Power Matters.™

RISC-V Soft Processor on PolarFire

What is RISC-V

- A New free and open ISA developed at UC Berkeley
  - RISC-V is “Instruction set architecture” (ISA). Not a processor.
  - The micro architecture implementations can be open or proprietary
  - Can expand to have open specifications for platforms, accelerators etc.
  - Permits reuse of the Software across many micro architectures
  - Goal is to encourage both open-source and proprietary implementations of the RISC-V ISA specification

- Designed for
  - Research, Education & Commercial use

- RISC-V foundation
  - Directs future development and foster adoption
  - https://riscv.org/

Frozen RISC-V Base & Extensions

- Four base integer ISA Variants
  - RV32I, RV64I, RV32E, RV128I (32, 64, 128 bit machines)
  - I : Base Integer instruction set

- Standard extensions - All frozen in 2014 – Forever
  - M: Integer multiply/divide
  - A: Atomic memory operations
  - F: Single Precision Floating Point
  - D: Double Precision Floating Point
  - G: IMAFD, General purpose ISA
  - Q: Quad Precision Floating Point
  - C: 16-bit Compressed instruction (RV32C, RV64C)

- Future extension
  - V: Vector extensions – in definition

RISC-V Privileged Architecture

- Four privilege modes
  - User mode – U
  - Supervisor mode – S
  - Hypervisor mode – H // not specified yet
  - Machine mode – M

- Supported combinations

M	// simple embedded
M, U	// embedded with protection
M, S, U	// operating systems
M, H, S, U	// systems running hypervisors

- Expect ratification CQ2

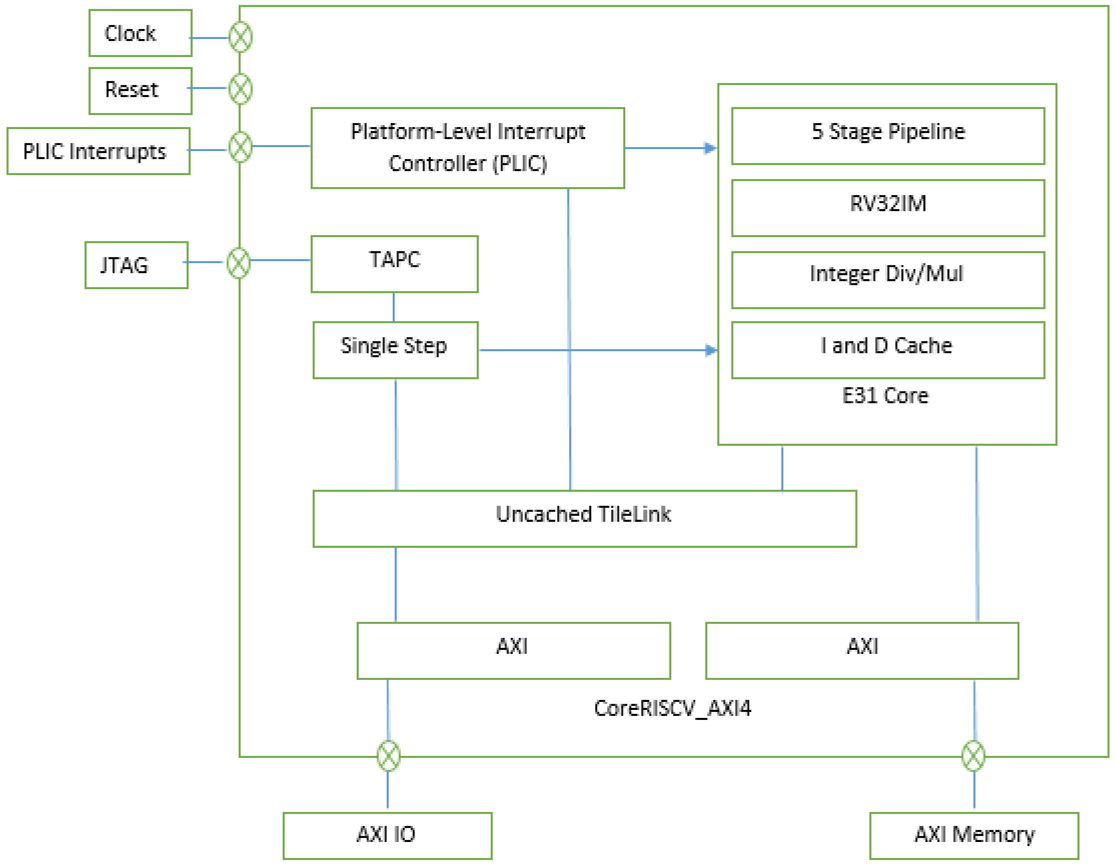
RISC-V Soft Processor on PolarFire

- CoreRISCV_AXI4
  - Based on SiFive's Coreplex E31 design
  - Will be available through Libero Catalog
    - IP Core
    - CoreRISCV_AXI4 Handbook
    - CoreRISCV_AXI4 release note
    - RISC-V privileged ISA spec
    - RISC-V user level ISA Spec

- Key Features
  - Supports the RISC-V standard RV32IM ISA
  - Integrated 8Kbytes instructions cache and 8 Kbytes data cache
  - Two external AXI interfaces for IO and memory
  - Support up to 31 programmable interrupts
  - Debug unit with a JTAG interface
  - Best suited for low-mid range Microcontroller applications

CoreRISCV_AXI4

- Block Diagram



CoreRISCV_AXI4

- CoreRISCV_AXI4 Processor Core
  - Based on the E31 Coreplex Core by SiFive
  - Provides a single hardware thread (*hart*) (one PC + one set of Registers)
  - Machine-mode privileged architecture
  - Supports the RISCV standard RV32IM ISA
    - "I" stands for "Base Integer"
    - "M" stands for "Integer Multiplication and Division"

CoreRISCV_AXI4

- Two External AXI interfaces
  - AXI memory interface
    - Cached access to instruction and data memory
  - AXI I/O interface
    - Un-cached accesses to I/O peripherals

- Memory System
  - First-level instruction cache
    - The instruction cache is 8 KB
    - Direct-mapped with a 64 bytes line size.
    - One Clock cycle Access latency.
  - First-level data cache
    - The data cache size is 8 KB,
    - Direct-mapped with a line size of 64 bytes.
    - The access latency is two clock cycles for full words and three clock cycles for smaller quantities.
  - Un-cached memory access for I/O

CoreRISCV_AXI4

Interrupt Sources

- Local Interrupts
  - Wired directly to the CPU internally
  - Standard software interrupts (*Exceptions , traps*)
  - Timer interrupt

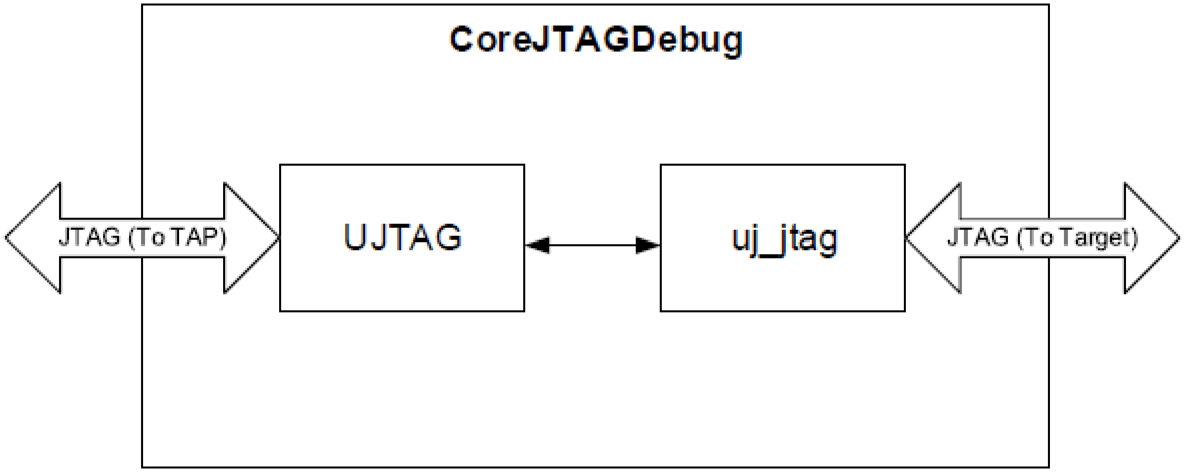
- Global interrupts
  - Routed via Platform-Level Interrupt Controller(PLIC)
  - Supports up to 31 external interrupt sources
  - All external interrupts are single priority level at priority 1.
  - Supports Level triggered interrupts
  - External interrupts in the system can be connected here.

CoreRISCV_AXI4

- JTAG Interface
  - Industry-standard 1149.1 JTAG interface
  - Supports Interactive debug
  - Supports Hardware breakpoints (maximum 2)
  - Accessible via FlashPro5 JTAG programmer/Debugger
  - Unlike CoreCortexM1, UJTAG module is not part of CoreRISCV_AXI4
  - Use CoreJTAGDebug for JTAG access

CoreJTAGDebug

- Key Features
  - Facilitates JTAG interface to soft processor CoreRISCV_AXI4
  - Provides fabric access to the JTAG interface
  - Configurable IR Code support for JTAG tunneling
    - Allows debug support for multiple soft-processors
  - Future version is aimed towards debug support for multiple soft-processors



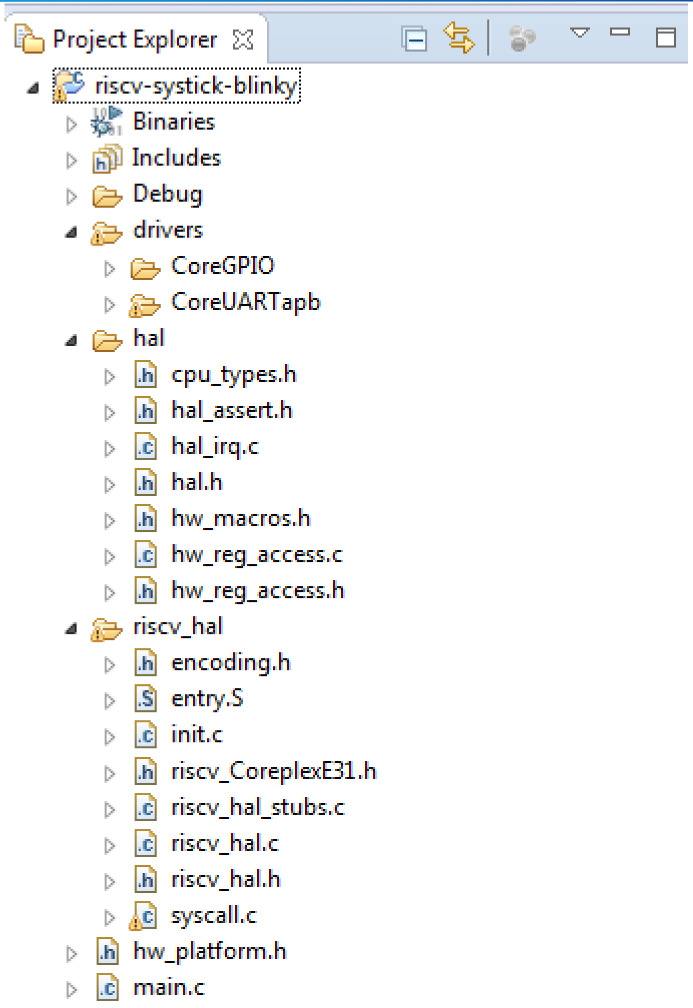
CoreRISCV_AXI - SoftConsole

- SoftConsole 5.0
  - Installer available here
  - Works with FlashPro5.
  - Supported platforms
    - Ubuntu 14.04.5 LTS Desktop 32 and 64 bit
    - Ubuntu 16.04.1 LTS Desktop 32 and 64 bit
    - CentOS/Red Hat Enterprise Linux 6.8 Desktop 32 and 64 bit
    - CentOS/Red Hat Enterprise Linux 7.2 Desktop 64 bit
    - OpenSuse Leap 42.1 [Gnome/Kde desktop] 64 bit
  - Platforms not supported yet
    - Linux distributions/versions other than those listed above
    - Microsoft Windows
    - Virtual machines
  - Release note available here
    - Installation instruction
    - Known issues

CoreRISCV_AXI - SoftConsole

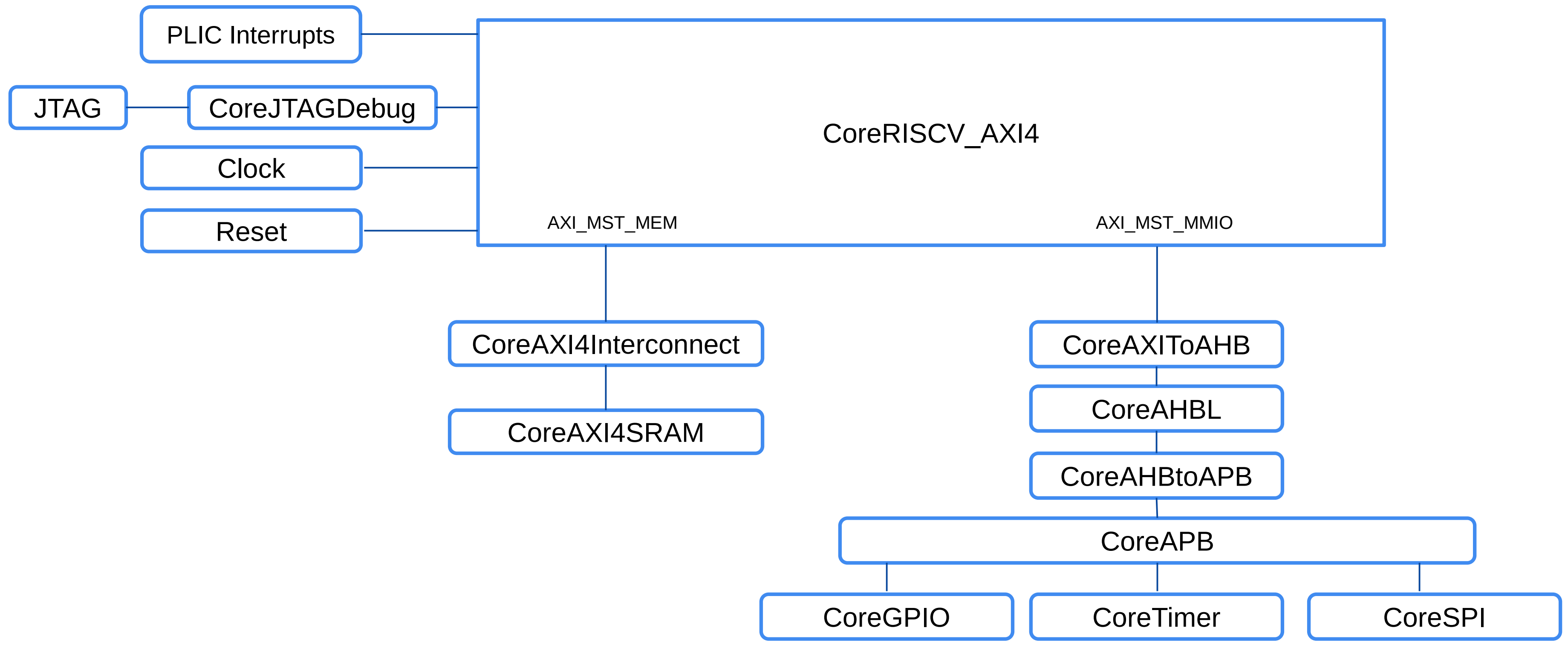
- Firmware project structure
  - hal
    - Hardware abstraction layer
    - Same as CortexM3 HAL on SmartFusion2
    - Register accesses e.g. HAL_set_32bit_reg
  - riscv_hal
    - RISCV Startup code
    - PLIC control
    - Equivalent to CMSIS on CortexM3 HAL on SmartFusion2
  - drivers
    - user peripheral drivers e.g. CoreSPI, CoreGPIO

- riscv_hal is available on github
  - https://github.com/RISCV-on-Microsemi-FPGA/riscv-hal


Project Explorer

```
riscv-systick-blinky
    Binaries
    Includes
    Debug
    drivers
        CoreGPIO
        CoreUARTapb
    hal
        cpu_types.h
        hal_assert.h
        hal_irq.c
        hal.h
        hw_macros.h
        hw_reg_access.c
        hw_reg_access.h
    riscv_hal
        encoding.h
        entry.S
        init.c
        riscv_CoreplexE31.h
        riscv_hal_stubs.c
        riscv_hal.c
        riscv_hal.h
        syscall.c
    hw_platform.h
    main.c
```

CoreRISCV_AXI4 system on PolarFire



CoreRISCV_AXI4 – Booting the system

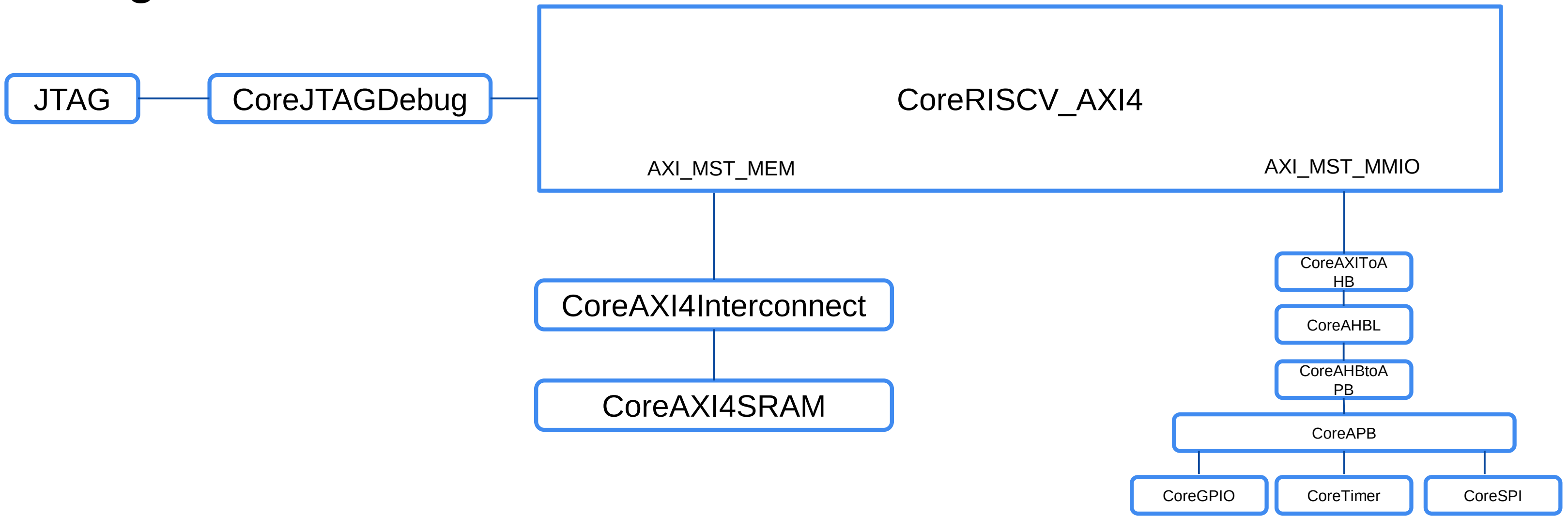
- Boot options
  - Debug Mode
    - Executing code from LSRAM using JTAG

  - Release mode
    - Design Initialization using PolarFire System Controller
      - Image can reside in sNVM, uPROM, or SPI-Flash
      - At power-up the image is loaded into LSRAM.
      - Image is then executed out of LSRAM.
      - Able to select locations post P&R

    - CoreBootStrap
      - Hardware boot-loader
      - Loads image into LSRAM from SPI-Flash using soft logic

Booting CoreRISCV_AXI4 on PolarFire

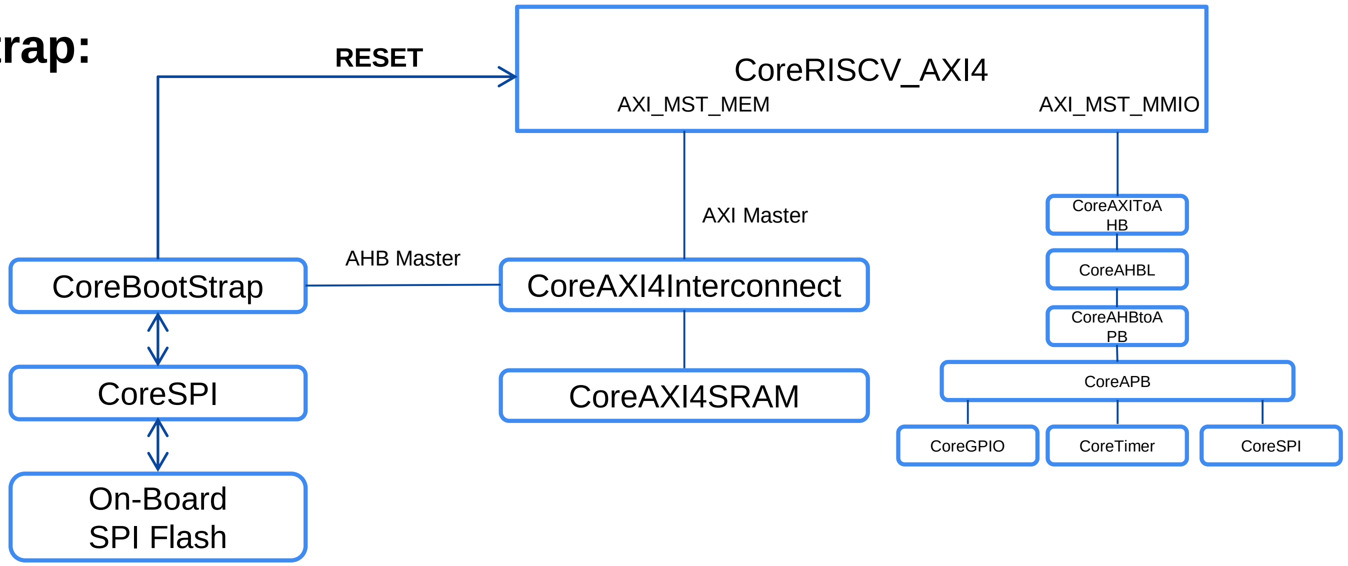
❑ **Debug Mode:**



- FlashPro5 connected to JTAG port for interactive debug
- Download image to CoreAXI4SRAM connected on the CoreRISCV_AXI4 code memory space
- After reset CoreRISCV_AXI4 fetches and executes code from CoreAXI4SRAM

Booting CoreRISCV_AXI4 on PolarFire

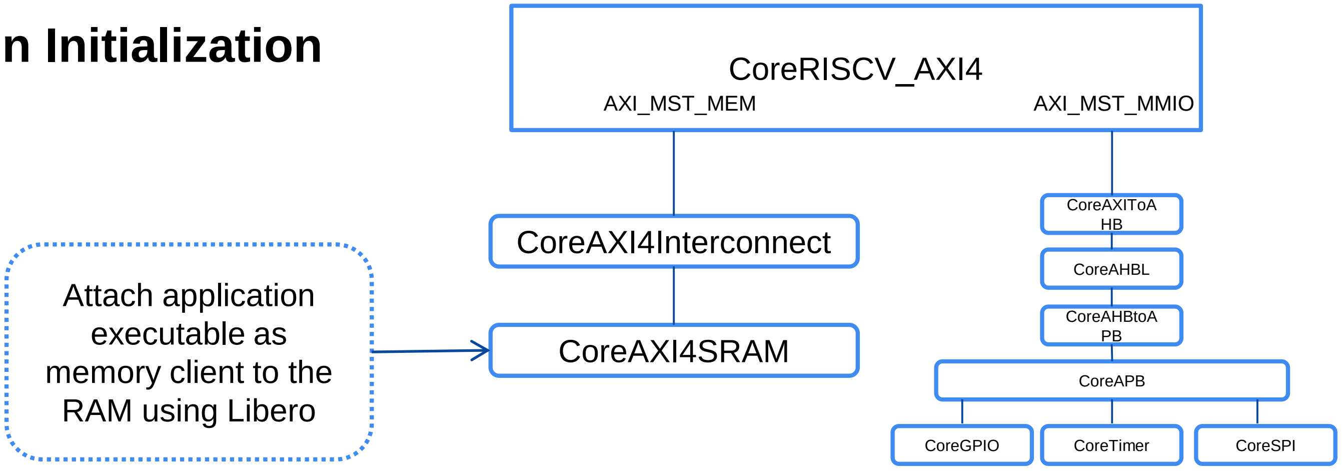
- CoreBootstrap:



- Connect CoreAXI4SRAM on code memory space of CoreRISCV_AXI4 memory map.
- Use CoreBootStrap IP as first level Boot loader
- The CoreBootStrap Holds the CoreRISCV_AXI4 in reset while it copies the application executable from on-board SPI Flash memory to CoreAXI4SRAM after system reset.
- Application executable should be pre-loaded on the SPI Flash memory.
- Releases reset on CoreRISCV_AXI4 after completion of copying executable to CoreAXI4SRAM
- CoreRISCV_AXI4 now fetches and executes code from CoreAXI4SRAM.

Booting CoreRISCV_AXI4 on PolarFire

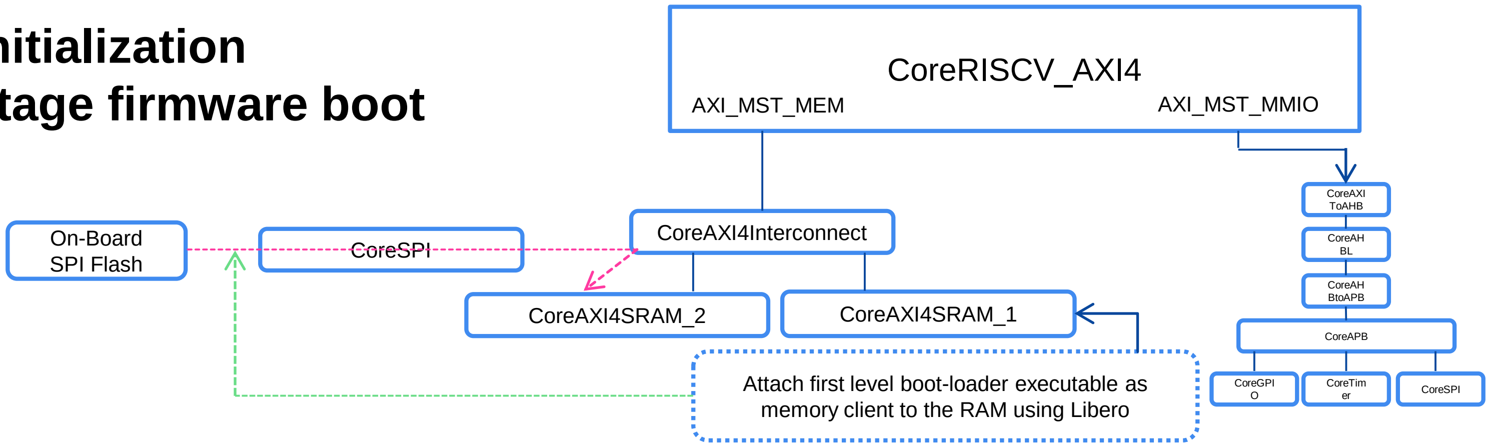
❑ Design Initialization



- Connect CoreAXI4SRAM on code memory space of CoreRISCV_AXI4 memory map.
- Use the configurator of CoreAXI4SRAM to attach a memory initialization file to it. This file should contain the application executable.
- Use the Libero “Design and memory Initialization” GUI post P&R
- The memory client will be stored on chosen memory, e.g. sNVM, uPROM etc. Memory client will be part of bit-stream output of the Libero design.
- On system reset, System Controller will initialize the destination CoreAXI4SRAM with memory client from chosen memory.
- The CoreRISCV_AXI4 will now execute application executable from CoreAXI4SRAM after it comes out of reset.

Booting CoreRISCV_AXI4 on PolarFire

❑ Design Initialization
– two stage firmware boot



- Connect CoreAXI4SRAM_1 in the code memory space of CoreRISCV_AXI4 memory map.
- Use the configurator of CoreAXI4SRAM_1 to attach a memory initialization file to it. This file should contain the first level boot loader executable code.
- Use the Libero “Design and memory Initialization” GUI post P&R
- The memory client will be stored on chosen memory, e.g. sNVM, uPROM etc. Memory client will be part of bit-stream output of the Libero design.
- At system initialization time, System Controller will initialize the destination CoreAXI4SRAM_1 with memory client from chosen memory.
- The CoreRISCV_AXI4 will now execute first level boot loader from CoreAXI4SRAM_1 after it comes out of reset.
- The boot loader functionality is to copy application executable from on-board SPI Flash memory to CoreAXI4SRAM_2.
- After copy, the CoreRISCV_AXI4 jumps to address of CoreAXI4SRAM_2 to start executing the application image.
- Application executable should be pre-loaded on the SPI Flash memory.
- The base addresses of CoreAXI4SRAM_1, CoreAXI4SRAM_2 and the linker scripts of both executable codes must be coordinated to point to correct memory map.

Thank you