Ada & PolarFire® SoC

A Software and Hardware Alloy for Safety & Security

RISC-V Workshop, Zurich

June 12, 2019

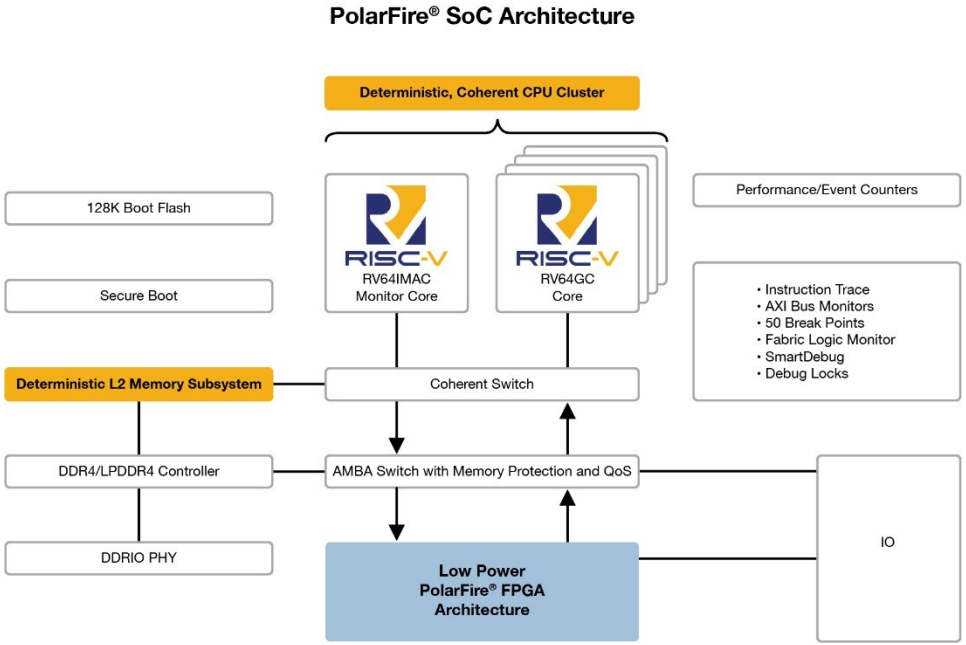
Pierre Selwan, System Architect, Microsemi, a Microchip Technology Company

Fabien Chouteau, Senior Embedded Software Engineer, AdaCore

Agenda

- **PolarFire® SoC**
  - High level architecture
  - Core complex
  - Deterministic execution
- **Ada and SPARK**
  - Programming language choice
  - Ada features for safe and secure software development
  - SPARK, the safety critical subset of Ada
  - Real-time support from Ravenscar microkernel

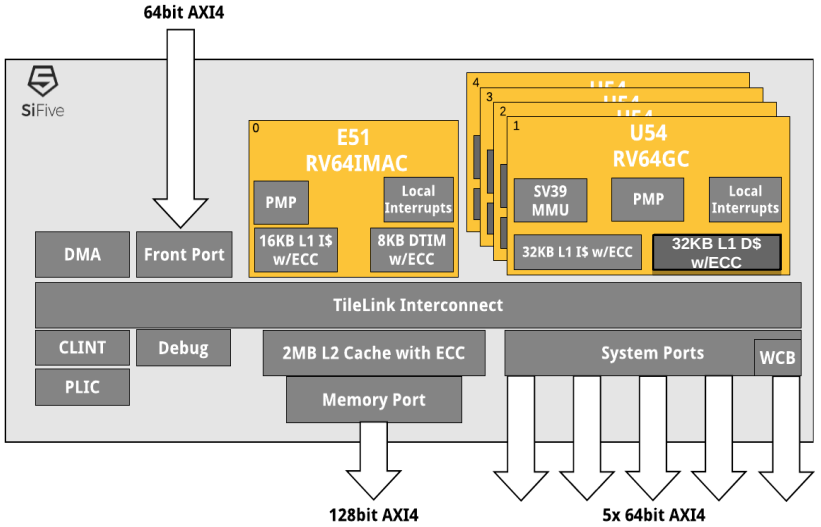
PolarFire SoC: A Reliable Programmable SoC FPGA



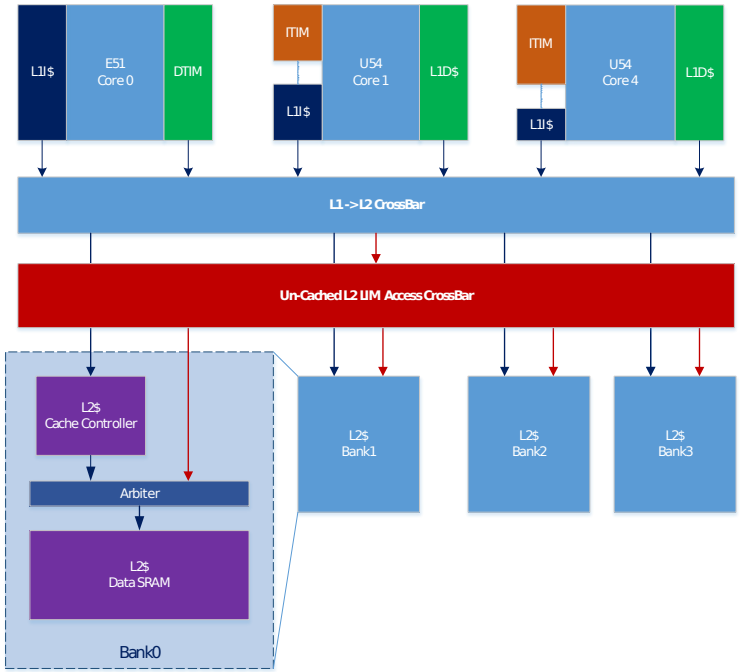
- Linux® and hard real time
- SECDED on all memories
- NSEU immune FPGA configuration
- Spectre/meltdown immune
- Secure boots
  - Default and user defined

PolarFire SoC CoreComplex
High Performance, Flexible

- **4x U54 RV64GC application cores**
  - Linux capable
  - 5-stage in-order
- **1x E51 RV64IMAC monitor core**
- **Fully coherent**
- **Real-time capability**
  - Flexible L1 and L2 memory system
  - Physical memory protection on each core
  - Option to disable branch prediction



Deterministic Execution With L1-ITIM and L2-LIM



- **Reconfigurable L1I$ to ITIM (Instruction Tightly Integrated Memory)**
  - Emulates tightly coupled memory
  - ITIM size: 28KB max out of 32KB L1I$
- **Reconfigurable L2$ to LIM (Loosely Integrated Memory)**
  - Emulates on chip memory
  - Out of reset, 15 out of 16 ways L2$ configured as L2-LIM (2MB – 128KB) max size
  - Once enabled, ways can not be turned back into L2-LIM
- **L2 LIM access**
  - Instruction access is cached in L1I$
  - Data access to L2-LIM is not cached in L1D$
  - Fixed latency with maximum jitter of 2 cycles (core clock)

Language Choice

A few questions to select a language:

… let’s consider a safety-critical embedded application …

What’s the contribution to the software development cycle?

Does it support the paradigms I need?

Does it generate efficient and deterministic code?

Can the language help write more reliable code?

Can the language help write more maintainable code?

Ada

noun

\ ˈā-də \

: The programming language solution for safety critical embedded applications

SPARK

noun

\ ˈspärk \

: The Ada subset fit for formal methods static analysis techniques

C - A Specification

```
typedef float temperature;

void update (temperature *t);
```

Ada - The Same Specification

The targets support at least 17 digits of precision (64 bits floating point).

The type accepts values between -213.15 and 1,000. These values can be represented in memory.

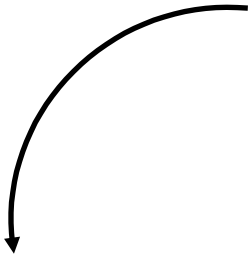
```
type Temperature is digits 17 range -213.15 .. 1_000.0;

procedure Update (T : in out Temperature);
```

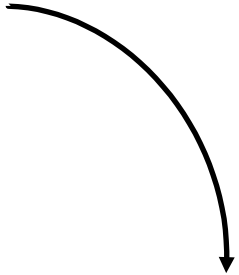
T is used by Update and is initialized at call time.

T is possibly modified by Update and updated after the call.

Ada - Avoiding Compiler Guesses

```
   A : Integer;
   B : Float;
   C : Integer;
begin
   C := A / B;
```





```
C := A / Integer (B);
```

```
C := Integer (Float (A) / B);
```

Ada – Why is it Useful?

The intent of the code is clearer for implementers and users.

The compiler can check local consistency at compile time.

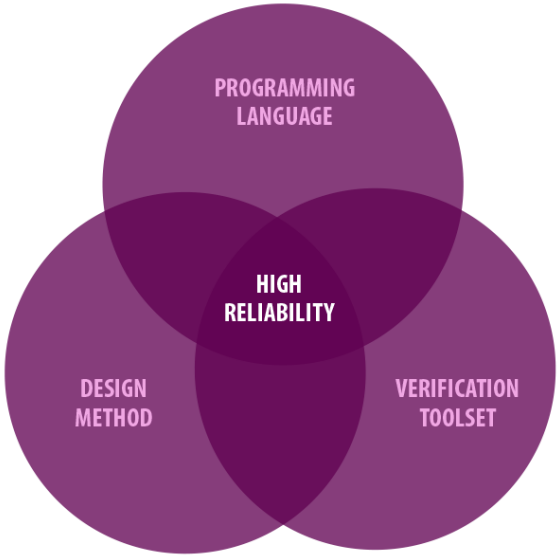
The test can check consistency at execution time.

Static analysis tools can demonstrate local or global consistency.

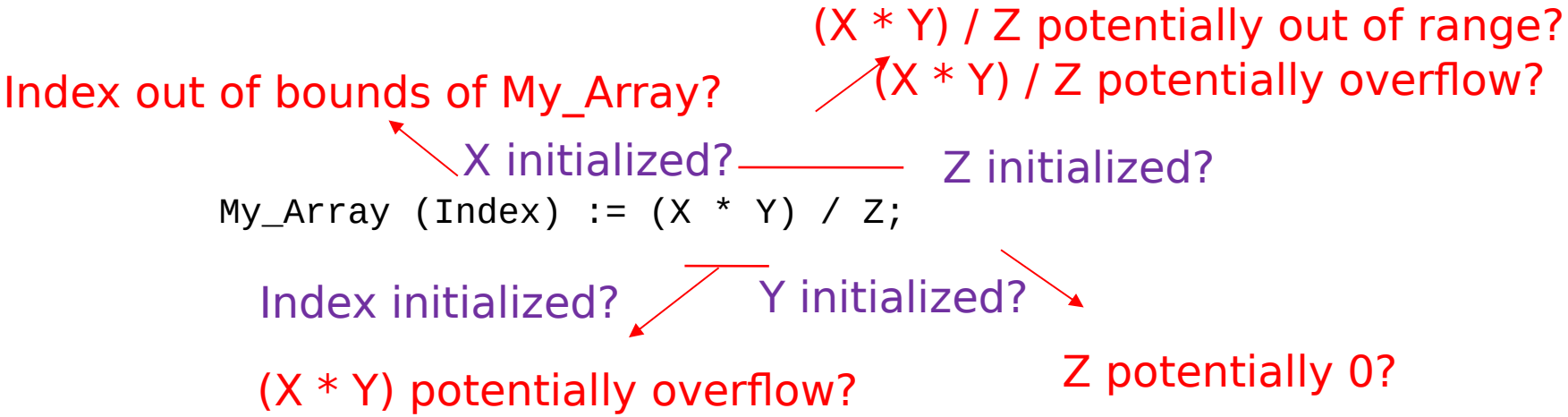
SPARK – The Ada Subset

SPARK is a software development technology specifically designed for engineering high-reliability applications

- A formally-defined programming language supporting static analysis
- Suite of tools to perform analysis
- Based on statically provable contracts and testing

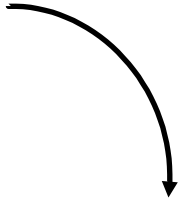


SPARK – Absence of Runtime Errors



SPARK – Replacing Defensive Code

```
procedure Next (A : in out Array; Iterator : in out Integer; Stop : Value);

procedure Next (A : in out Array; Iterator : in out Integer; Stop : Value) is
begin
   if Iterator in A'Range then
      loop
         Iterator := Iterator + 1;
         exit when Iterator not in A'Range or else A (Iterator) = Val;
      end loop;
   end if;
end Next;
```

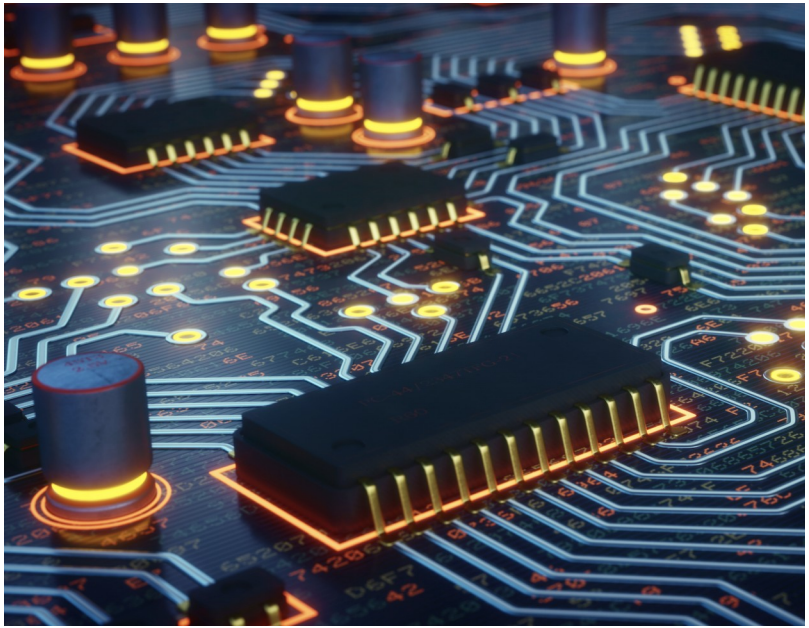


```
procedure Next (A : in out Array; Iterator : in out Integer; Stop : Value)
  with Pre => Iterator in A'Range;

procedure Next (A : in out Array; Iterator : in out Integer; Stop : Value)
is
begin
   loop
      Iterator := Iterator + 1;
      exit when Iterator not in A'Range or else A (Iterator) = Val;
   end loop;
end Next;
```

SPARK – Formal Verification

```
function Find (A : My_Array; V : Value; Start : Integer) return
Integer

   with Pre => Start in A'Range,

        Post => (if (for all E in Start .. A'Last => A (E) /= V)
                 then Find'Result = -1
                 else A(Find'Result) = V);
```

Ravenscar – Real-Time Multitasking



- **The Ravenscar profile**
  - Removes features that aren't fit for real-time operation
- **Lightweight runtime with Ravenscar support**
- **Features:**
  - Real-time tasking support
  - Priority-based preemptive scheduling
  - Protected objects
  - Multicore support
  - Standard library support

Reliable Hardware Needs Reliable Software

PolarFire SoC

- Determinism built-in
- Exceptional reliability
- Defense grade security

Ada and SPARK

- Formally verifiable software language
- Real-time, lightweight multitasking
- Proven technology in the safety critical realm of space and aviation

See us at the RISC-V Summit
December 9 – 12, 2019, in San Jose, California

AdaCore

Thank You