PolarFire® SoC Secure Boot Methodologies

Presenter: Johnny Kim – Senior Embedded Solutions Engineer

MICROCHIP

Agenda

- Threats
- What we mean about security
- PolarFire® SoC Start-up
- Secure Boot Authentication Framework

Supply Chain and Equipment Threats

Insiders
Industrial Espionage
Criminal Profiteers
Nation-States

Trojan Horse in Hard IP
Trojan Horse in IC Design
Insert Trojan in Mask

Trojan Horse in Soft IP
Trojan Horse in FPGA
Modified EDA Tools

Side Channel Analysis
3rd Party Clones
Tampering

Chip Manufacturer → Gray Market → OEM → System User

Overbuild & Steal Wafers
Load Wrong Keys
Sell Failed Devices

Re-mark packages
Refurbish Used Parts
Steal Finished Goods

Load Wrong Keys
Load Wrong Configuration
Overbuild Equipment
Reverse Engineer

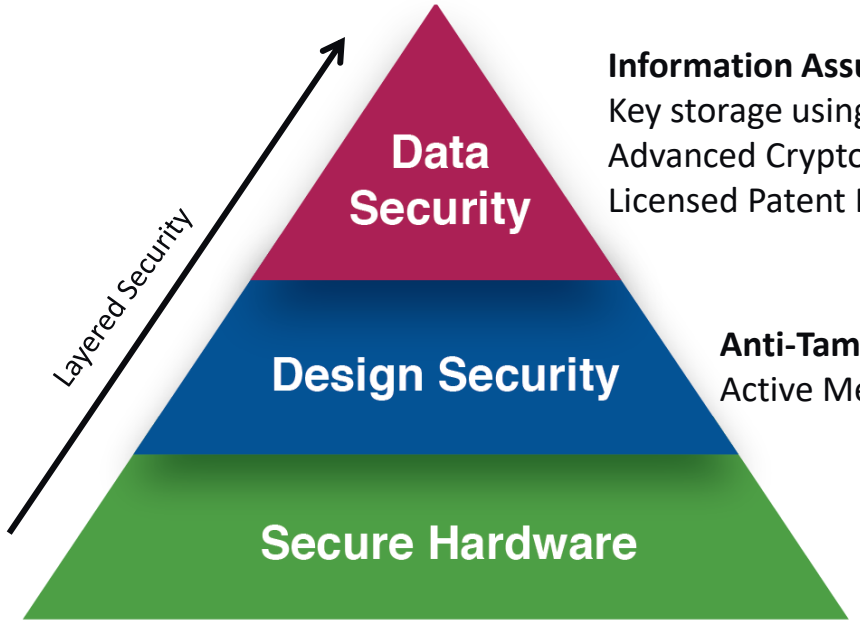
Fielded Systems

Secure Hardware and Trust | Design Security & Anti-Tamper | Data Security & Information Assurance

If your Supply Chain is not secure how can your systems be?

Security is All About Layers

Secure Systems need
Secure Hardware, Design Security and Data Security



Information Assurance:
Key storage using Physically Unclonable Function (PUF)
Advanced Crypto Accelerators
Licensed Patent Protected DPA Resistance Pass through License

Anti-Tamper: Secure Bitstream, Tamper Detection, Active Mesh, No Copying. Cloning, or Reverse Engineering

Trust: Licensed Patent Protected DPA Resistance, NIST Certified Crypto Accelerators, Secure Supply Chain

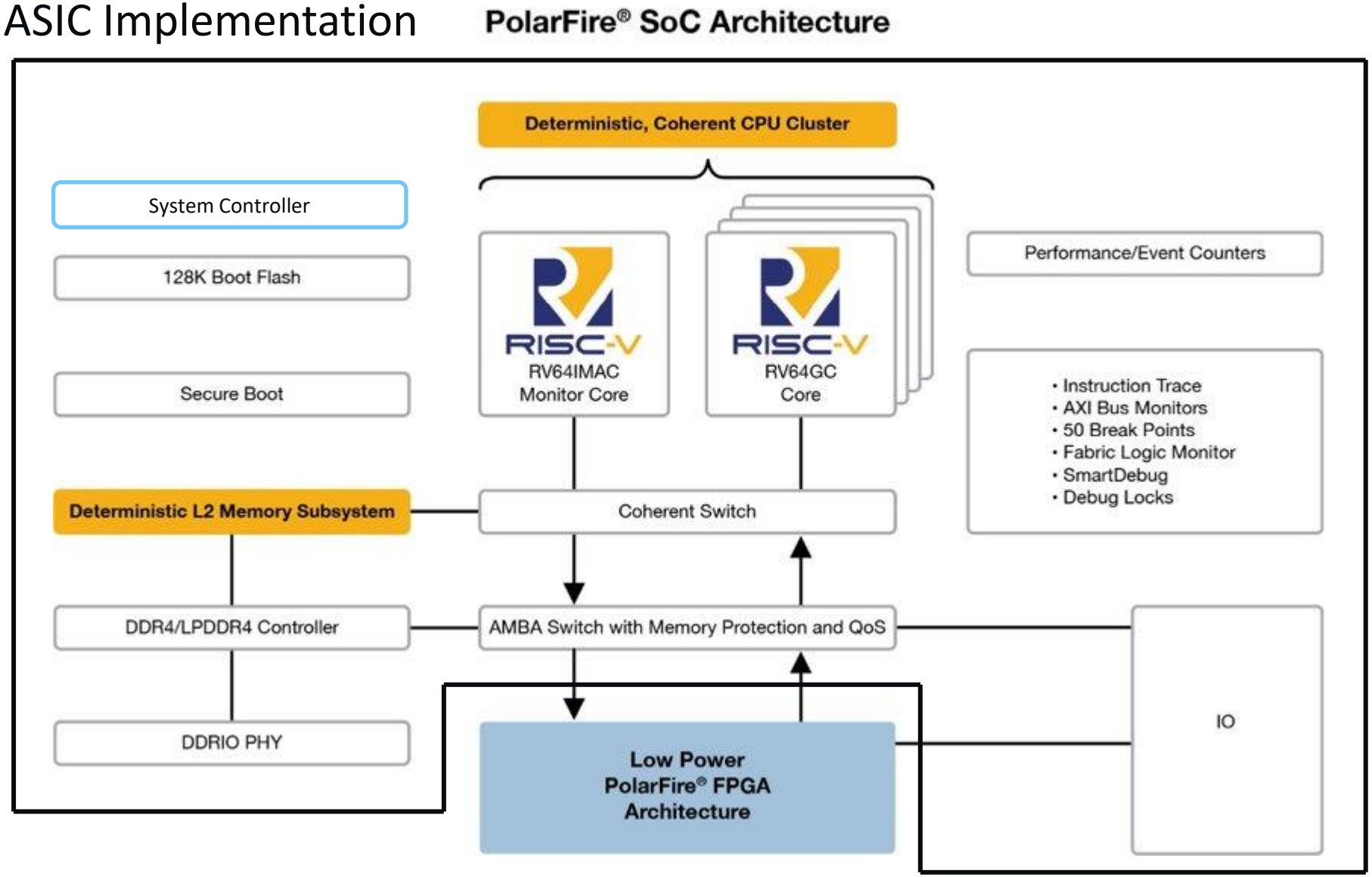
LICENSED DPA COUNTERMEASURES™

Microchip FPGAs provide a solid foundation for building your security applications

MICROCHIP

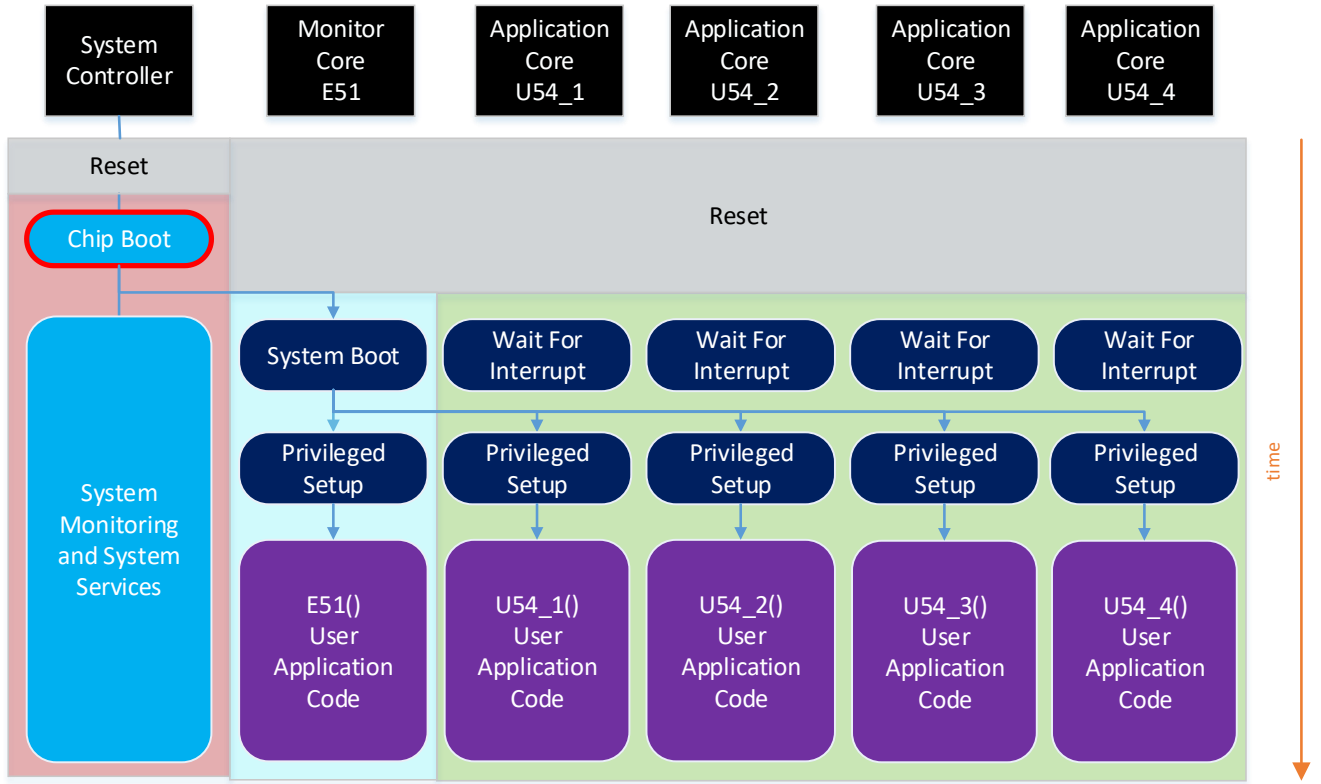
PolarFire® SoC

Low Power ASIC Implementation



PolarFire® SoC Start-up Overview

- **The System Controller is the first part of the system to come out of reset**
  - Powers up the FPGA (<10 ms)
  - Performs low level initialization of PolarFire SoC
  - Configures the reset vector of the RISC-V processors
    - All RISC-V processor cores execute the exact same code coming out of reset
  - Releases the RISC-V processors from reset
  - Total Time ~ 10.2 ms



MICROCHIP

System Controller

- **Decrypts bitstream in a DPA resistant manner**
- **Sets tamper alarms**
- **Provides system services**
  - Programming, PUF services, R/W to sNVM, Device ID, Device Certificate, others
- **Protected by active mesh, scrambled busses, others**
- **Powers up the device in an orderly fashion**

Boot Mode Parameters

U_MSS_BOOTMODE

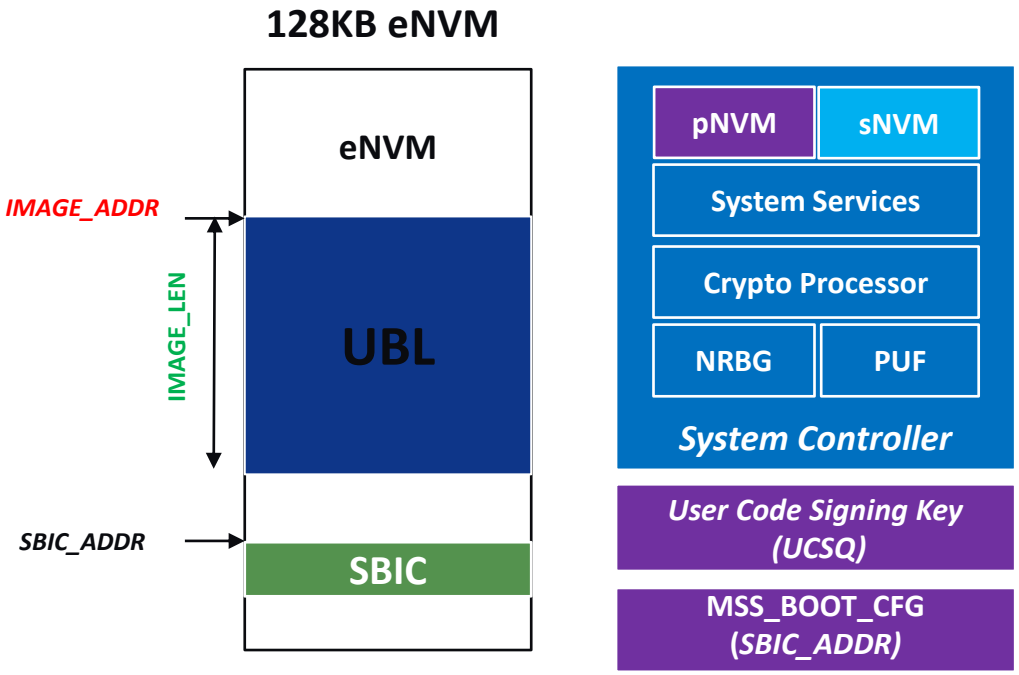
	Mode	Description
0	Idle	Core complex idles
1	Non-secure boot	Core complex boots directly from address defined by U_MSS_BOOTADDR
2	User boot loader	Core complex boots using code copied from sNVM
3	Factory boot loader	Core complex boots using the factory secure boot protocol

MSS_BOOT_CFG when U_MSS_BOOT_MODE = 3

Offset (bytes)	Size (bytes)	Name	Description
0	4	U_MSS_SBIC_ADDR	Address of SBIC in MSS address space

Secure Boot Image Certificate (SBIC)

Authentication Framework



Value	Description
IMAGE_ADDR	Address of UBL in SOC Memory map
IMAGE_LEN	Size of UBL in Bytes
$BOOT\_VEC_0$	Boot Vector of UBL for the Monitor Core
$BOOT\_VEC_{1-4}$	Boot Vectors for User Cores
H	UBL Image Hash (SHA512-256)
CODESIG	SBIC Digital Signature (ECDSA)

Secure Boot Image Certificate (SBIC) (168 bytes)

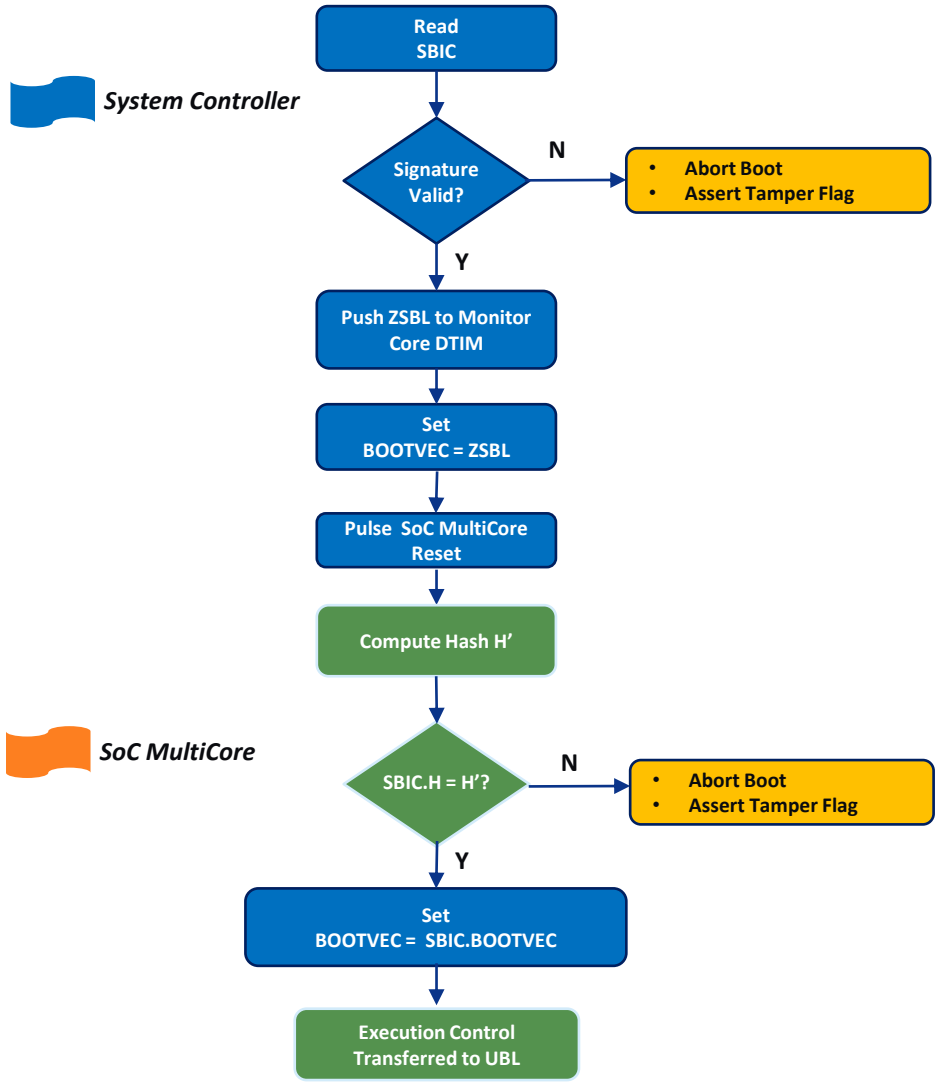
- **pNVM: Private Non-Volatile Memory**
  - UCSQ is a public key programmed onto the device during programming
    - **Corresponds to the UCSK private key used to sign the SBIC**
  - Stores address of SBIC in MSS_BOOT_CFG

$$CODESIG = ECDSA_{SIGN}(UCSK, IMAGE\_ADDR \mid IMAGE\_LEN \mid BOOTVEC_{0-4} \mid H)$$

$$ECDSA_{VERIFY}(UCSQ, IMAGE\_ADDR \mid IMAGE\_LEN \mid BOOTVEC_{0-4} \mid H, CODESIG)$$

Elliptic Curve Digital Signature Algorithm (ECDSA)

Authentication Flow



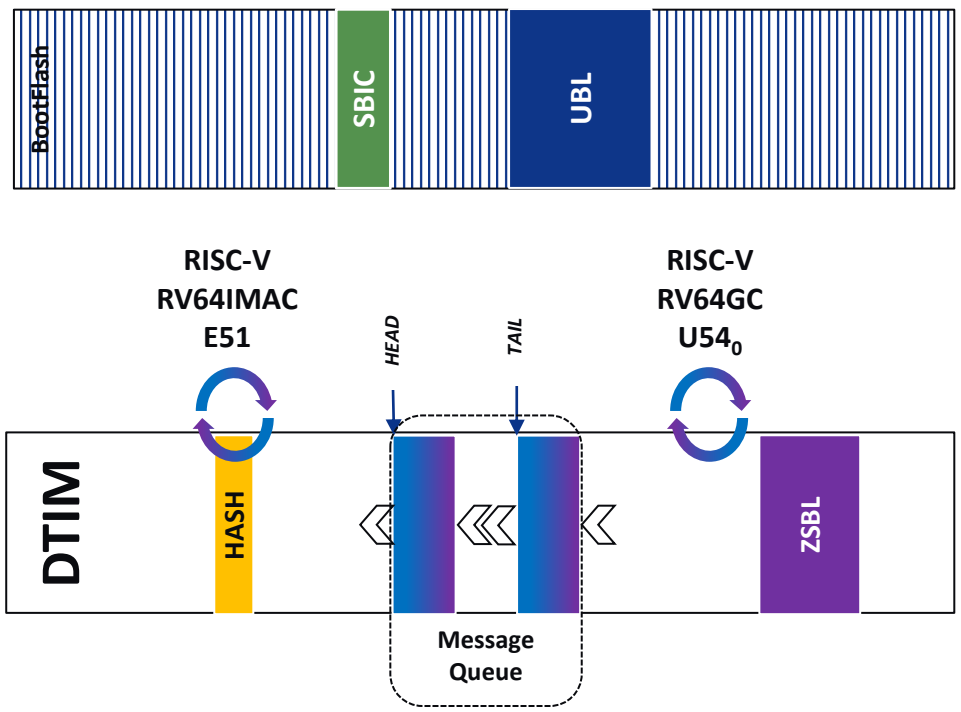
System controller pulls SBIC and authenticates using ECDSA on side channel resistant crypto processor

System controller pushes ZSBL from ROM/pNVM to monitor core DTIM

ZSBL computes UBL Hash H' and compares it to SBIC.H

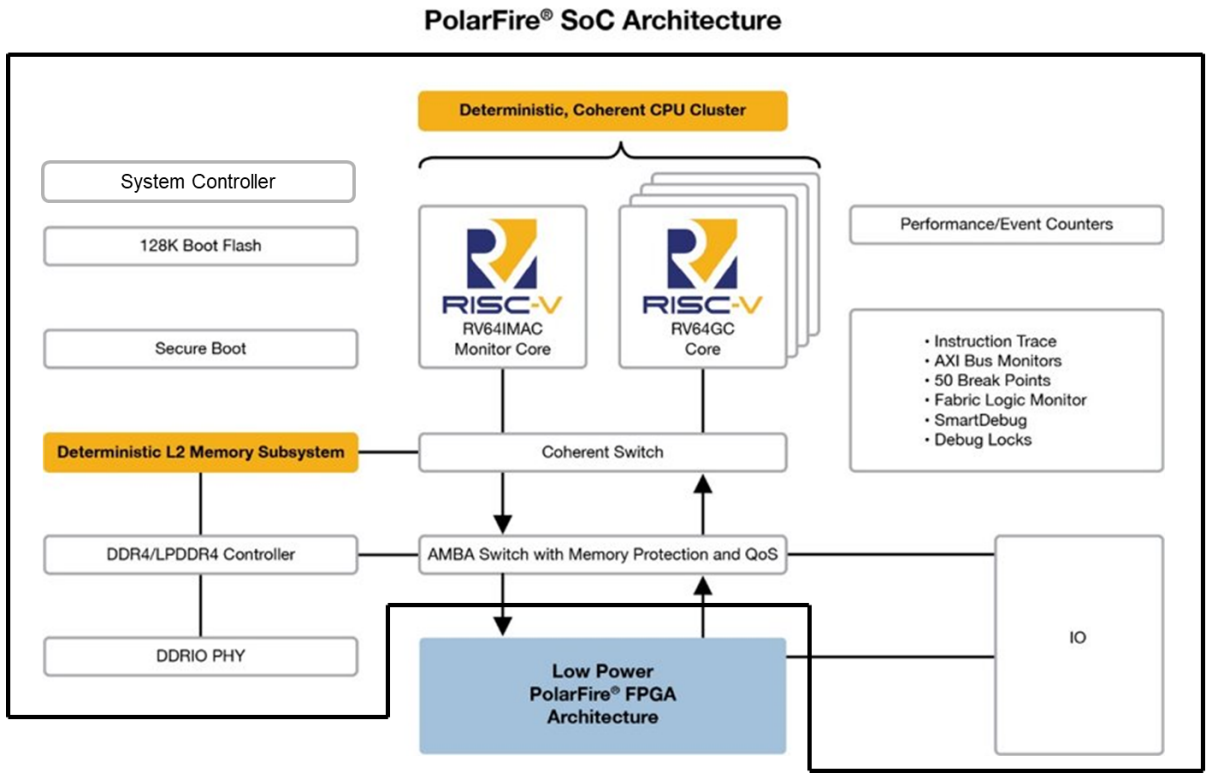
Transfer execution control to UBL

UBL Secure Hash (SHA-512/256)



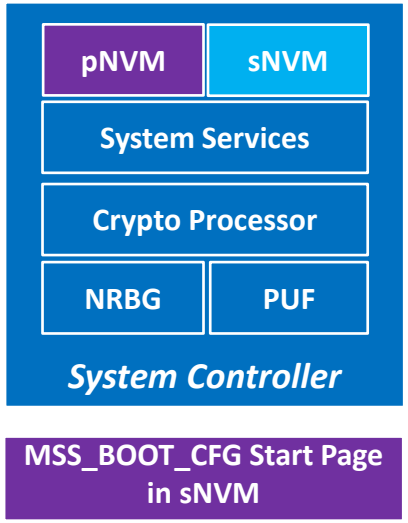
- **SHA-512/256 is cryptographically similar to SHA-256 but uses 64-bit arithmetic**
  - Has more rounds than SHA256 but processes twice as much data
- **Algorithm consists of 2 main operations:**
  - Message scheduling
  - Hash computation
- **Message scheduling runs on U54$_0$**
  - Blocks of 1024 bits are processed and pushed onto message queue
- **Hash computation runs on E51**
  - E51 reads each block from message queue and adds to hash accumulator
- **Message scheduling has complexity similar to hash computation**
  - Achieves close to 2x speed up over single-threaded approach

Execution Times



- **PolarFire® SoC Time to First Instruction**
  - 10.2 ms
- **Secure Boot Execution Time:**
  - 8.4 ms per 128 KB @ 600 MHz
  - 18.6 ms to authenticate 128 KB image

User Boot Loader Framework



Value	Description
IMAGE_ADDR	Address of UBL in SOC Memory map
IMAGE_LEN	Size of UBL in Bytes
$BOOT\_VEC_0$	Boot Vector of UBL for the Monitor Core
$BOOT\_VEC_{1-4}$	Boot Vectors for User Cores
H	UBL Image Hash (SHA512-256)
IMAGE	User Boot Loader Image

User Boot Loader Image Format

- **sNVM: Private Non-Volatile Memory**
  - Store UBL in sNVM
  - Pages can be marked as “ROM”
    - Only updateable from bitstream
  - Hash computed while being transferred from sNVM to the E51 DTIM
  - Can optionally PUF protect the sNVM contents
  - Any errors raises the Boot Fail tamper flag

Security Built for Defense, Ready for IoT

- **PolarFire® SoC inherits best in class PolarFire FPGA Security**
  - DPA resistant bitstream programming
  - Anti-tamper
  - Cryptographical bound supply chain assurance
  - Physically unclonable function
  - True random number generator
  - Side channel resistant crypto coprocessor

- **PolarFire SoC adds:**
  - + Standard secure boot and user defined Secure Boot
  - + Spectre and Meltdown immunity
  - + Physical memory protection
  - + SECDED on all MSS memories

MICROCHIP

Thank You