)

	SoC shall only perform SRAM repair on pwrgood events and prior to caliptra_rst_b deassertion.	Statement of conformance	Functional (Warm Reset, Hitless Update)
Backend convergence	Caliptra is validated and backend converged at 400MHz and at process nodes - TSMC 5nm, -- <To be filled accurately>		Functional
Power saving	Caliptra clock gating shall be controlled by Caliptra firmware alone and SOC is provided a global clock gating enable signal (and a register) to control.		Required for Caliptra threat model
	SoC shall not power-gate Caliptra independently of the entire SoC.	Statement of conformance	Required for Caliptra threat model
PAUSER	SoC shall drive PAUSER input in accordance with the IP integration spec.	Statement of conformance	?
Error reporting	SoC shall report Caliptra error outputs.	Statement of conformance	Telemetry & monitoring
	SoC shall only recover Caliptra fatal errors via SoC power-good reset.	Statement of conformance	Required for Caliptra threat model

15 FAQ

15.1 Verilog File Lists

Verilog file lists are generated via VCS and included in the config directory for each unit.
New files added to the design should be included in the vf list, either manually or by utilizing VCS to regenerate the vf file.

16 LINT Rules

TODO 0p5: This is a WIP list

16.1.1 Recommended LINT Rules

The following LINT rules are the recommended minimum set for standalone analysis of Caliptra IP. The same set are recommended as a minimum subset that may be applied by Caliptra Integrators.

Table 6: Recommended Lint Rules

-- Error: "x" in casez statements not allowed -- Error: All instance inputs must be driven -- Error: An event variable is declared but never triggered -- Error: Bit truncation hazard; LHS/RHS truncation of extra bits -- Error: Blocking and Non-blocking assignment to a signal/variable detected -- Error: Case expression width mismatch; Case expression width does not match case select expression width -- Error: Combinational loops detected -- Error: Constant value clock pin of sequential instance -- Error: Detected a logical/scalar operation on a vector -- Error: Detected a tristate is used below top-level of design -- Error: Detected always or process constructs that do not have an event control -- Error: Detected arithmetic comparison operator with unequal length -- Error: Detected conversion of unsigned (reg type) to integer -- Error: Detected floating/unconnected inout port of an instance -- Error: Detected loop step statement variables incorrectly incremented / decremented -- Error: Detected nonblocking assignment in a combinational always block -- Error: Detected reset/set used both synchronously and asynchronously -- Error: Detected signal read inside combinational always block missing from sensitivity list -- Error: Detected tri-state 'Z' or '?' value used in assign or comparison -- Error: Detected two state data type signals; Must support 4 state data type -- Error: Detected undriven but loaded input of an instance -- Error: Detected undriven but loaded net is detected

--
Error: Detected undriven but loaded output port of module
--
Error: Detected undriven output pins connected to instance input
--
Error: Detected unequal length operands in the bit-wise logical, arithmetic, and ternary operators
--
Error: Detected unpacked structure declaration outside the package
--
Error: Duplicate conditions of a case/unique-case/priority-case
--
Error: Function return does not set all bits of return variable
--
Error: Inout port is not read or assigned
--
Error: Instance pin connections must use named-association rather than positional association
--
Error: LHS/RHS mismatch hazard; Multi-bit expression assigned to single bit signal
--
Error: Latch inference not permitted
--
Error: Must declare enum base type explicitly as sized logic type
--
Error: Negative or enum array index detected
--
Error: Non-synthesizable construct; Functions of type real detected
--
Error: Non-synthesizable construct; Repeat statement
--
Error: Non-synthesizable construct; delays ignored by synthesis tools
--
Error: Non-synthesizable construct; modelling style where clock and reset cannot be inferred in sequential inference
--
Error: Non-synthesizable construct; states are not updated on the same clock phase in sequential inference
--
Error: Null Ports detected
--
Error: Port referred before definition
--
Error: Range index or slice of an array discrepancy
--
Error: Read before set hazard in blocking assignment signal
--
Error: Recursive task hazard
--

Error: Redeclaration of a port range
--
Error: Text Macro Redefinition TMR
--
Error: Variable is too short for array index
--
Fatal: Asynchronous reset inference must have "if" statement as first statement in the block
--
Fatal: Blocking assignment detected in sequential always block
--
Fatal: Detected a function or a sub-program sets a global signal/variable
--
Fatal: Detected a function or a sub-program uses a global signal/variable
--
Fatal: Detected assignment to input ports
--
Fatal: Detected edge and non-edge conditions in block sensitivity list
--
Fatal: Detected variable whose both the edges are used in an event control list
--
Fatal: Event control detected in RHS of assignment statement
--
Fatal: Illegal case construct label detected
--
Fatal: Module instance port connection mismatch width compared to the port definition
--
Fatal: Non-synthesizable construct; Case equal operators (===) (!==) operators may not be synthesizable
--
Fatal: Non-synthesizable construct; Detected real operands that are used in logical comparisons
--
Fatal: Non-synthesizable construct; Detected real variables that are unsynthesizable
--
Fatal: Non-synthesizable construct; MOS switches, such as cmos, pmos, and nmos
--
Fatal: Non-synthesizable construct; disable statements detected
--
Fatal: Non-synthesizable construct; event control expressions have multiple edges in sequential inference
--
Fatal: Non-synthesizable construct; event variables
--
Fatal: Non-synthesizable construct; the tri0 net declarations
--
Fatal: Non-synthesizable construct; time declarations
--
Fatal: Non-synthesizable construct; tri1 net declarations

```
--
Fatal: Non-synthesizable construct; trireg declarations
--
Fatal: The 'default' or 'others' must be last case in a case statement
```