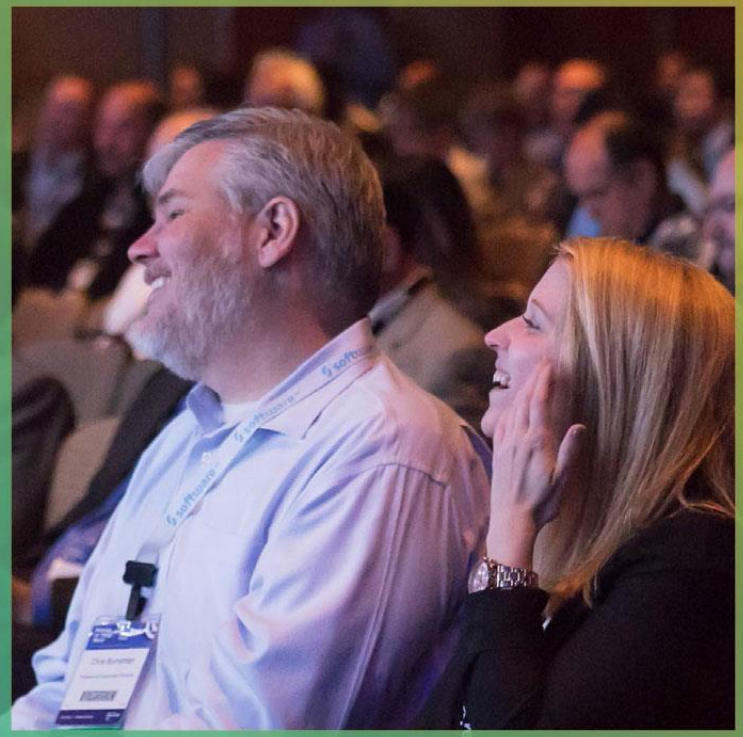
RISC-V Summit

December 3 - 6, 2018
Santa Clara Convention Center
CA, USA



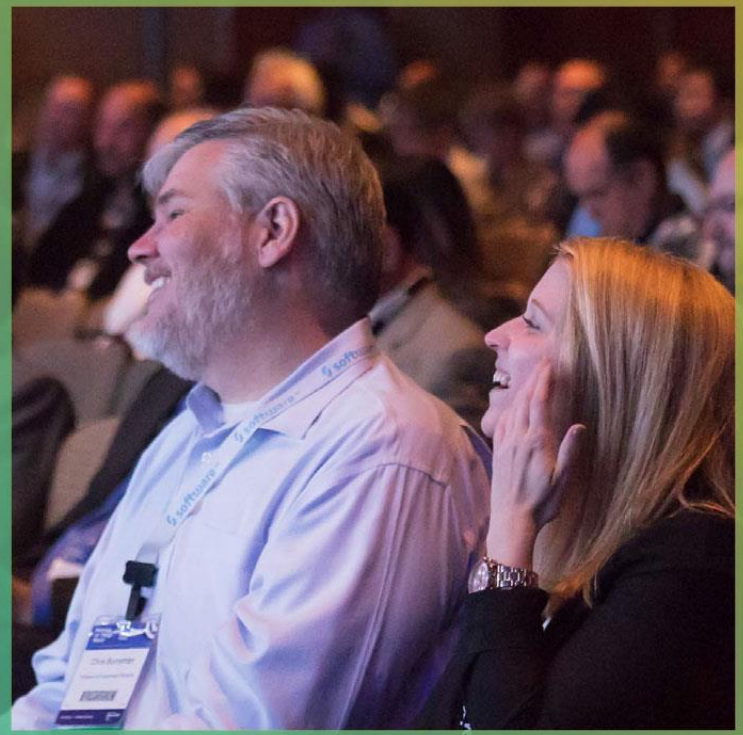
Revolutionizing the Computing Landscape and Beyond

https://tmt.knect365.com/risc-v-summit

@risc_v

RISC-V Summit

RISC-V MultiCore Secure Boot



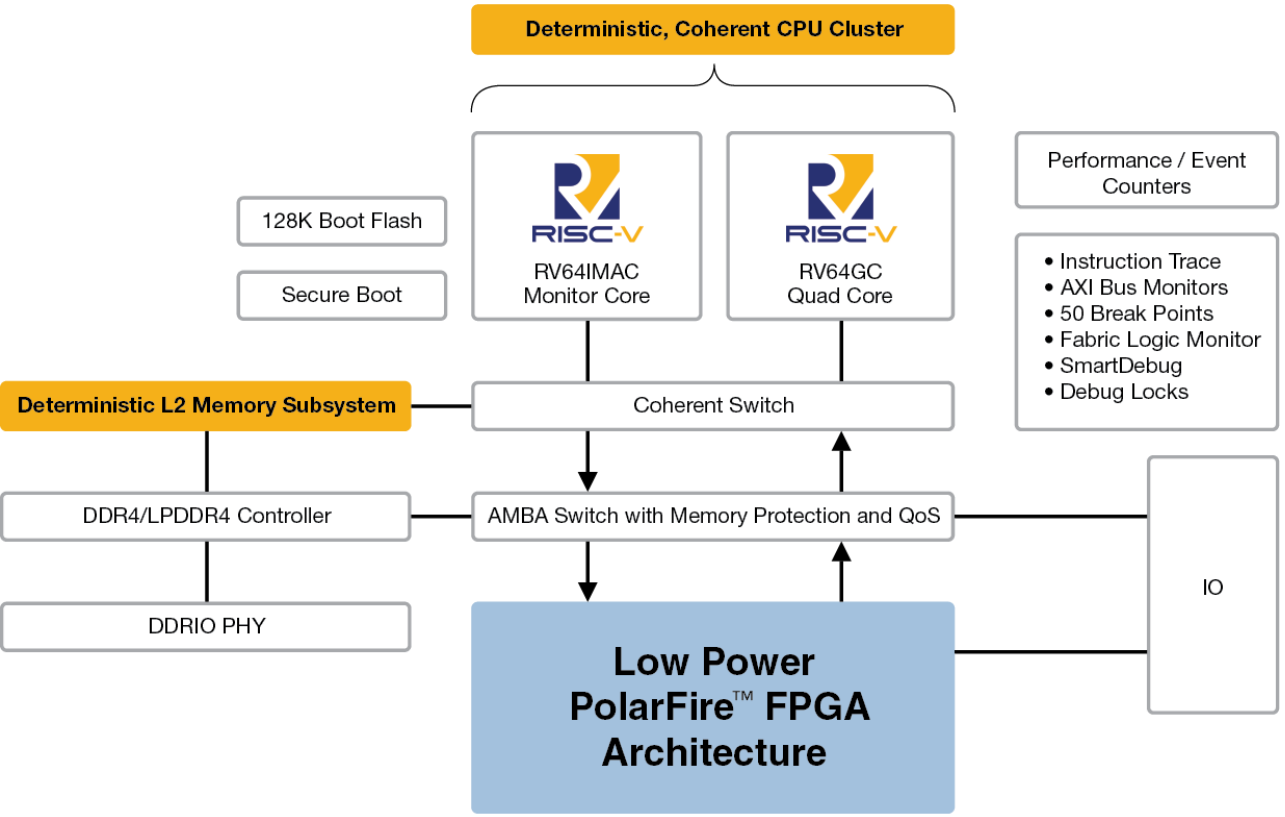
Ken Irving
Chief Engineer
Microsemi,
a Microchip company

Pierre Selwan
Chief Architect
Microsemi,
a Microchip company

https://tmt.knect365.com/risc-v-summit

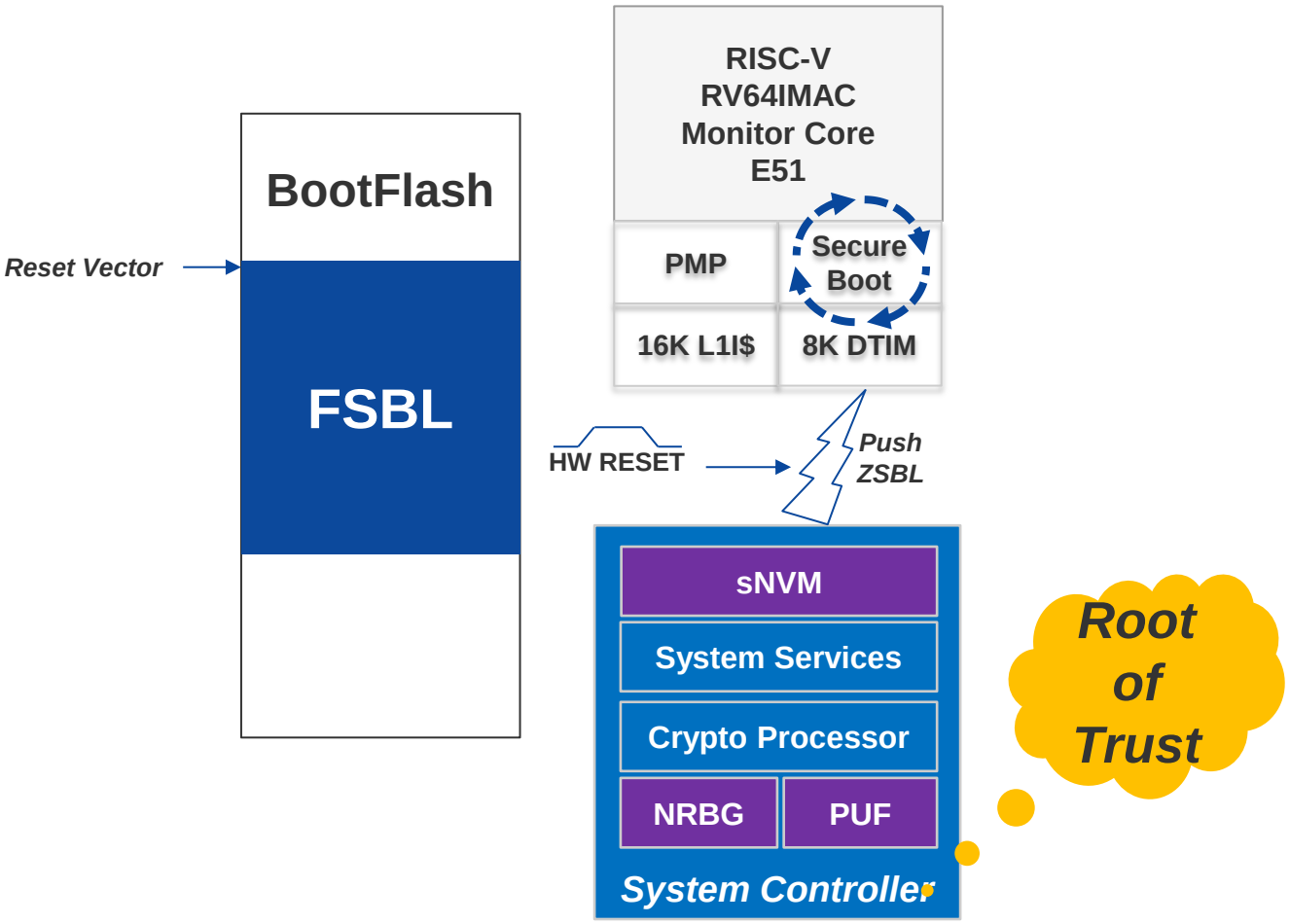
@risc_v

PolarFire SoC FPGA



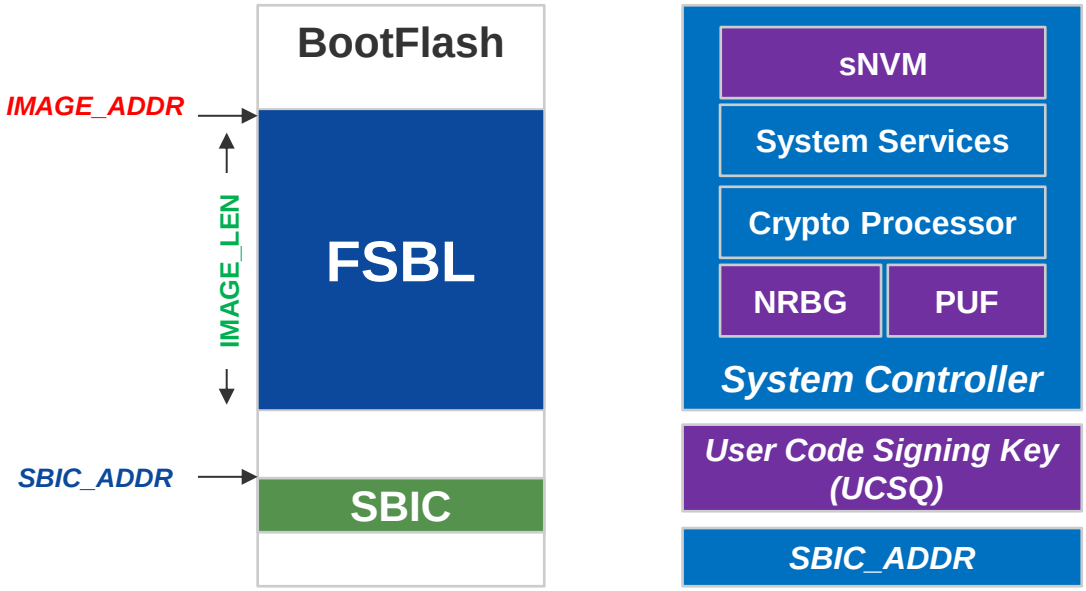
- PolarFire SoC FPGA is built around a multi-core RV64 capable of running full Linux and real-time OS, extensible through the FPGA fabric.
  - Based on SiFive U54-MC CoreComplex featuring E51/U54 processors
- Hybrid compute platform combines multi-core CPU, peripherals and custom HW accelerators in the FPGA fabric, which provides compelling value when balancing flexibility, functionality, throughput, power consumption and form factor.
- Security has emerged as the preeminent concern as SoC FPGAs are proliferated into a vast number of embedded systems applications where higher levels of reliability and tamper-resistance are fundamental requirements.
- Security concern is compounded when one considers “field programmability,” which is inherent to the very basic value proposition of such devices.

Secure Boot



- Guards against sophisticated methods of attack whereby a malicious external agent tampers with the boot image stored in bootflash (e.g Linux FSBL)
- Authenticates the image in bootflash before transferring execution control to the OS boot loader pointed to by reset vector
- FPGA system controller (root of trust) manages the authentication process and certifies boot image using crypto functionality built into the FPGA backbone
  - Push “zero state boot loader” (ZSBL) upon detecting HW reset.
  - Release monitor core from reset and executes authentication on FSBL image pointed to by reset vector.
  - If authentication is successful, transfer execution control back to FSBL, otherwise abort.

Authentication Framework



- ZSBL bootloader authenticates FSBL image in bootflash which contains:
  - Actual FSBL image
  - SBIC data structure generated during bootflash programming and stored @ SBIC_ADDR
- Authenticity of SBIC is verified by FPGA system controller using ECDSA*:*
  - UCSQ is a public key programmed on the device by the user
  - Corresponds to UCSK private key used to sign the SBIC during programming

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

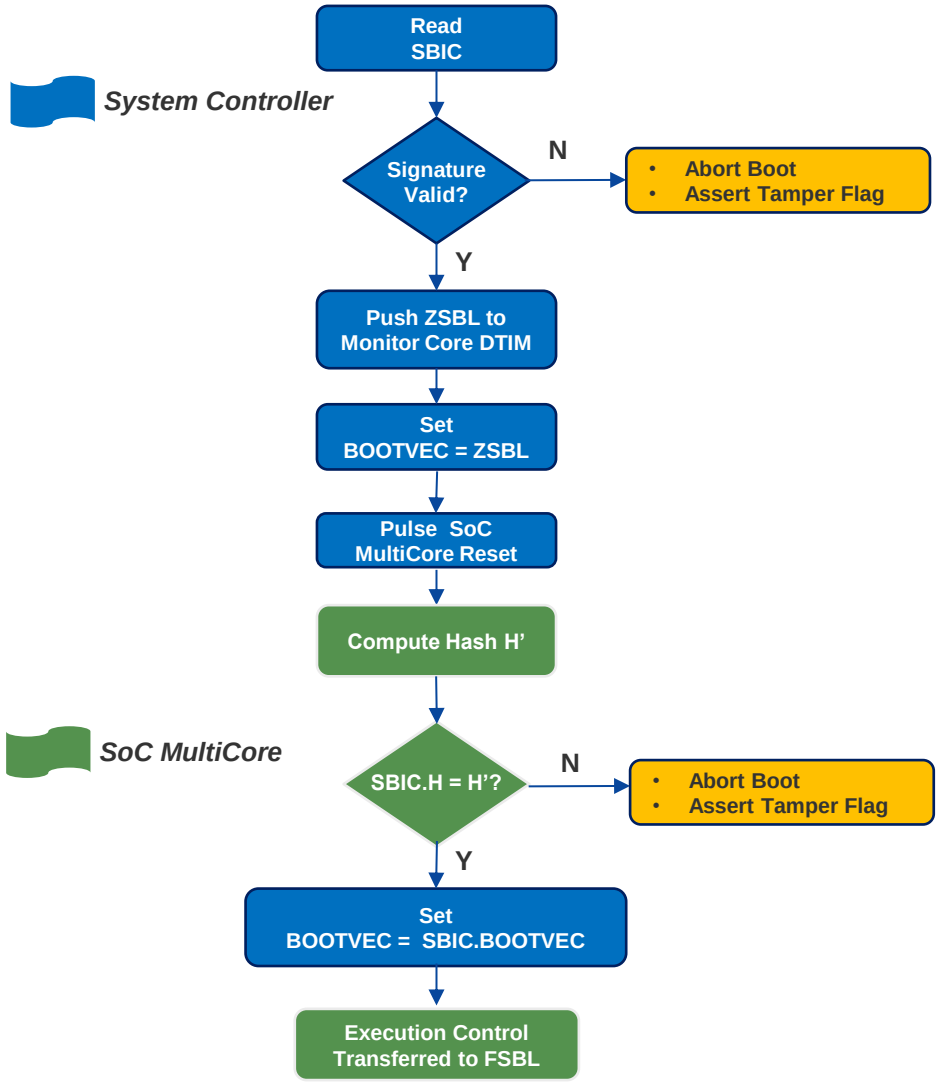
Secure Boot Image Certificate (SBIC)

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

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

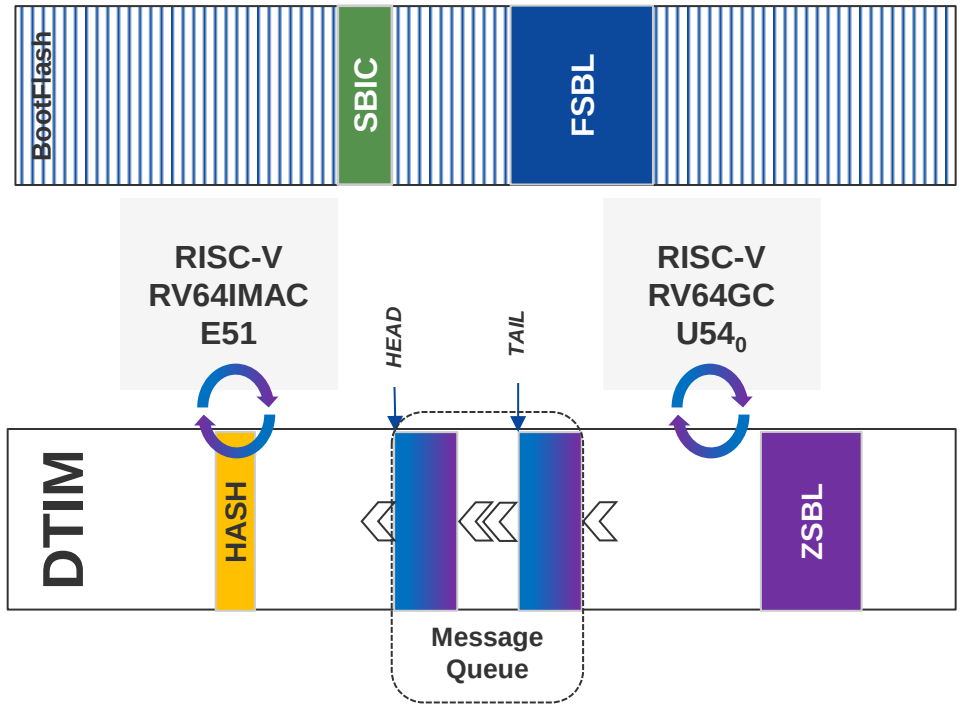
Elliptic Curve Digital Signature Algorithm (ECDSA)

Authentication Flow



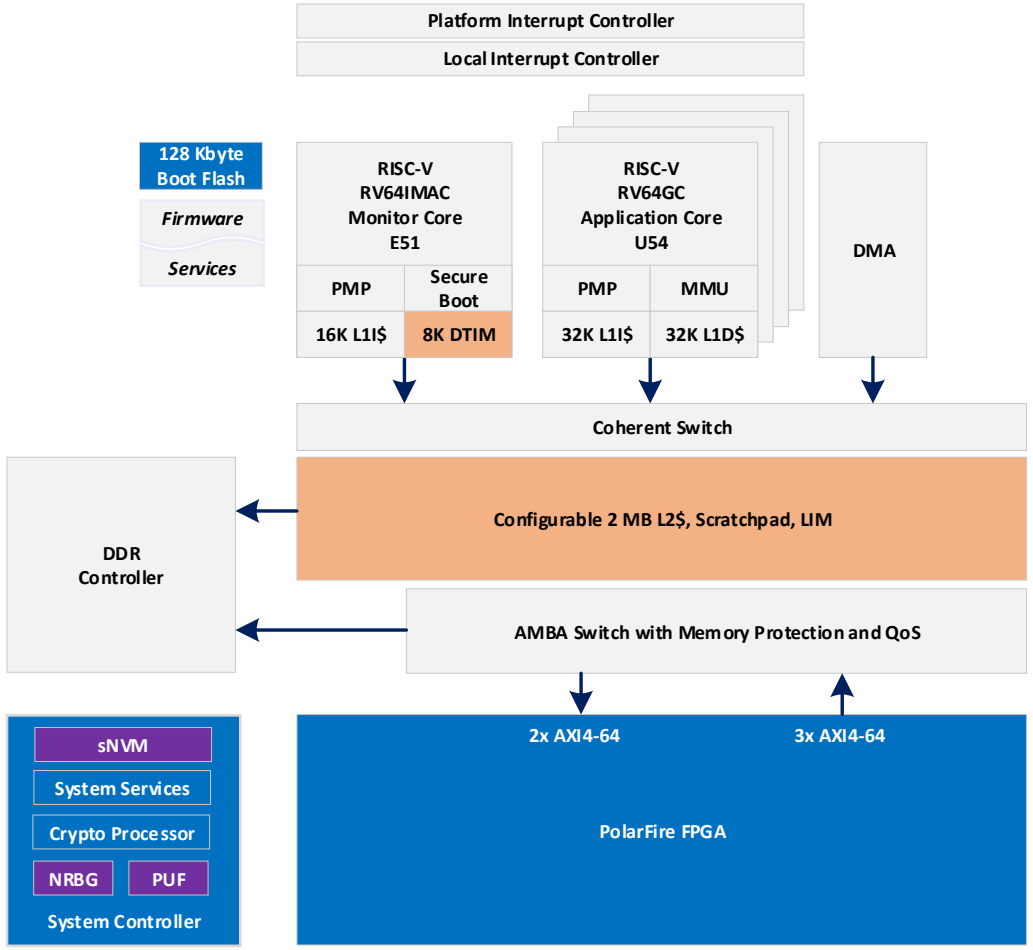
- System controller pulls SBIC and authenticates using ECDSA on crypto processor
- System controller pushes ZSBL from ROM/sNVM to monitor core DTIM
- ZSBL computes FSBL Hash H' and compares it to SBIC.H
- Transfer execution control to FSBL
- Invalid SBCI signature or invalid hash:
  - Abort boot and halt multi-core
  - Tamper flag raised to FPGA

Boot Image 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

Conclusion

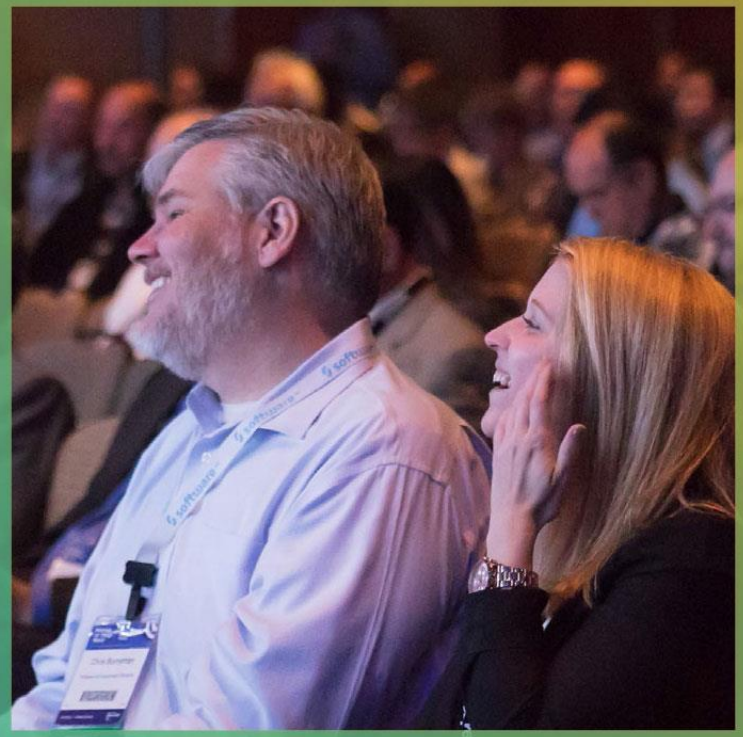


PolarFire SoC FPGA

- Secure Hash implementation described here is optimized for the proper balance of compute v/s IO
  - Bootflash read cycle time ~= 40ns
- Experimented with multiple variations of code/data locations in DTIM v/s L2 LIM
  - Code/data in DTIM yielded the highest throughput
- Throughput achieved is deemed more than adequate for target applications
  - 8.4ms per 128KB @600MHz
- Flexible and cost-effective
  - All RV64 code including atomic operations for queue pointer synchronization
  - No special accelerators required

RISC-V Summit

Thank you



https://tmt.knect365.com/risc-v-summit

@risc_v