OPEN

Compute Project

Caliptra Integration Specification

V0.8

CONTRIBUTORS

Caliptra Consortium

REVISION HISTORY

Date	Revision #	Description
6/20/2022	V0.1	Initial draft
11/11/2022	V0.5	Updated SOC interface ● Interface rules ● Boot FSM ● Mailbox Protocol Described SRAM implementation and SOC integrator instructions for all exported SRAM I/Fs Add Lint Rules
4/10/2023	V0.8	Stabilize interface definitions Add SRAM parameterization requirements Add TRNG Req API
6/20/2023		Add: Example SRAM Machine Check Reliability Integration Add: CDC Constraints Add: DC Data

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

Commented [1]: It looks like we're forking off copies of these IP blocks into subdirectories in the caliptra repo.
1. What's the general methodology to keep open-source implementations up to date when bugs are fixed upstream?
2. Is there a good way to quickly "diff" the changes that we're making in Caliptra vs what is upstream?

Commented [2]: Update to latest VeeR links: https://github.com/chipsalliance/Cores-VeeR

		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 DC Data

IP Name	Date	Path Group	Target Freq	QoR WNS	QoR Achieveable Freq
		CALIPTRACLK	500MHz	-15.93	496MHz
		JTAG_TCK	100MHz	4606.5	100MHz
		clock_gating_default	500MHz	26.56	500MHz
		io_to_io	500MHz	-599.82	385MHz
		io_to_flop	500MHz	0.25	500MHz
CALIPTRA_WRAPPER	6/15/2023	flop_to_io	500MHz	-627.58	381MHz

IP Name	Date	Stdcell Area	Macro Area	Memory Area	Total Area	Flop Count	No Clock Regs/Pins Count	FM Status	FM Eqv Pts	FM Non-Eqv Pts	FM Abort Pts	M Non-Con
CALIPTRA_WRAPPER	6/15/2023	75756	5948	186520	268224	37483	0	FAILED	133398	71	0	0

4 Block Diagram

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

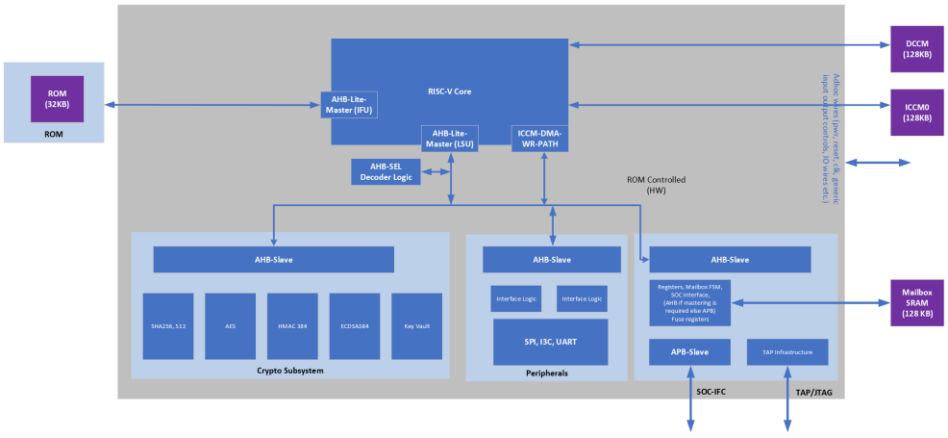


Figure 1: Caliptra Block Diagram

4.1 Boot Media Independent (Passive) vs Boot Media Integrated (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.

5 SOC Interface Definition

TODO 0p5: This is a WIP list

Commented [3]: Swap for updated diagram

Commented [4R3]: done

Commented [5]: Need to rename to
* Boot Media Integrated
* Boot Media Dependent

5.1 Block Diagram

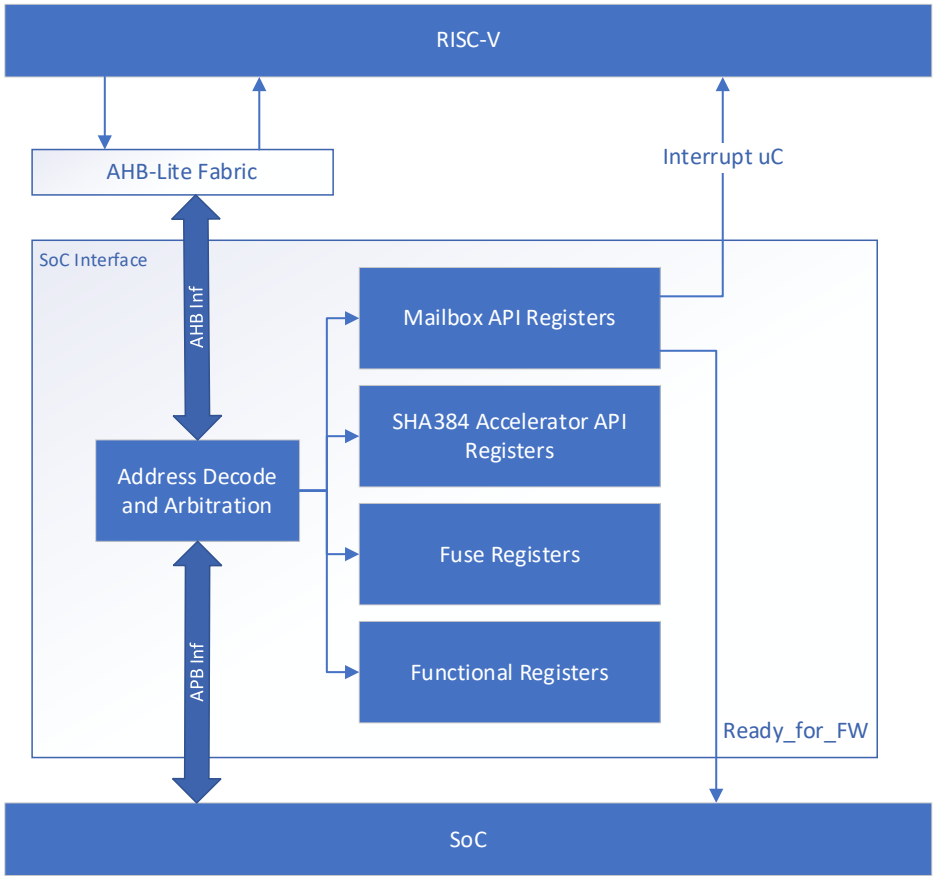


Figure 2: SoC Interface Block Diagram

5.2 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

CPTRA_SET_MBOX_PAUSER_INTEG	5	Each bit hardcodes the valid PAUSER for mailbox at integration time
CPTRA_MBOX_VALID_PAUSER	[4:0][31:0]	Each parameter corresponds to a hardcoded valid PAUSER value for mailbox, set at integration time. Must set above parameter for corresponding valid pauser to be used.
CPTRA_DEF_MBOX_VALID_PAUSER	32	Sets the default valid PAUSER for mailbox accesses. This PAUSER is valid as long as any VALID_PAUSER are unlocked or not set by INTEG parameter.
CPTRA_SET_FUSE_PAUSER_INTEG	1	Sets the valid PAUSER for fuse accesses at integration time.
CPTRA_FUSE_VALID_PAUSER	32	Overrides the programmable valid PAUSER for fuse accesses when above parameter is set to 1

5.3 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		Convergence & 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 is a separate signal than scan chain enable that goes into scan cells). 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.

Formatted: Font color: Auto

5.4 Architectural Registers and Fuses

Control registers and fuses are documented on github.

External Registers -

caliptra_top_reg — caliptra_top_reg Reference (chipsalliance.github.io)

Internal Registers -

clp — clp Reference (chipsalliance.github.io)

5.5 Fuses

Fuses are writable only one time, and require a cptra_pwrgood to be recycled 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 to lock the fuse writes and to proceed with the boot flow.

5.6 Interface Rules

5.6.1 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.

5.6.2 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
- Access to mailbox memory region with invalid PAUSER will be dropped
- PSLVERR is asserted for any of the above conditions

All accesses MUST be 32-bit aligned. Misaligned writes will be dropped, and reads will return 0x0.

5.6.3 Undefined mailbox usages

A trusted/valid requester that locks the mailbox and never releases the lock 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.

5.6.4 Straps

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

5.6.5 Deobfuscation Key

SOC ECO’s the key at the tape-in time of the SOC and must be protected from common knowledge. For a given SOC construction, this can be driven using a PUF too.

It must follow the security rules defined in the arch spec

Commented [6]: @Michael Norris let's add a sub-heading here for reset rules, including a timing diagram and the rules that:
1. pwrgood = 0, rst = 0 at t = 0
2. pwrgood = 1, rst remains constantly 0
3. rst = 1 some time later
4. pwrgood never allowed to return to 0 unless power cycle
5. warm reset is allowed

Commented [7R6]: We can probably use the HW Boot diagram:
Caliptra.vsdx (sharepoint.com)

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

6 SOC Interface Operation

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.

6.1 Boot FSM

Commented [8]: Move to SOC Interface

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.

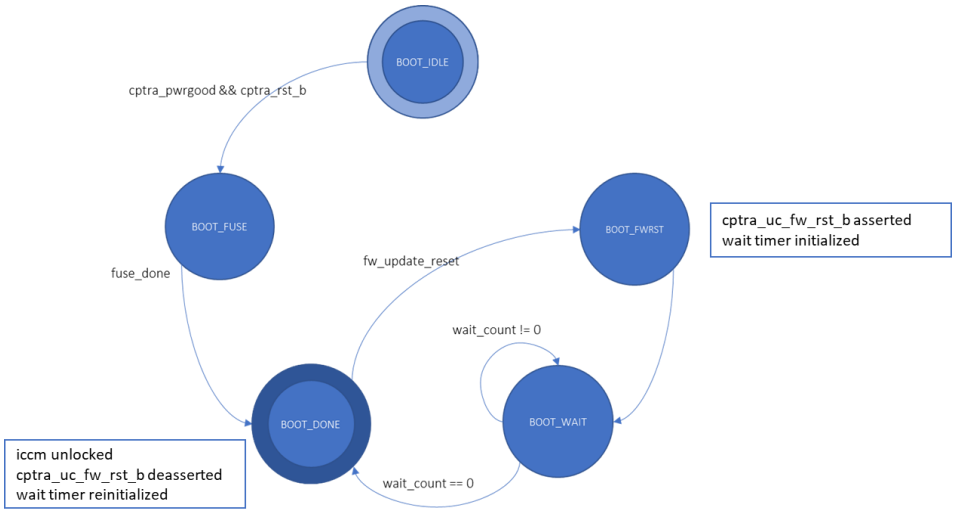


Figure 3: Mailbox Boot FSM State Diagram

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.

6.2 SoC Access Mechanism

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.

6.3 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.

6.4 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 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.

If the SOC violates this protocol, the mailbox flags a protocol violation and enters an error state. Two protocol violations are detected:

1. Access without lock: Writes to any mailbox register by SOC or reads from the dataout register, without having first acquired the lock, are a violation. If any agent currently has the lock, accesses by agents other than the one holding the lock are ignored. If no agent currently has the lock, the aforementioned violations result in a flagged error, leading to the recovery behavior described below.
2. Out of Order Access: SOC must follow the rules herein for Sender and Receiver protocol defining access ordering and progression for a mailbox command. If, after acquiring the lock, an SOC agent performs any register write (or read from the dataout register) outside of the prescribed ordering, this is a flagged violation. Such access by any SOC agent that does not have the lock is ignored.

Once flagged, a mailbox protocol violation is reported to the system several ways:

- The Mailbox FSM will enter the ERROR state in response to an out of order access, which is readable via the Mailbox Status register. Accesses without lock do not result in a state change. The Mailbox may only be restored to the IDLE state by:
    - System reset
    - Write to the force unlock register by firmware inside Caliptra (via internal bus)
- Mailbox protocol violations are reported as fields in the HW ERROR Non-Fatal register. These events also cause assertion of the cptra_error_non_fatal interrupt signal to SOC. Upon detection, SOC may acknowledge the error by clearing the error field in this register via bus write.
- Mailbox protocol violations generate an internal interrupt to the Caliptra microcontroller. Caliptra Firmware is aware of the protocol violation.

Upon entering the ERROR state, the lock signal will only be cleared after the state machine is restored to IDLE as described above.

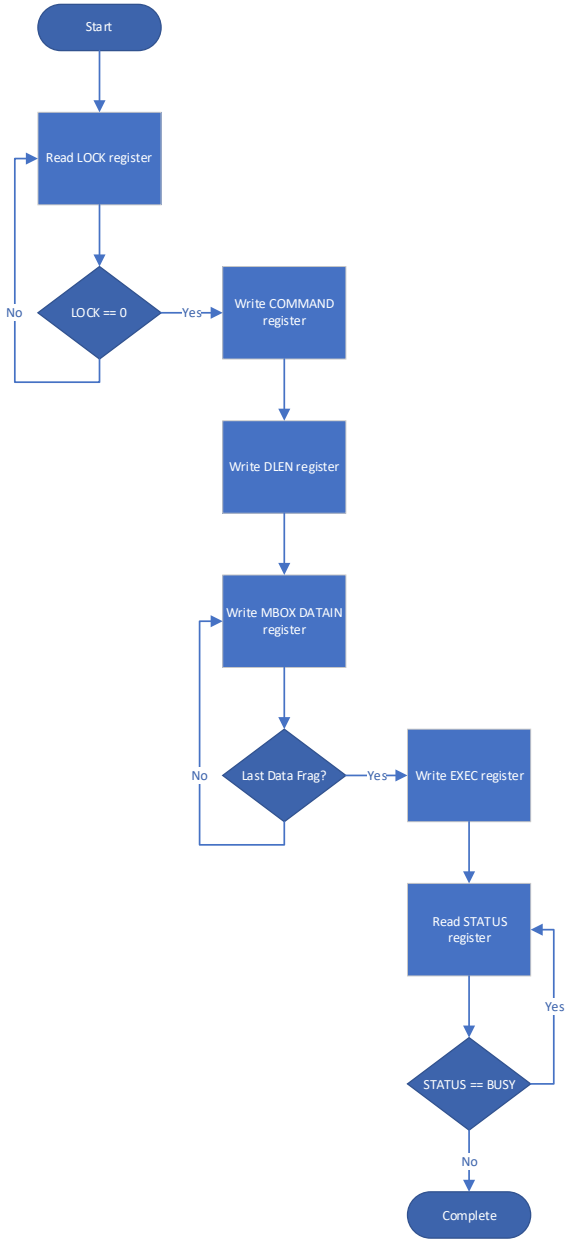


Figure 4: Sender protocol flow chart

6.5 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.

Caliptra will not initiate any mailbox commands that require a response from the SoC. Caliptra initiated mailbox commands are “broadcast” and available to any user on the SoC. SoC will not be able to write the DLEN or DATAIN register while processing a Caliptra initiated mailbox command.

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 CMD register.
4. Receiver reads the MBOX DATAOUT register.
    - Continue reading MBOX DATAOUT register until DLEN bytes are read.
5. If a response is required, the receiver can populate the mailbox with the response by updating the DLEN register and writing to DATAIN with the response.
6. Set the mailbox status register appropriately to hand control back to the sender.
7. The sender will reset the EXECUTE register.
    - This releases the LOCK on the mailbox.

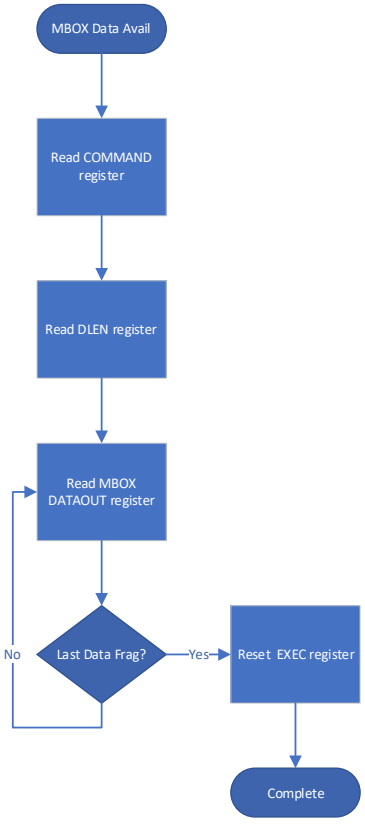


Figure 5: Receiver protocol flowchart

6.6 Mailbox Arbitration

From a mailbox protocol point of view, as long as CPTRA_VALID_PAUSER 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.

There is no fair arbitration scheme between SOC & uController. It is first come - first serve. That said, when the mailbox is locked for uController and SOC has requested for the mailbox (but obviously didn't get it as it is under use), there is an interrupt that gets generated to uController as a notification.

Further, there is no arbitration between various PAUSER attributes. PAUSER attributes exist for security & filtering reasons only.

6.7 MAILBOX PAUSER Attribute Register

- 5 PAUSER attribute registers are implemented at SOC interface
- At boot time, a default SOC/PAUSER can access the mailbox. The value of this PAUSER is an integration parameter CPTRA_DEF_MBOX_VALID_PAUSER.
- The value of CPTRA_MBOX_VALID_PAUSER[4:0] register can be programmed by SOC and once locked becomes a valid PAUSER for accessing the mailbox.
- Alternatively, CPTRA_SET_MBOX_PAUSER_INTEG parameter can be set along with the corresponding CPTRA_MBOX_VALID_PAUSER parameter at integration time.
- SOC logic (ROM, HW) that is using the caliptra mailbox right out of cold reset, should send the mailbox accesses with the default PAUSER, CPTRA_DEF_MBOX_VALID_PAUSER.
- For CPTRA_MBOX_VALID_PAUSER[4:0], the corresponding lock bits MUST be programmed to ‘1 for PAUSER values to be used for mailbox to accept transactions from non-default PAUSERS.
- It is strongly recommended that these PAUSER registers are either set at integration time through integration parameters OR be programmed by the SOC ROM before any mutable FW or ROM patches are applied.

Table 5: PAUSER Register Definition

Register	Description
CPTRA_MBOX_VALID_PAUSER[4:0][31:0]	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.
CPTRA_MBOX_PAUSER_LOCK[4:0]	5 registers, bit 0 of each locks and marks VALID for the corresponding VALID_PAUSER register

6.8 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.

7 SOC SHA Acceleration Block

7.1 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.

(Please see the HW specification for additional details)

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)

7.2 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.

8 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 deasserts 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.

TRNG DATA register is tied to TRNG VALID PAUSER. SOC can program the TRNG VALID PAUSER and lock the register using TRNG_PAUSER_LOCK[LOCK]. This will ensure that TRNG DATA register is RWable by only the PAUSER programmed into the TRNG_VALID_PAUSER register. If the CPTRA_TNRG_PAUSER_LOCK.LOCK is set to '0, then any agent can write to the TRNG DATA register. If the lock is set, only agent with specific TRNG_VALID_PAUSER can write.

9 SRAM Implementation

9.1 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"

Mailbox SRAM is implemented with ECC protection. Data width for the mailbox is 32-bits, with 7 parity bits for a Hamming based SECDED (single-bit error correction and double-bit error detection).

9.2 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).

9.3 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.

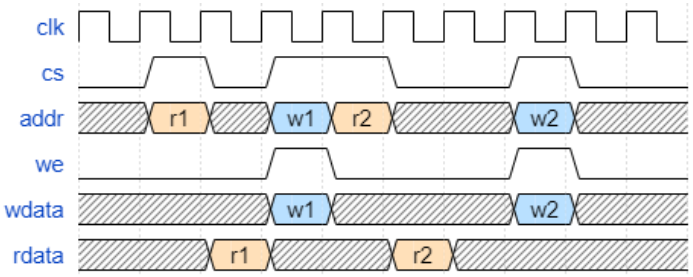


Figure 6: SRAM Interface Timing

9.4 SRAM parameterization

Parameterization for ICCM/DCCM memories is derived from the configuration of the VeeR RISC-V core that has been selected for Caliptra Integration. Parameters defined in the VeeR core determine signal dimensions at the Caliptra top-level interface and drive requirements for SRAM layout. Refer to Section 5.3 for details about interface parameterization. The following configuration options from the RISC-V Core dictate this behavior:

Table 6: SRAM Parameterization

Parameter	Value	Description
ICCM_ENABLE	1	Configures Instruction Closely-Coupled Memory (ICCM) to be present in VeeR core.
ICCM_NUM_BANKS	4	Determines the number of physical 39-bit (32-bit data + 7-bit ECC) SRAM blocks that are instantiated in the ICCM.
ICCM_INDEX_BITS	13	Address bit width for each ICCM Bank that is instantiated.
ICCM_SIZE	128	Capacity of the ICCM in KiB. Total ICCM capacity in bytes is given by 4 * ICCM_NUM_BANKS * $2^{ICCM\_INDEX\_BITS}$.
DCCM_ENABLE	1	Configures Data Closely Coupled Memory (DCCM) to be present in VeeR core.
DCCM_NUM_BANKS	4	Determines the number of physical 39-bit (32-bit data + 7-bit ECC) SRAM blocks that are instantiated in the DCCM.
DCCM_INDEX_BITS	13	Address bit width for each DCCM Bank that is instantiated.
DCCM_SIZE	128	Capacity of the DCCM in KiB. Total DCCM capacity in bytes is given by 4 * DCCM_NUM_BANKS * $2^{DCCM\_INDEX\_BITS}$.

9.5 Example SRAM Machine Check Reliability Integration

Below is an example implementation of Integrator Machine Check Reliability implementation.

Note that the example assumes that data and ecc codes are in non-deterministic bit-position in the exposed SRAM interface bus. Accordingly, redundant correction coding is shown in the integrator level logic (i.e. integrator_ecc(calitpra_data, caliptra_ecc)). If the Caliptra data and ECC are deterministically separable at the Caliptra interface, the integrator would have discretion to store the ECC codes directly and calculate integrator ECC codes for the data alone.

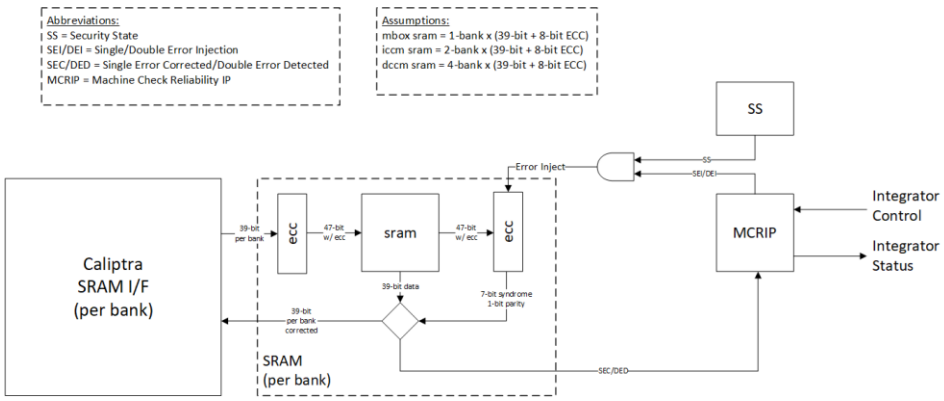


Figure 7: Example Machine Check Reliability Implementation

- Error Detection and Logging

1. Caliptra IP shall interface to ECC protected memories,
2. Caliptra IP calculates and applies its own ECC code, which produces a total of 39-bit data written to external/AMD instantiated SRAMs,
3. Each 39-bit bank memory internally calculates 8-bit ECC on a write and stores 47-bit of data with ECC into SRAM,
4. On read access syndrome is calculated based on 39-bit data,
5. If parity error is detected and syndrome is valid then the error deemed single-bit and correctable,
6. If no parity error is detected but syndrome == 0 OR syndrome is invalid the error is deemed uncorrectable,
7. On both single and double errors the read data is modified before being returned to Caliptra,
8. Since single-bit errors shall be corrected through AMD instantiated logic, Caliptra will never see single bit errors from SRAM,
9. Double-bit or uncorrectable errors would cause unpredictable data to be returned to Caliptra, since this condition shall be detected and reported to MCRIP, there is no concern or expectation that Caliptra will operate correctly after double error,

10. On detection single errors are reported as transparent to MCRIP, double errors are reported as fatal,
11. Along with error severity MCRIP logs physical location of the error,
12. Once MCRIP logs an error it has a choice to send out in-band notification to an external agent,
13. MCRIP logs can be queried by SOC software.

- Error Injection

1. MCRIP supports two error injection modes,
2. Intrusive error injection can force single or double error to be injected, which would result in incorrect data to be returned on read access,
3. The intrusive error injection mode is disabled in Production fused parts via Security State signal,
4. Non-intrusive error injection allows external software to write into MCRIP error log registers,
5. The non-intrusive error injection does not interfere with the operation of memories,
6. The non-intrusive error injection is functional in Production fused parts.

- Caliptra Error/Recovery Flow

1. Caliptra Stuck:
    a. SOC BC timeout mechanism with 300us timeout.
2. Caliptra reports non-fatal Error during boot flow:
    a. cptra_error_non_fatal is an output Caliptra signal, which shall be routed to SOC interrupt controller,
    b. SOC can look at Caliptra Non Fatal Error register for error source,
    c. Assume Caliptra can report a non-fatal error at any time,
    d. SOC should monitor the error interrupt or check it before sending any mailbox command,
    e. In the event of a non-fatal error during boot (i.e. prior to Ready for RT signal) SOC should enter recovery flow and attempt to boot again using alternate boot part/partition,
    f. If SOC sees that a non-fatal error has occurred AFTER receiving the Ready for RT signal, SOC may attempt to recover Caliptra by executing the “Run Self-Test” mailbox command (not yet defined),
    g. If this command completes successfully, SOC may continue using Caliptra as before,
    h. If this command is unsuccessful, Caliptra is in an error state for the remainder of the current boot.
    i. Non-fatal ECC errors will never be reported by Caliptra, SOC needs to monitor MCRIP for non-fatal Caliptra ECC errors.
3. Caliptra reports Fatal Error during boot flow:
    a. cptra_error_fatal is an output Caliptra signal, which shall be routed to SOC interrupt controller,
    b. SOC can look at Caliptra Fatal Error register for error source,
    c. Assume Caliptra can report a Fatal Error at any time,
    d. Fatal errors are generally hardware in nature. SOC may attempt to recover by full reset of the entire SoC, or simply move on and know that Caliptra will be unavailable for the remainder of the current boot,
    e. We cannot assume that uncorrectable errors will be correctly detected by Caliptra, ECC Fatal errors shall be reported by SOC MCRIP.

SOC Integration Requirements

Table 7: 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	Before scan is enabled (separate signal that SOC will implement on scan insertion), SoC shall set Caliptra's scan_mode indication to '1 to allow secrets/assets to be flushed.	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). Please refer to section **Sender Protocol** for details regarding error conditions.	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 conformance	Required for Caliptra threat model
	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	Statement of	Required for Caliptra threat

	TRNG registers.	conformance	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
TRNG PAUSER Programming rules	• If SOC doesn't program the CPTRA_TRNG_PAUSER_LOCK[LOCK] , then Caliptra HW will NOT accept TRNG data from any SOC entity. • If SOC programs CPTRA_TRNG_VALID_PAUSER and sets	Security	Required for Caliptra threat model

	CPTRA_TRNG_PAUSER_LOCK[LOCK] , then Caliptra HW will accept TRNG data only from the entity that is programmed into the PAUSER register. ● It is strongly recommended that these PAUSER registers are either set at integration time through integration parameters OR be programmed by the SOC ROM before any mutable FW or ROM patches are absorbed.		
MAILBOX PAUSER programming rules	● 5 PAUSER attribute registers are implemented at SOC interface ● At boot time, a default SOC/PAUSER can access the mailbox. The value of this PAUSER is an integration parameter CPTRA_DEF_MBOX_VALID_PAUSER. ● The value of CPTRA_MBOX_VALID_PAUSER[4:0] register can be programmed by SOC and once locked becomes a valid PAUSER for accessing the mailbox. ● Alternatively, CPTRA_SET_MBOX_PAUSER_INTEG parameter can be set along with the corresponding CPTRA_MBOX_VALID_PAUSER parameter at integration time. ● SOC logic (ROM, HW) that is using the caliptra mailbox right out of cold reset, should send the mailbox accesses with the default PAUSER, CPTRA_DEF_MBOX_VALID_PAUSER. ● For CPTRA_MBOX_VALID_PAUSER[4:0], the corresponding lock bits MUST be programmed to '1 for PAUSER values to be used for mailbox to accept transactions from non-default PAUSERS. ● It is strongly recommended that these PAUSER registers are either set at integration time through integration parameters OR be programmed by the SOC ROM before any mutable FW or ROM patches are applied.	Security	Required for Caliptra threat model

10 FAQ

10.1 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.

11 CDC Analysis and Constraints

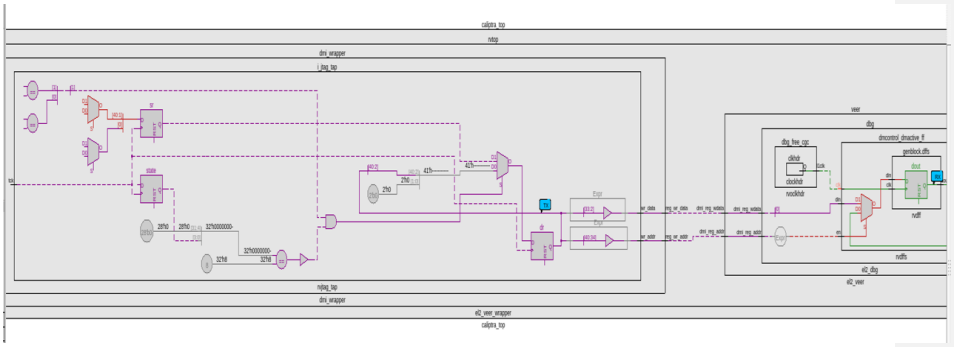
```
=============================================================
Section 3 : CDC Results
=============================================================
CDC Summary
=============================================================
Violations (47)
-------------------------------------------------------------
Single-bit signal does not have proper synchronizer.         (27)
Multiple-bit signal across clock domain boundary.            (20)
```

11.1 Analysis of Missing Synchronizers

- All the signals whether single bit or multi-bit are originating from rvjtag_tap module internal register on TCK clock and Sink/Endpoint is rvdff register which are in CalitpraClockDomain clock.
- JTAG does a series of “jtag writes” for each single “register write”.
- We only need to synchronize the controlling signal for this interface.
- Inside the dmi_wrapper, the dmi_reg_en and dmi_reg_wr_en comes from dmi_jtag_to_core_sync which is 2FF synchronizer.

```
345    rvdffs #(4) dmcontrolff (.din({dmi_reg_wdata[31:30],dmi_reg_wdata[28],dmi_reg_wdata[1]}),
.dout({dmcontrol_reg[31:30], dmcontrol_reg[28], dmcontrol_reg[1]}), .en(dmcontrol_wren),
.rst_l(dbg_dm_rst_l), .clk(dbg_free_clk));
346    rvdffs #(1) dmcontrol_dmactive_ff (.din(dmi_reg_wdata[0]), .dout(dmcontrol_reg[0]),
.en(dmcontrol_wren), .rst_l(dbg_rst_l), .clk(dbg_free_clk));
347    rvdff  #(1) dmcontrol_wrenff(.din(dmcontrol_wren), .dout(dmcontrol_wren_Q),
.rst_l(dbg_dm_rst_l), .clk(dbg_free_clk));
```

```
341    assign dmcontrol_wren      = (dmi_reg_addr ==  7'h10) & dmi_reg_en & dmi_reg_wr_en;
```



11.2 CDC Analysis Conclusions

- Missing synchronizers appear to be the result of “inferred” and/or only 2-FF instantiated synchronizers.
  - dmi_jtag_to_core_sync.v contains inferred 2FF synchronizers on the control signals “dmi_reg_wr_en” and “dmi_reg_rd_en”.
  - 2FF synchronizer inferences are considered non-compliant and should be replaced by explicitly instantiated synchronization module which is intended to be substituted on a per-integrator basis.
    - cdc report scheme two_dff -severity violation
- Multi-bit signals are effectively pseudo-static and are qualified by synchronized control qualifiers.
  - Psuedo-static: wr_data, wr_addr
    - cdc signal reg_wr_data -module dmi_wrapper -stable
    - cdc signal reg_wr_addr -module dmi_wrapper -stable

11.3 CDC Constraints

- cdc report scheme two_dff -severity violation
- cdc signal reg_wr_data -module dmi_wrapper -stable
- cdc signal reg_wr_addr -module dmi_wrapper -stable

Commented [9]: @Norman.Stewart@amd.com - is module already available in the repository or should it be integrator specific? @howardtr@google.com - FYI. _Assigned to norman.stewart@amd.com_

Commented [10R9]: Hi Matt. Both VeeR and Caliptra are expected to have "placeholder" synchronizer modules which will be substituted by integrator specific equivalents. This is captured in issue here: https://github.com/chipsalliance/Cores-VeeR-EL2/issues/51

12 LINT Rules

TODO 0p5: This is a WIP list

12.1.1 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 8: 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