

The Era of SoC FPGAs

Nizar Abdallah, Ph.D.

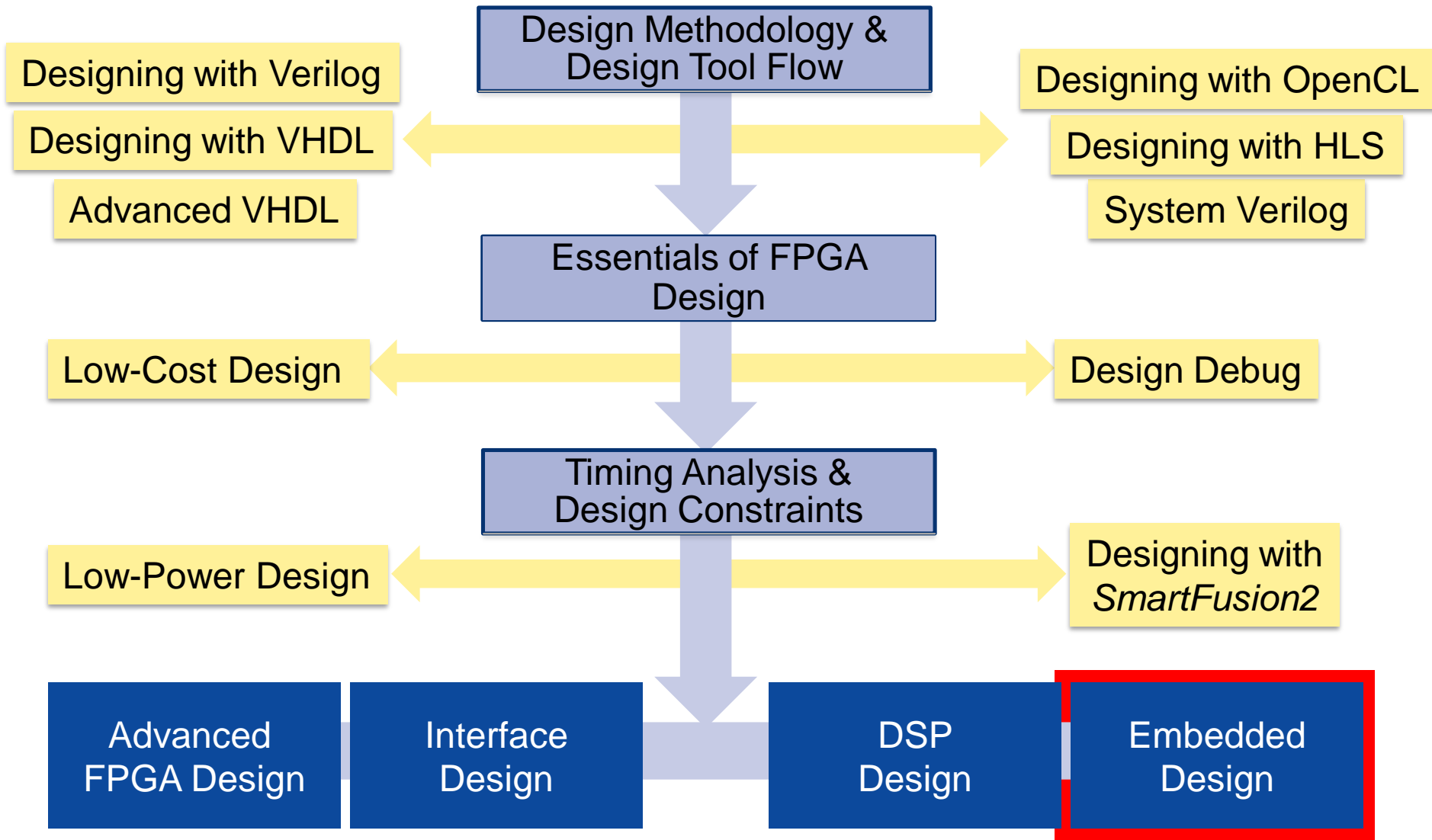
Workshop on FPGA Design for Scientific Instrumentation and Computing
International Centre for Theoretical Physics

November 2017

Outline

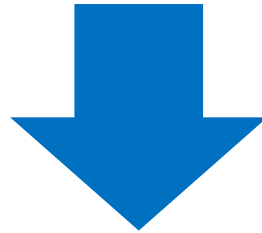
- Introduction
- SoC FPGA Architectures: an Overview
- The Processor Part in SoC FPGAs
- Design Flow in SoC FPGAs

Modern FPGA Design Curriculum



Today...

FPGAs & Processors
are meeting



the era of
Programmable SoC

What Happens in an Internet Minute?

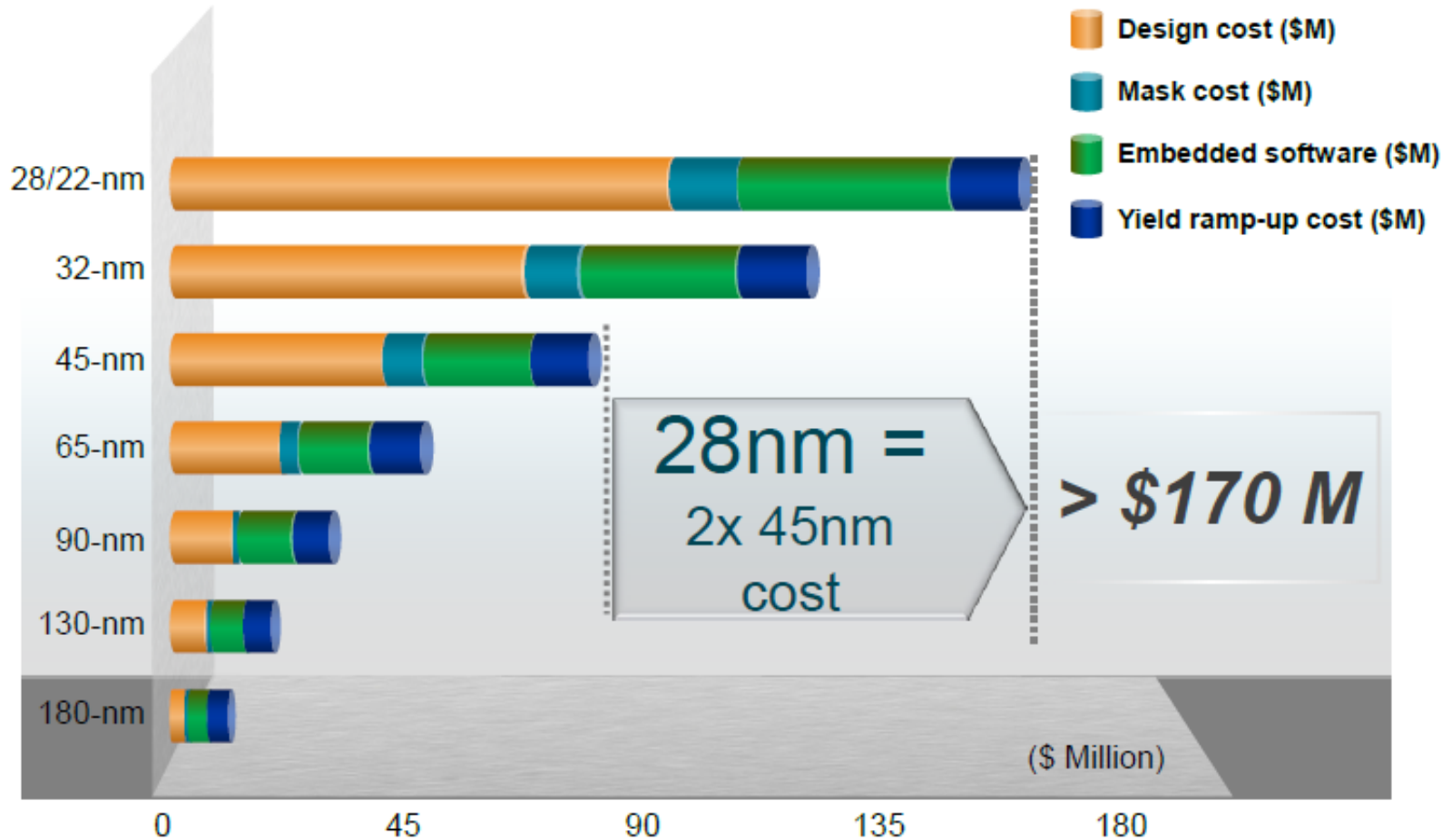


And Future Growth is Staggering



Design Cost

Estimated Chip Design Cost, by Process Node, Worldwide, 2011



More Intelligence in Every System

SMART Data Center Revolution

New Opportunities to Control Costs and Increase Strategic Advantage...

Smart wireless networks to the rescue

Carriers are turning toward more intelligent network management...

Smart Factories

For factory management in the future, it will become essential to strive to implement smart capabilities...

MACHINES THAT UNDERSTAND

embedded
VISION
ALLIANCE

The Next Big, Digital Economy; 'Smart Energy'

The energy market is undergoing a major transformation...

From Dumb Pipes to Smart Networks

Trend Data Center Infrastructure: Cloud Computing

Big Data

Increasing Volume, Velocity, and Variety



Low power

Reduce operation and cooling costs

Security

Both outside and inside

Programmable Imperative

SoC? FPGA? SoC FPGA?

- SoC
 - System on Chip
 - CPU Core + Peripherals
 - Programmable software
- FPGA
 - Field Programmable Gate Array
 - Plenty of I/O options
 - Extremely parallel architecture
 - Programmable hardware
- SoC FPGA
 - SoC & FPGA on a single chip
 - Connected through on-chip bus

Why SoC FPGA (one more time)?

- Reduce size => Reduce overall system cost
- Increase performance
- Lower power consumption
- Increase system reliability
- Need for special bus interface for a CPU
- Need for obscure amount of IOs
- Need for extra CPU power for your FPGA
- Need for extra FPGA speedup for your CPU functions

Available Today

| | Altera SoC FPGAs | Xilinx Zynq-7000 EPP | Microsemi SmartFusion2 |
|--------------------------|-------------------------|-----------------------------|-------------------------------|
| Processor | ARM Cortex-A9 | ARM Cortex-A9 | ARM Cortex-M3 |
| Processor Class | Application processor | Application processor | Microcontroller |
| Single or Dual Core | Single or Dual | Dual | Single |
| Processor Max. Frequency | 1.05 GHz | 1.0 GHz | 166 MHz |

- In addition to the processor, an SoC FPGA includes:
 - A rich set of peripherals,
 - On-chip memory,
 - An FPGA-style logic array, and
 - A lot of configurable I/Os

When does it make sense?

- Consider the following scenarios:
 1. The existing design uses an FPGA and a separate microprocessor?
 2. The current generation uses a proprietary ASIC that includes a microprocessor?
 3. A microprocessor being used today, but would benefit from a peripheral set more tailored to the application?
- What are the benefits in each case?

Architecture Matters

In Any Case...

Architecture Matters

Criteria for Choosing an SoC FPGA

- Design considerations & engineering trade-off decisions
- The selection criteria centers on the following areas:
 - Existing ecosystem (legacy IPs, Software...)
 - System performance
 - System reliability
 - System flexibility
 - System cost
 - Power consumption
 - Continuity (product roadmap)
 - Quality of the software solution (development tools)

System Performance

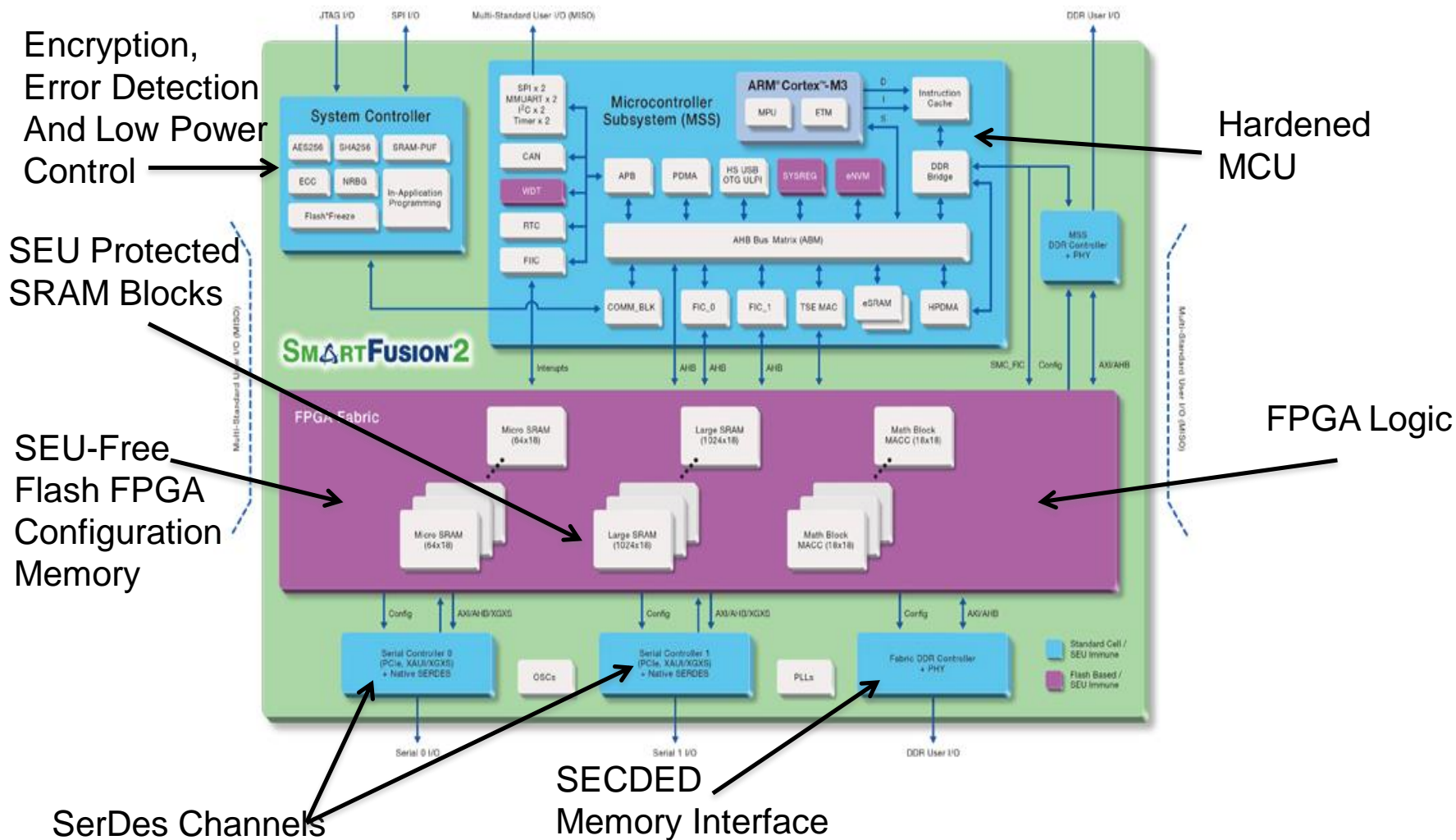
- Industrial Example: Motor Control
- The processing must be complete within a given window in time, every time

System Performance

- The processor performance
- The fabric performance
- The interconnect between fabric and processor
- Memory bandwidth

System Performance

The interconnect between fabric and processor

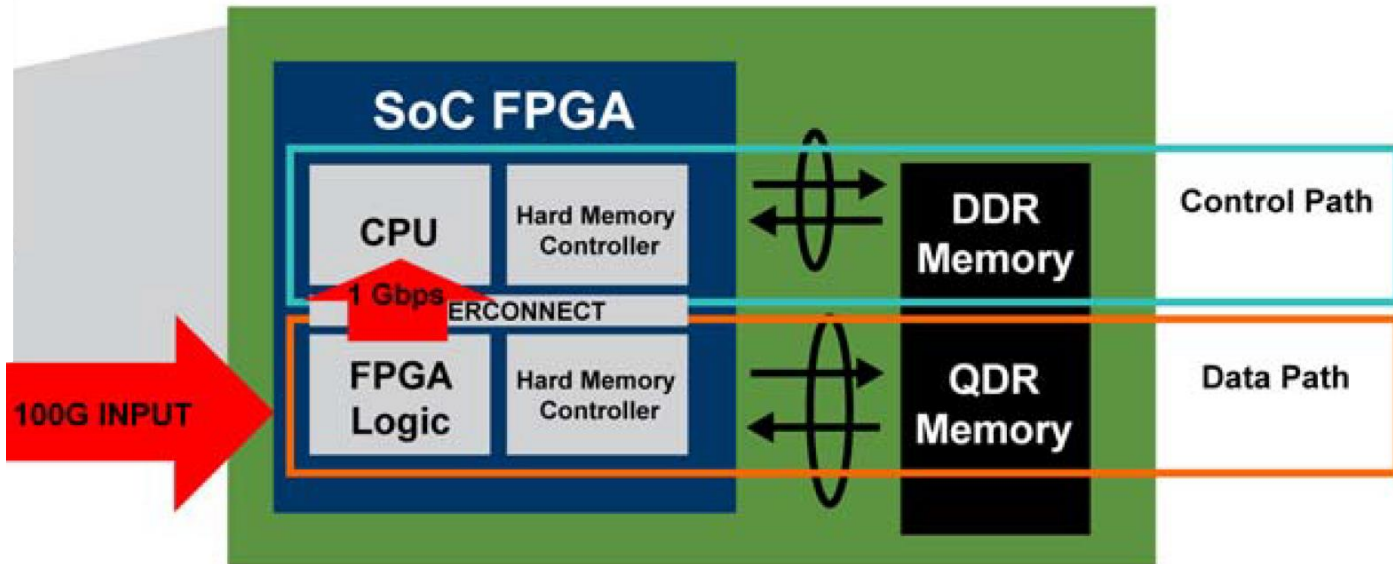


Data transfer between the memory, FPGA fabric, processor, and peripherals

System Performance

The interconnect between fabric and processor

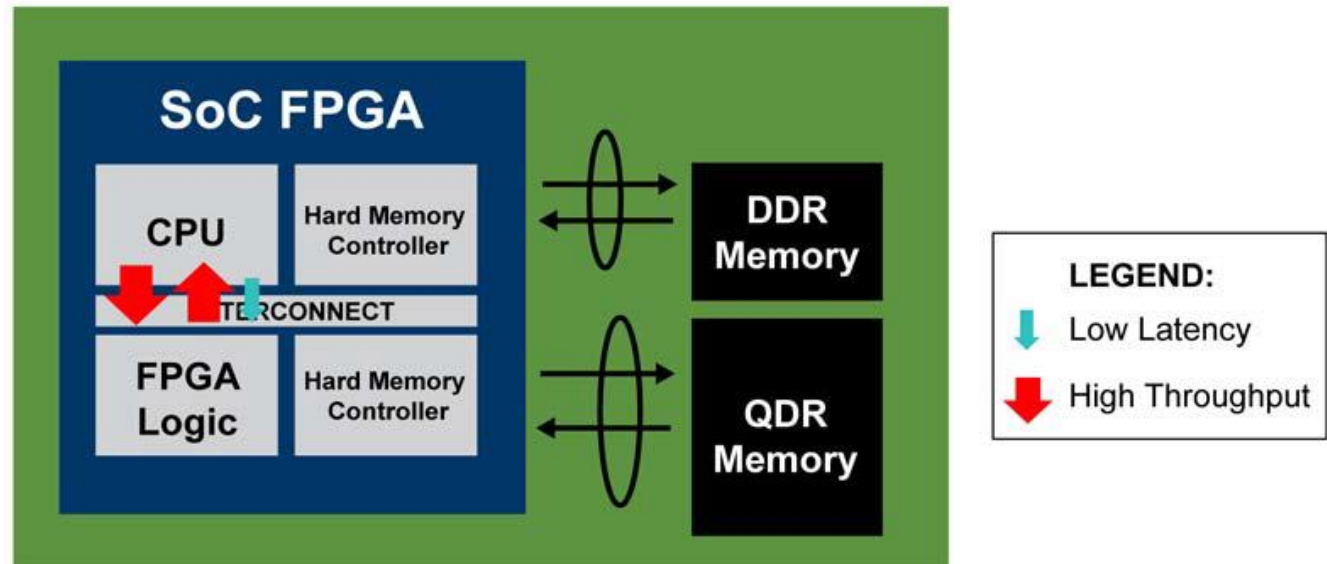
- Communication example



System Performance

The interconnect between fabric and processor

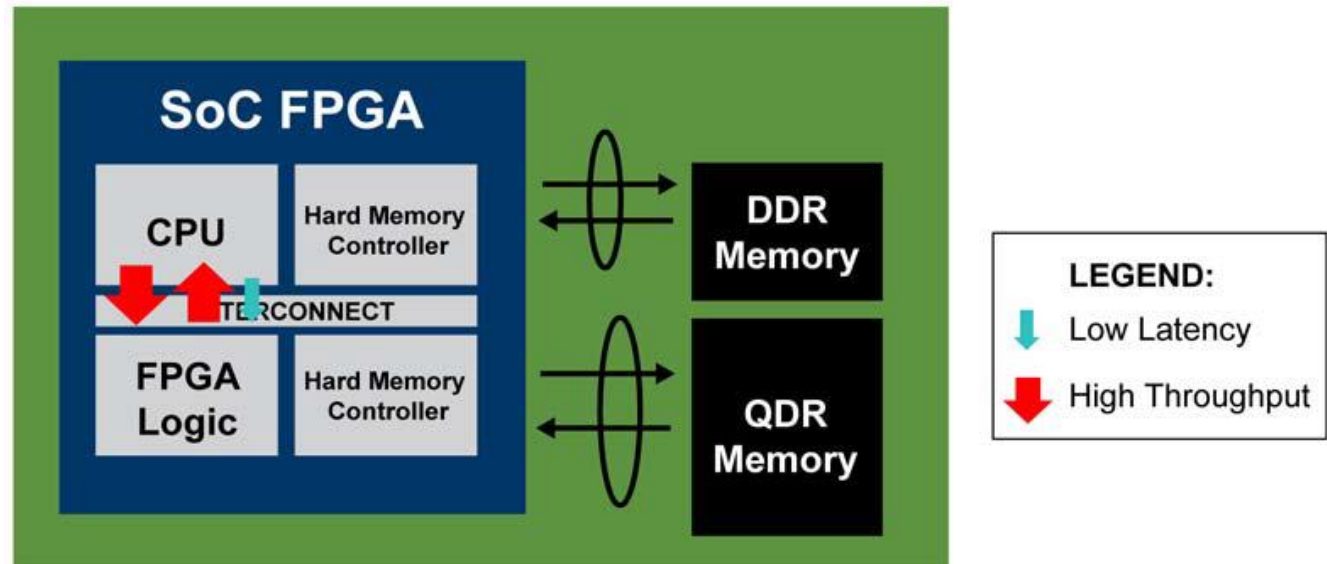
- Communication example: if needed, a low latency non-blocking bridge for control access in the FPGA



System Performance

The interconnect between fabric and processor

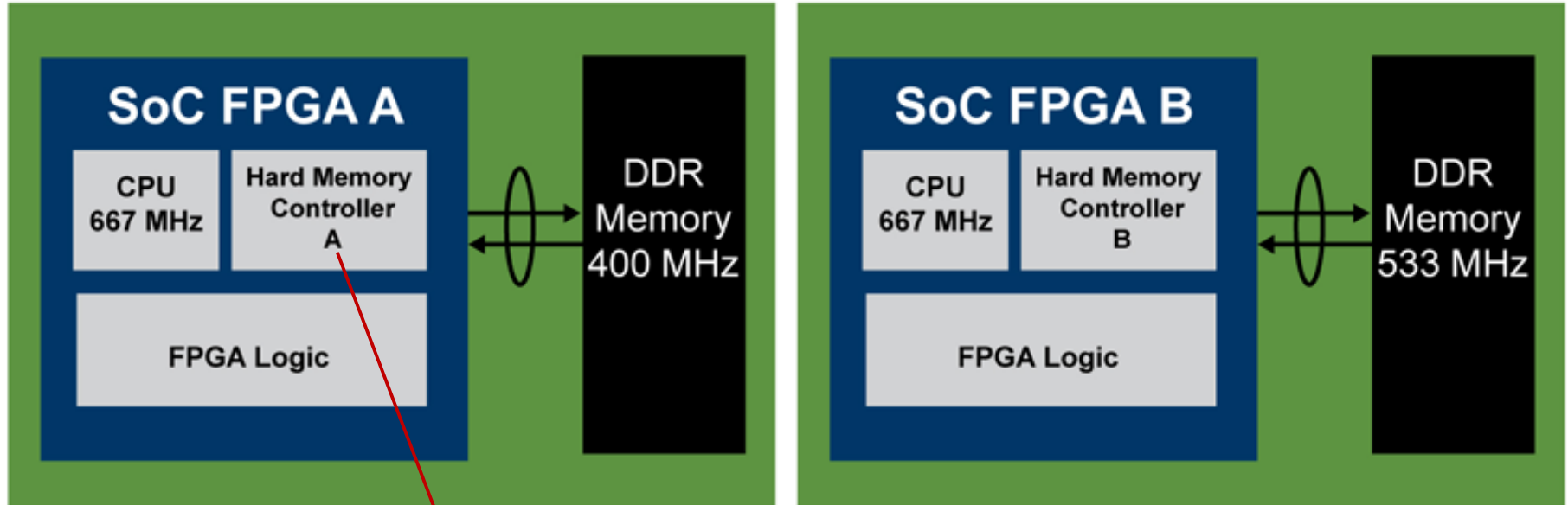
- Hardware acceleration example: When the acceleration results are needed by the processor
- In this case, in the other direction: Does the architecture include an Accelerator Coherency Port (ACP)?



System Performance

Memory bandwidth

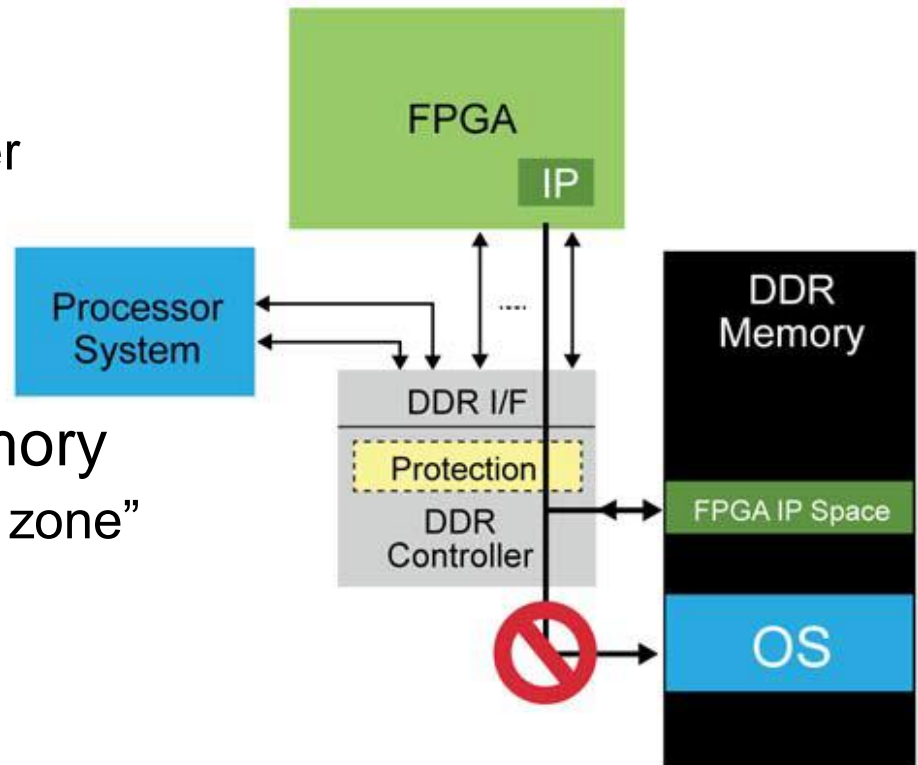
- Memory controllers as important as Memory speed
- Do you have separate hard memory controllers?
- How smart is the memory controller?



17% Faster using a smarter scheduling algorithm

System Reliability

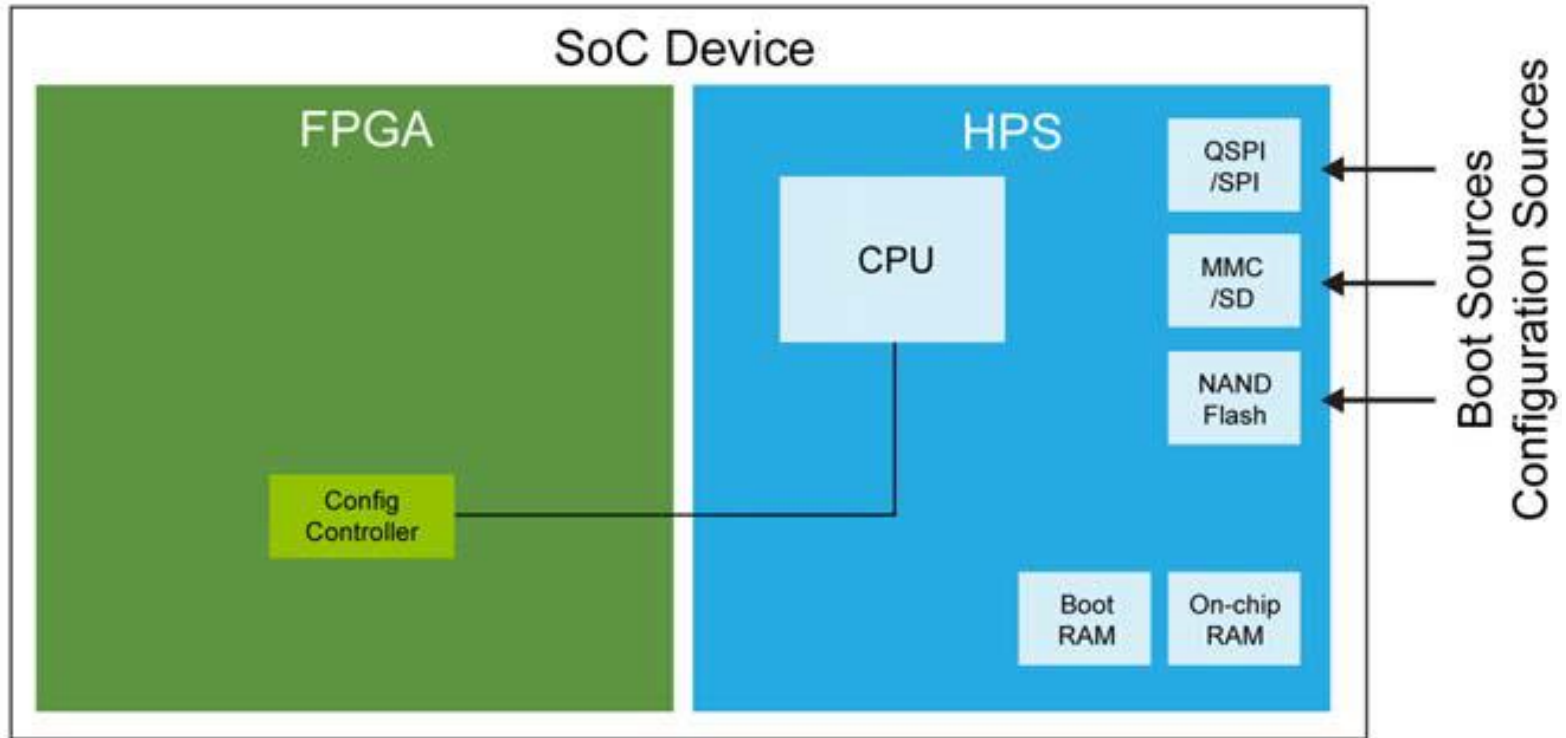
- Supporting ECC Memory for content protection
 - On-Chip RAM
 - External DDR Memory Controller
 - L1 Cache & L2 Cache
 - SPI Controller
 - DMA Controller
 - 10/100/1G Ethernet Controller
 - USB 2.0 OTG Controller
 - ...
- Protection for shared memory
 - Arm has the concept of “trust zone”



System Flexibility

- Extending the flexibility to the system level

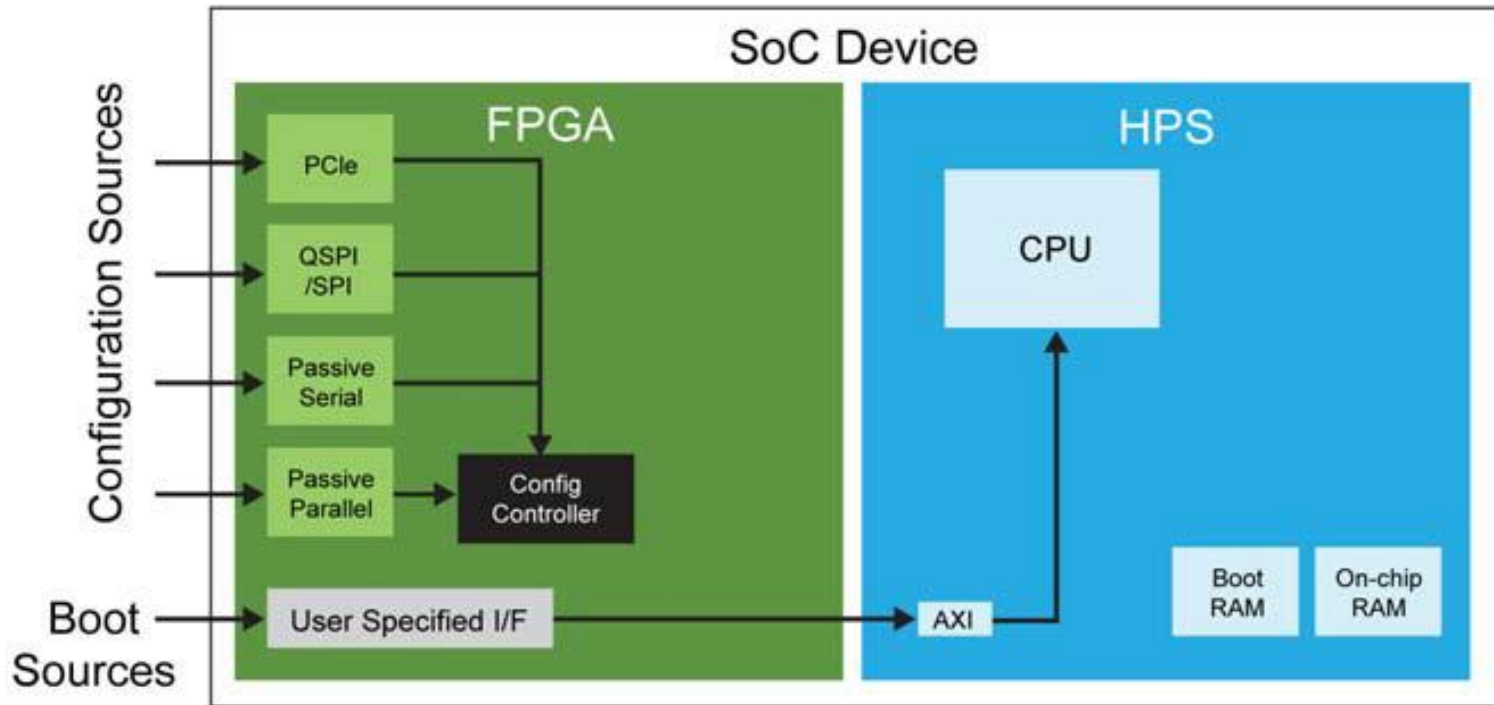
Processor boots first, then configures the FPGA



System Flexibility

- Extending the flexibility to the system level

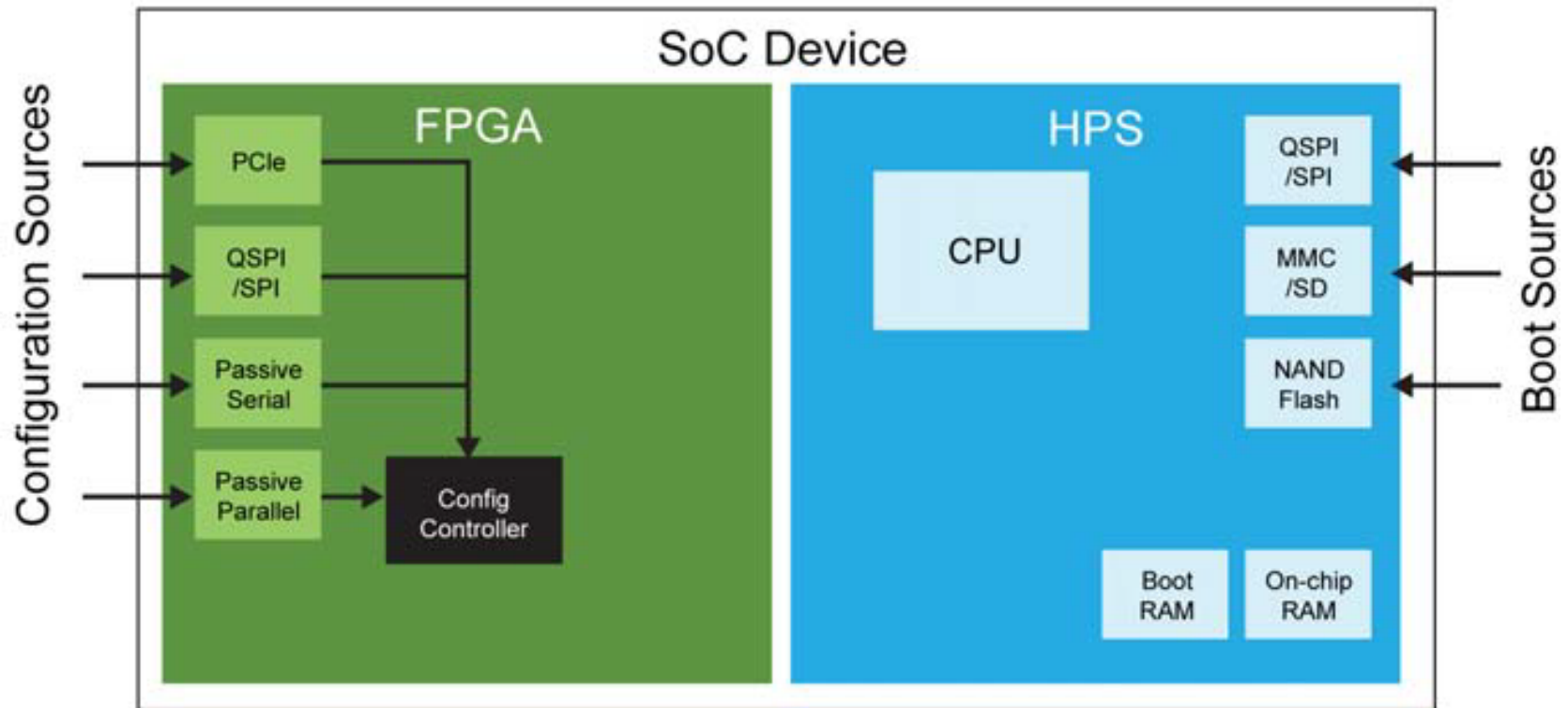
FPGA configures first, CPU boot through FPGA logic
(e.g. custom backplane I/F)



System Flexibility

- Extending the flexibility to the system level

Independent FPGA configuration and processor boot



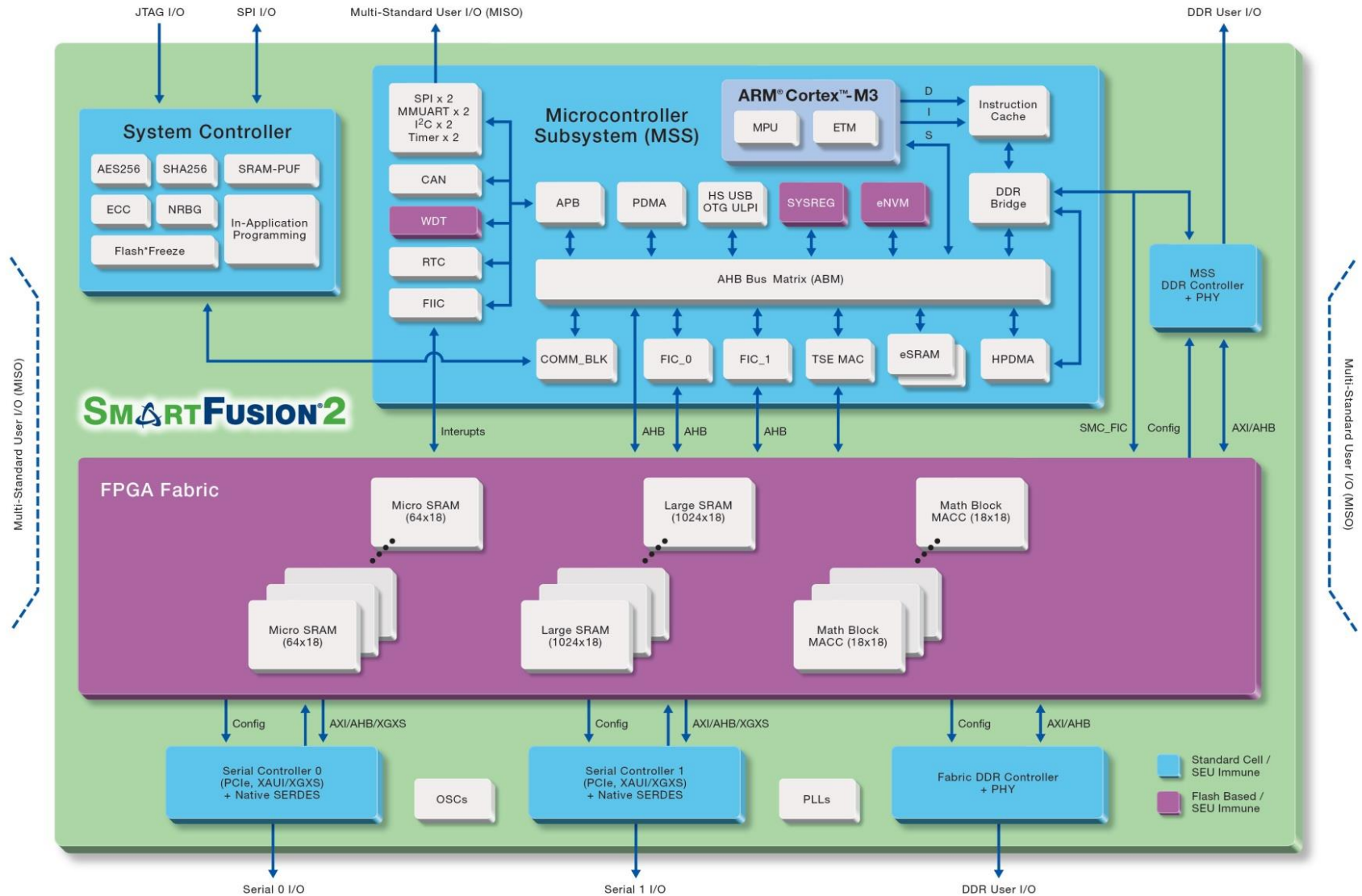
Criteria for Choosing an SoC FPGA

- Design considerations & engineering trade-off decisions
- The selection criteria centers on the following areas:
 - Existing ecosystem (legacy IPs, Software...)
 - System performance
 - System reliability
 - System flexibility
 - System cost
 - Power consumption
 - Continuity (product roadmap)
 - Quality of the software solution (development tools)

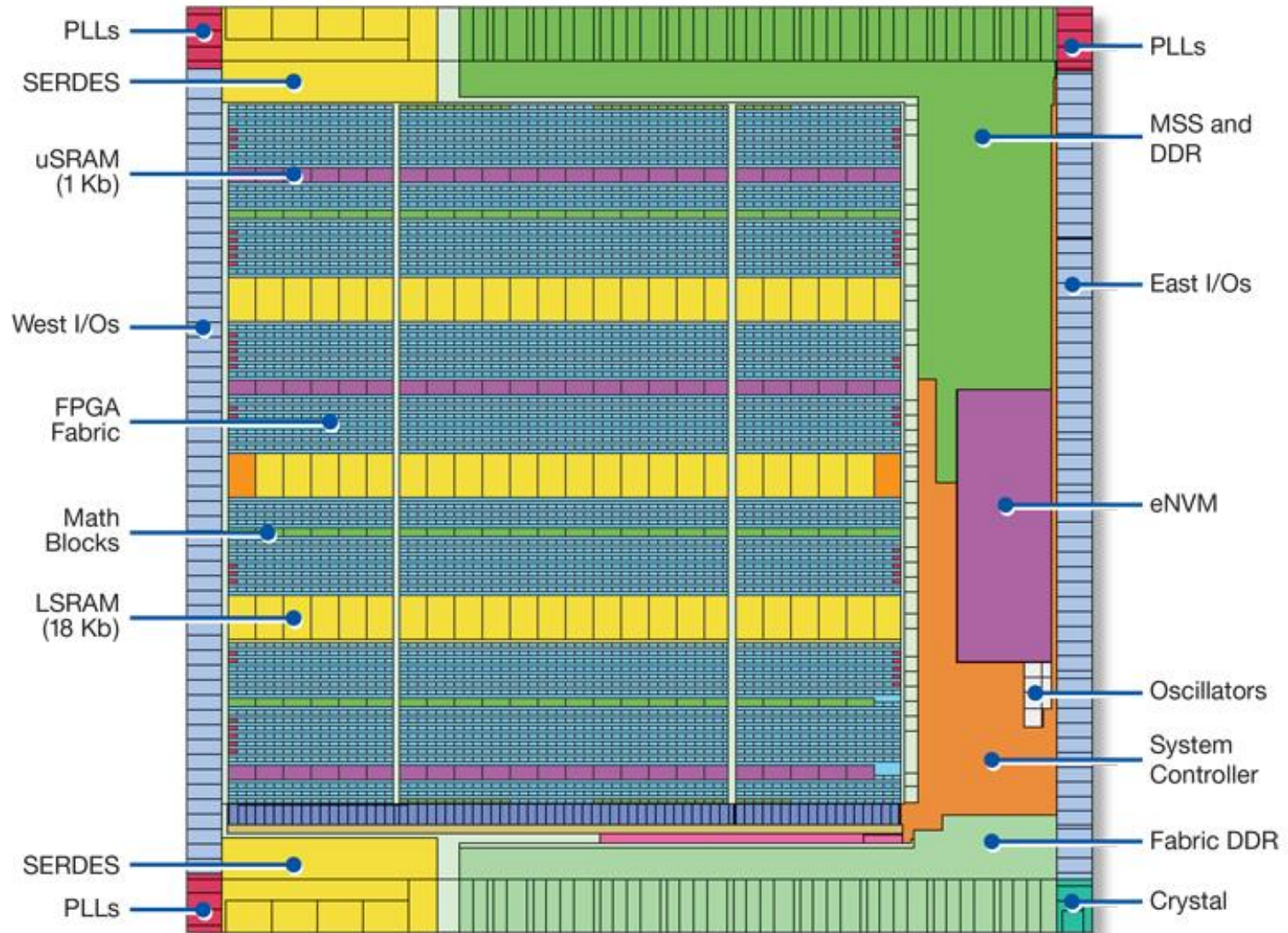
Embedded Processors

ARM Architecture Fundamentals

SmartFusion2 Architecture



SmartFusion2 Device Layout



Brief History

- ARM (Advanced Risc Machine) Microprocessor was based on the Berkeley/Stanford Risc concept
- Originally called Acorn Risc Machine because developed by Acorn Computer in 1985
- Financial troubles initially plagued the Acorn company but the ARM was rejuvenated by Apple, VLSI technology, and Nippon Investment and Finance

ARM Ltd

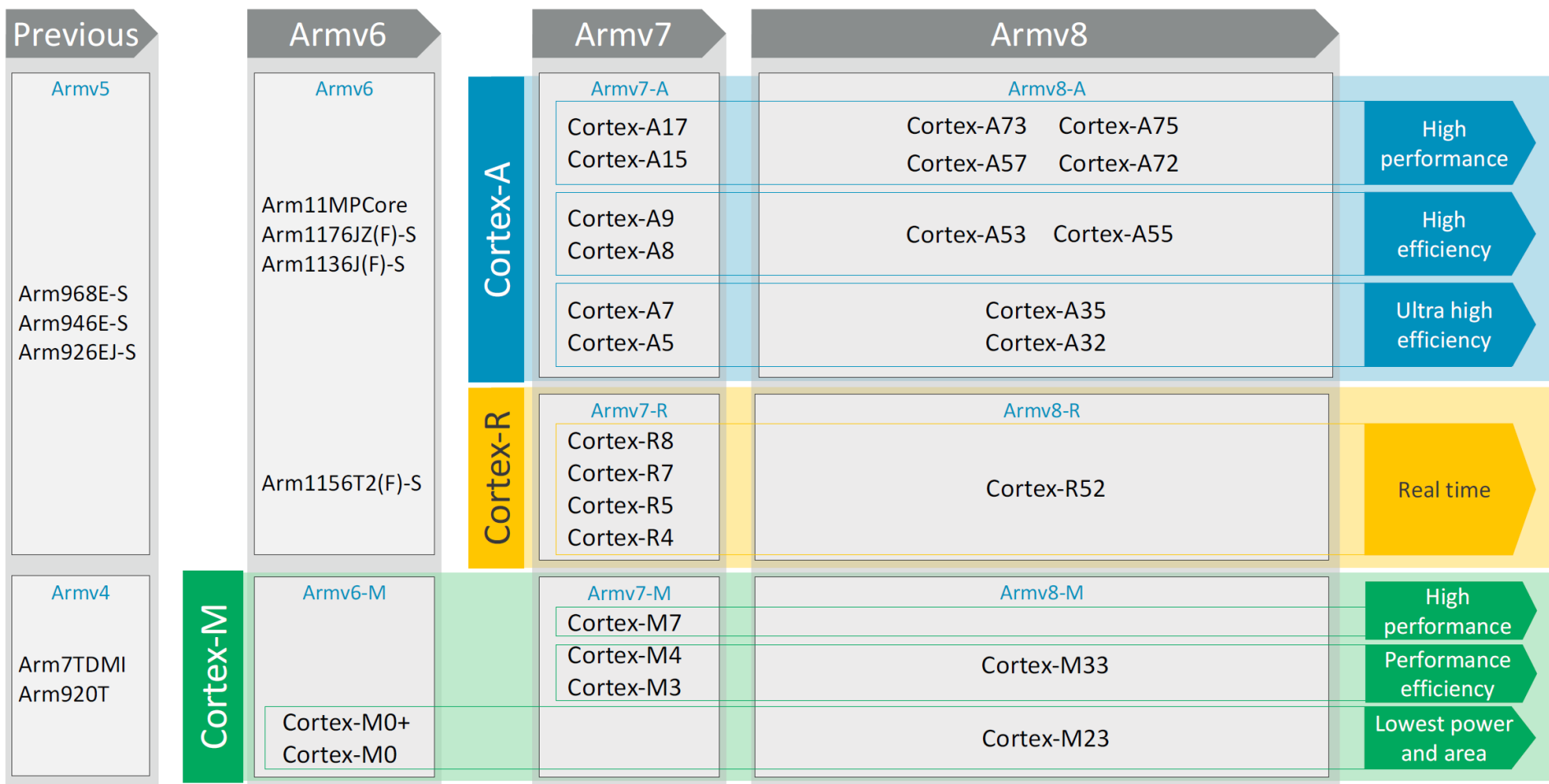
- Founded in November 1990
- Designs the ARM range of RISC processor cores
- Licenses ARM core designs to semiconductor partners who fabricate and sell to their customers
- Also develop technologies to assist with the design-in of the ARM architecture

ARM Partnership Model



Architecture Revisions

- Versions refer to the instruction set the ARM core executes



Inside An ARM Based System

- Hidden processor, debug the only visible port : JTAG/SWD
- Clocks and reset controllers + Interrupt controller
- On-chip interconnect bus architecture. For the vast majority ARM based system, this is the standard **AMBA** interconnect
- Two buses:
 - High perf system bus, AXI => Memory and other high speed devices
 - Low perf peripheral bus, APB => collect data from peripherals
- Some amount of on-chip memory and interfaces to external memory devices
- AMBA bus not exposed => not for external device interfaces

Development of the ARM Architecture

- v4T => v5TE => v6 => v7
- Continuous upgrade; each time adding new features but maintaining backward compatibility
- With v7, the concept of **Architecture Profile**: v7-A, v7-R, v7M
- Important difference between an architecture version and the implementation that supports such architecture
- An architecture defines how a processor behaves; its register set, instruction set, exception model, etc...
- The implementation behind can be significantly different but binary compatible (e.g. number of pipelines)

ARM Architecture v7 Profiles

- Application profile (ARMv7-A)
 - Memory management support (MMU) => virtual mem for Linux
 - Highest performance at low power
 - Influenced by multi-tasking OS system requirements
 - TrustZone for a safe extensible system
 - Optional Large Physical Address and Virtualization extensions
- Real-time profile (ARMv7-R)
 - Protected memory (MPU)
 - Low latency and predictability 'real time' needs
 - Tightly coupled memories for fast, deterministic access
 - No virtual memory support, but extension like low-interrupt latency
- Microcontroller profile (ARMv7-M)
 - Low gate count implementation
 - Deterministic & predictable behavior a key priority => fixed mem map
 - Deeply embedded use

Data Sizes and Instruction Sets

- The ARM is a 32-bit “RISC” load-store architecture
 - A 64-bit architecture in v8
 - Most instructions execute in a single cycle, orthogonal register set
 - Only memory accesses allowed are loads and stores
 - Most internal registers are 32-bit wide and processed by 32-bit ALU
- When used in relation to the ARM:
 - **Byte** means 8 bits
 - **Halfword** means 16 bits (two bytes)
 - **Word** means 32 bits (four bytes)
- Most ARMs implement two instruction sets
 - 32-bit ARM Instruction Set
 - 16/32-bit Thumb Instruction Set => greater density

Processor Modes (Cortex-A & R)

- The ARM has seven basic operating modes:
 - Each mode has access to its own stack space and a different subset of registers
 - Some operations can only be carried out in privileged mode
- | | | |
|-----------------|---|------------------|
| Exception modes | <ul style="list-style-type: none">• Supervisor (SVC) : entered on reset & when a SW Interrupt instruction is executed• FIQ : entered when a high priority (fast) interrupt is raised• IRQ : entered when a low priority (normal) interrupt is raised• Abort : used to handle memory access violations• Undef : used to handle undefined instructions | Privileged modes |
| | <ul style="list-style-type: none">• System : privileged mode using the same registers as user mode• User : unprivileged mode under which most tasks run | |

Processor Modes (Cortex-M)

- Two modes
 - **Thread** : Unprivileged, used for application code
 - **Handler** : Privileged, used for exception handling
- When the system resets, it starts in Thread mode, and automatically changes to Handler mode on an exception, returns to Thread mode when the handler completes
- System can be configured to have both modes privileged
- System can be configured to have both modes operate on the same stack

The ARM Register Set (Cortex-A & R)

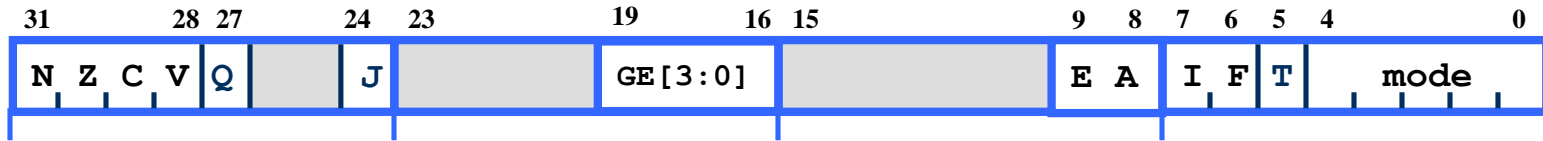
Current Visible Registers

| | |
|------------|----------|
| Abort Mode | r0 |
| | r1 |
| | r2 |
| | r3 |
| | r4 |
| | r5 |
| | r6 |
| | r7 |
| | r8 |
| | r9 |
| | r10 |
| | r11 |
| | r12 |
| | r13 (sp) |
| | r14 (lr) |
| r15 (pc) | |
| cpsr | |
| spsr | |

Banked out Registers

| | User | FIQ | IRQ | SVC | Undef |
|--|----------|----------|----------|----------|----------|
| | | r8 | | | |
| | | r9 | | | |
| | | r10 | | | |
| | | r11 | | | |
| | | r12 | | | |
| | r13 (sp) | r13 (sp) | r13 (sp) | r13 (sp) | r13 (sp) |
| | r14 (lr) | r14 (lr) | r14 (lr) | r14 (lr) | r14 (lr) |
| | | spsr | spsr | spsr | spsr |

Program Status Registers



- ALU Condition code flags (set & tested)
 - N = Negative result from ALU
 - Z = Zero result from ALU
 - C = ALU operation Carried out
 - V = ALU operation overflowed
- Sticky Overflow flag - Q flag
 - Architecture 5TE/J only
 - Indicates if saturation has occurred
- J bit
 - Architecture 5TEJ only
 - J = 1: Processor in Jazelle state
- GE[3:0] used by some SIMD instructions to record multiple results
- Interrupt Disable bits.
 - I = 1: Disables the IRQ.
 - F = 1: Disables the FIQ.
- T Bit
 - Architecture xT only
 - T = 0: Processor in ARM state
 - T = 1: Processor in Thumb state
- Mode bits
 - Specify the processor mode
 - Can be changed in privileged mode

Exception

- **Internal**

- Memory protection fault

- **External**

- Bus error

- **Synchronous**

- SVC instruction

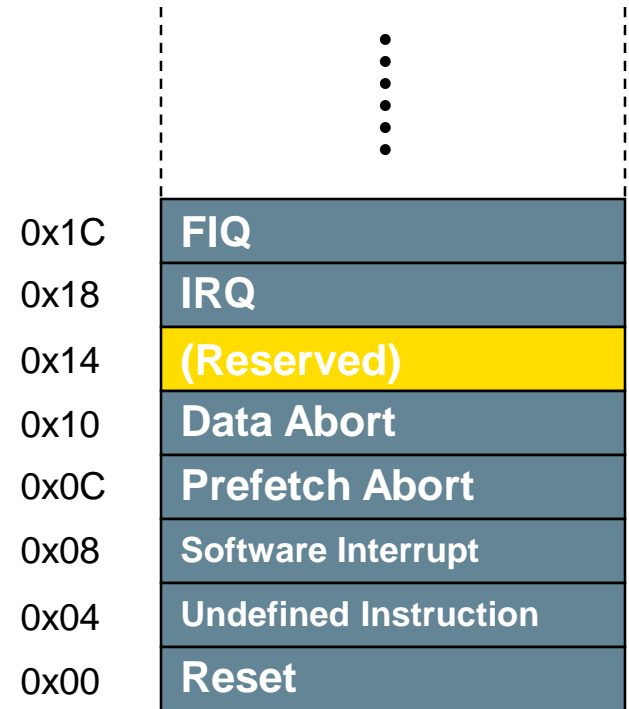
- **Asynchronous**

- Timer interrupt

Handled the same way

Exception Handling

- Save processor status
 - Copies CPSR into SPSR_<mode>
 - Stores the return address in LR_<mode>
- Change processor status for exception
 - Mode field bits
 - ARM or Thumb (T2) state
 - Interrupt disable bits (if appropriate)
 - Sets PC to vector address
- Execute exception handler
- Return to main application
 - Restore CPSR from SPSR_<mode>
 - Restore PC from LR_<mode>

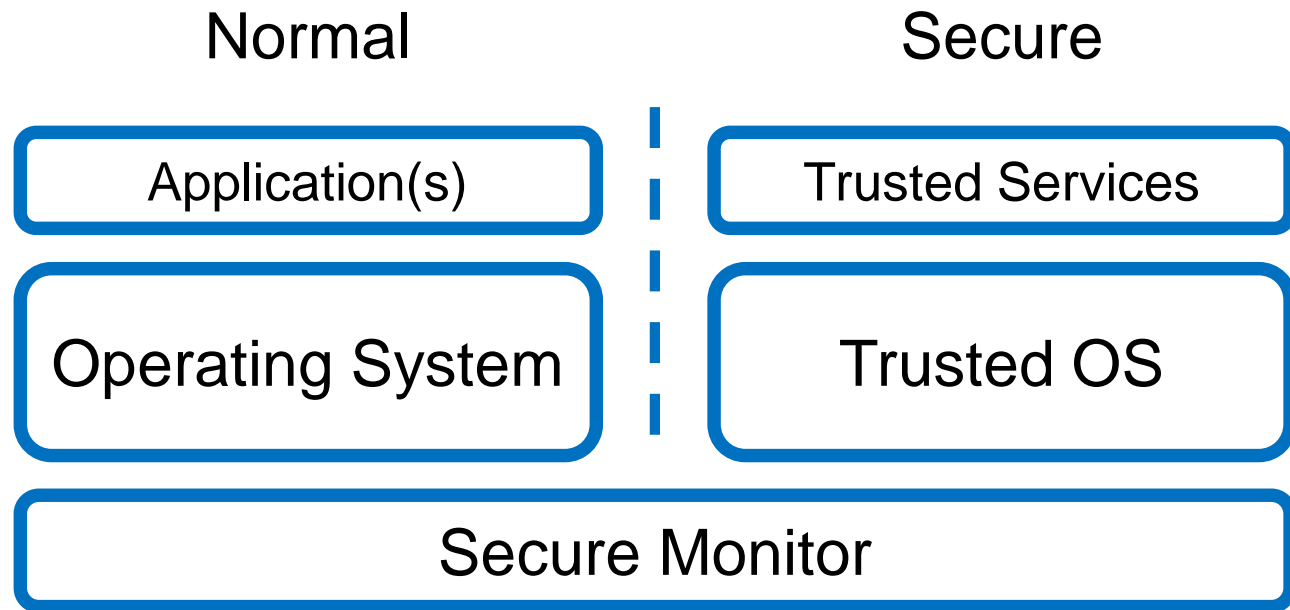


Vector Table

Vector table can be at
0xFFFF0000 on ARM720T
and on ARM9/10 family devices

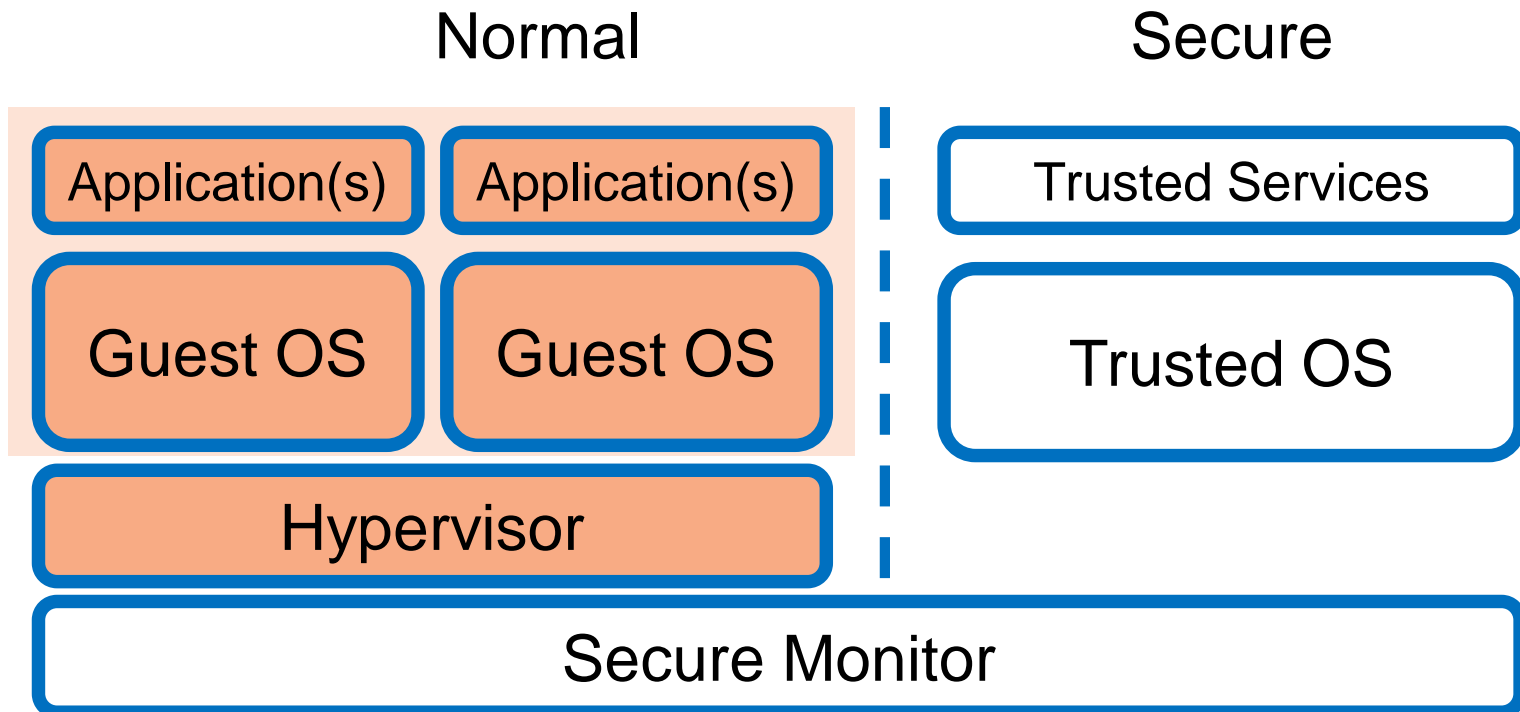
Security Extensions (TrustZone)

- Optional for v7-A
- Processor provides two worlds – “secure” and “normal”
- “monitor” mode acts as a gatekeeper for moving between worlds



Virtualization Extensions

- Optional for v7-A
- Processor provides two worlds – “secure” and “normal”
- “monitor” mode acts as a gatekeeper for moving between worlds



ARM Instruction Set

- Not all the details are here
- All the instructions are 32-bit long
- Most instructions can be conditionally executed
- Load/Store instruction set – no direct manipulation of memory content

```
SUB r0, r1, #5
```

```
r0 = r1 - 5
```

ARM Instruction Set

- Not all the details are here
- All the instructions are 32-bit long
- Most instructions can be conditionally executed
- Load/Store instruction set – no direct manipulation of memory content

```
ADD r2, r3, r3, LSL #2
```

$$r2 = r3 + (r3 * 4)$$

ARM Instruction Set

- Not all the details are here
- All the instructions are 32-bit long
- Most instructions can be conditionally executed
- Load/Store instruction set – no direct manipulation of memory content

```
ANDS r4, r4, #0x20
```

```
r4 = r4 & 0x20
```

ARM Instruction Set

- Not all the details are here
- All the instructions are 32-bit long
- Most instructions can be conditionally executed
- Load/Store instruction set – no direct manipulation of memory content

ADDEQ r5, r5, r6

if (EQ) $r5 = r5 + r6$

ARM Instruction Set

- Not all the details are here
- All the instructions are 32-bit long
- Most instructions can be conditionally executed
- Load/Store instruction set – no direct manipulation of memory content

B <Label>

PC-relative branch

ARM Instruction Set

- Not all the details are here
- All the instructions are 32-bit long
- Most instructions can be conditionally executed
- Load/Store instruction set – no direct manipulation of memory content

```
LDR r0, [r1]
```

```
r0 = *r1
```

ARM Instruction Set

- Not all the details are here
- All the instructions are 32-bit long
- Most instructions can be conditionally executed
- Load/Store instruction set – no direct manipulation of memory content

STRNEB r2, [r3, r4]

if (NE) $*(r3 + r4) = r2$

Thumb Instruction Set

- All instructions are 16-bit
- About 17% improvement in code density at the expense of performance

| | |
|---------|--------|
| ARM | 32-bit |
| Thumb | 16-bit |
| Thumb-2 | 32-bit |
| Thumb-2 | 16-bit |


Program Counter (r15)

- When the processor is executing in ARM state:
 - All instructions are 32 bits wide
 - All instructions must be word aligned
 - Therefore the **pc** value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned)
- When the processor is executing in Thumb state:
 - All instructions are 16 bits wide
 - All instructions must be halfword aligned
 - Therefore the **pc** value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned)

Conditional Execution and Flags

- ARM instructions can be made to execute conditionally by post-fixing them with the appropriate condition code field.
- This improves code density *and* performance by reducing the number of forward branch instructions.

```
CMP    r3, #0
BEQ    skip
ADD    r0, r1, r2
skip
```

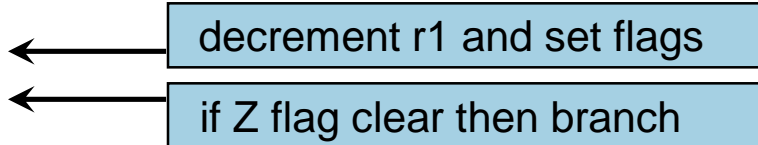


```
CMP    r3, #0
ADDNE  r0, r1, r2
```

- By default, data processing instructions do not affect the condition code flags but the flags can be optionally set by using “S”. CMP does not need “S”.

loop

```
...
SUBS  r1, r1, #1
BNE  loop
```



Condition Codes

- The possible condition codes are listed below
 - Note AL is the default and does not need to be specified

| Suffix | Description | Flags tested |
|--------------|-------------------------|----------------------|
| EQ | Equal | Z=1 |
| NE | Not equal | Z=0 |
| CS/HS | Unsigned higher or same | C=1 |
| CC/LO | Unsigned lower | C=0 |
| MI | Minus | N=1 |
| PL | Positive or Zero | N=0 |
| VS | Overflow | V=1 |
| VC | No overflow | V=0 |
| HI | Unsigned higher | C=1 & Z=0 |
| LS | Unsigned lower or same | C=0 or Z=1 |
| GE | Greater or equal | N=V |
| LT | Less than | N!=V |
| GT | Greater than | Z=0 & N=V |
| LE | Less than or equal | Z=1 or N=!V |
| AL | Always | |

Conditional execution examples

C source code

```
if (r0 == 0)
{
    r1 = r1 + 1;
}
else
{
    r2 = r2 + 1;
}
```

ARM instructions

unconditional

```
CMP r0, #0
BNE else
ADD r1, r1, #1
B end
else
    ADD r2, r2, #1
end
...
```

conditional

```
CMP    r0, #0
ADDEQ r1, r1, #1
ADDNE r2, r2, #1
...
```

- 5 instructions
- 5 words
- 5 or 6 cycles

- 3 instructions
- 3 words
- 3 cycles

Data Processing Instructions

- Consist of :

- Arithmetic: **ADD** **ADC** **SUB** **SBC** **RSB** **RSC**
- Logical: **AND** **ORR** **EOR** **BIC**
- Comparisons: **CMP** **CMN** **TST** **TEQ**
- Data movement: **MOV** **MVN**

- These instructions only work on registers, NOT memory.

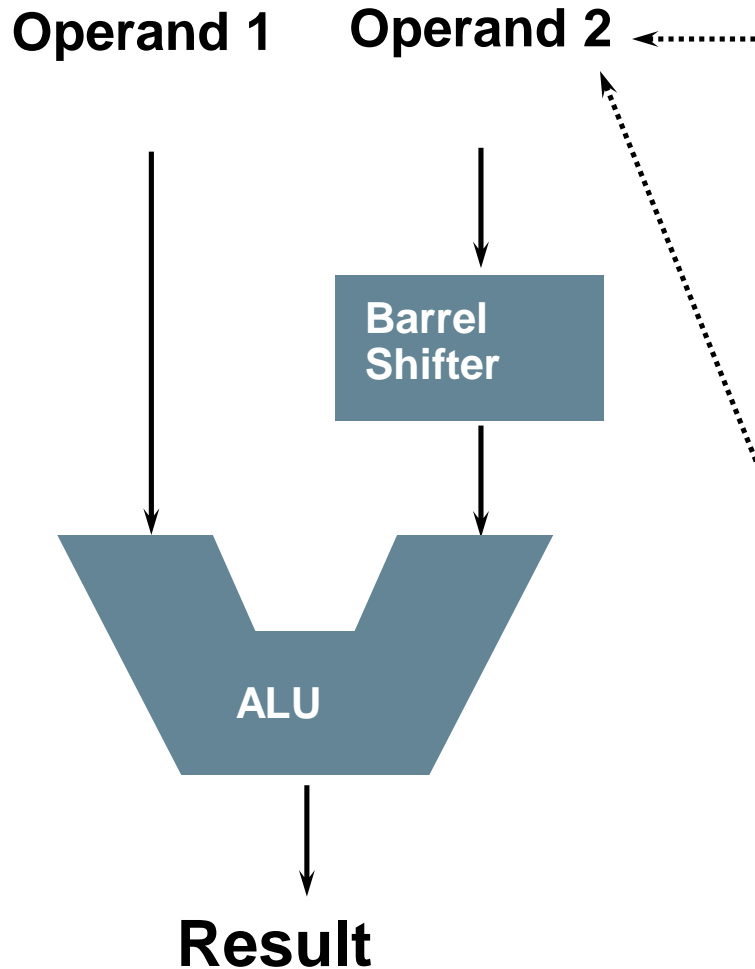
- Syntax:

<Operation>{<cond>}{S} Rd, Rn, Operand2

- Comparisons set flags only - they do not specify Rd
- Data movement does not specify Rn

- Second operand is sent to the ALU via barrel shifter.

Using a Barrel Shifter: The 2nd Operand



Register, optionally with shift operation

- Shift value can be either:
 - 5 bit unsigned integer
 - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value

- 8 bit number, with a range of 0-255.
 - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers

Data Processing Exercise

1. How would you load the two's complement representation of -1 into Register 3 using one instruction?
2. Implement an ABS (absolute value) function for a registered value using only two instructions.
3. Multiply a number by 35, guaranteeing that it executes in 2 core clock cycles.

Data Processing Solutions

1. MOVN r6, #0

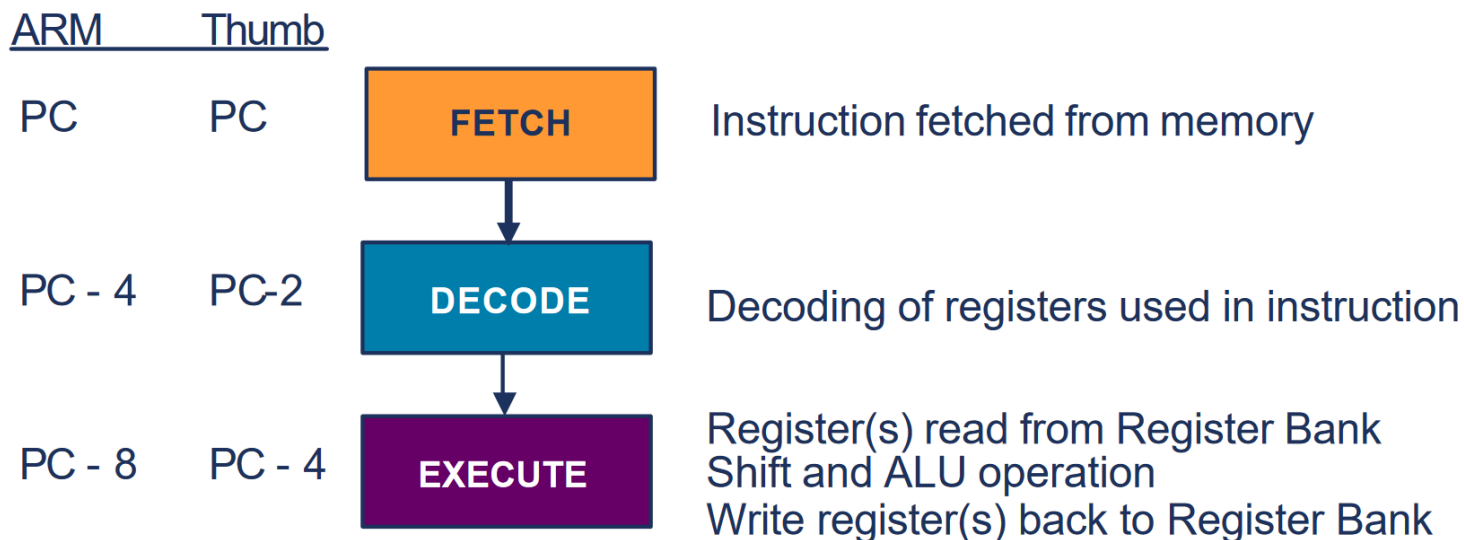
2. MOVS r7,r7 ; set the flags
RSBMIr7,r7,#0 ; if neg, r7=0-r7

3. ADD r9,r8,r8,LSL #2 ; r9=r8*5
RSB r10,r9,r9,LSL #3 ; r10=r9*7

The Instruction Pipeline

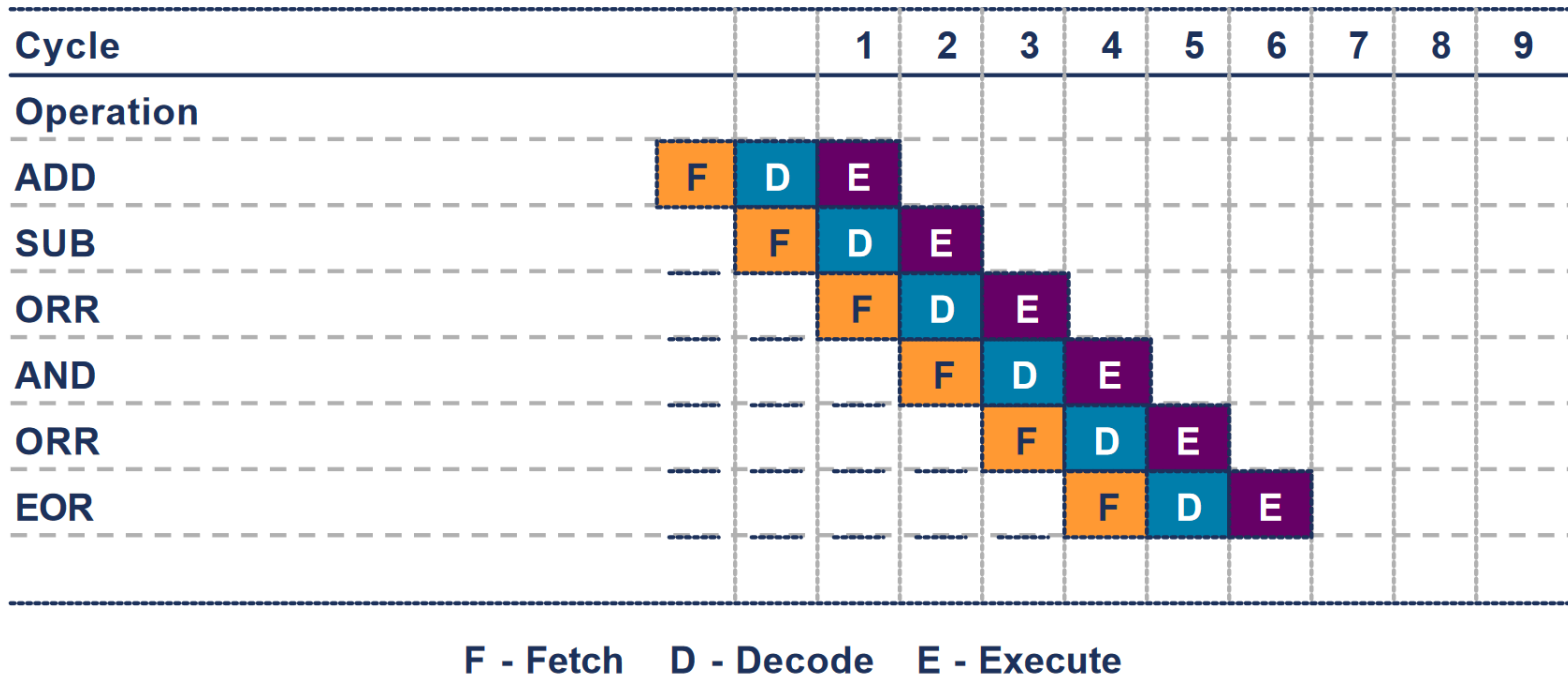
The Instruction Pipeline

- The ARM7TDMI uses a 3 stage pipeline in order to increase the speed of the flow of instructions to the processor
 - Allows several operations to be performed simultaneously, rather than serially



- The PC points to the instruction being fetched, not executed
 - Debug tools will hide this from you
 - This is now part of the ARM Architecture and applies to all processors

Optimal Pipelining



- All operations here are on registers (single cycle execution)
- In this example it takes 6 clock cycles to execute 6 instructions
- Clock cycles per Instruction (CPI) = 1

Branch Pipelining Example

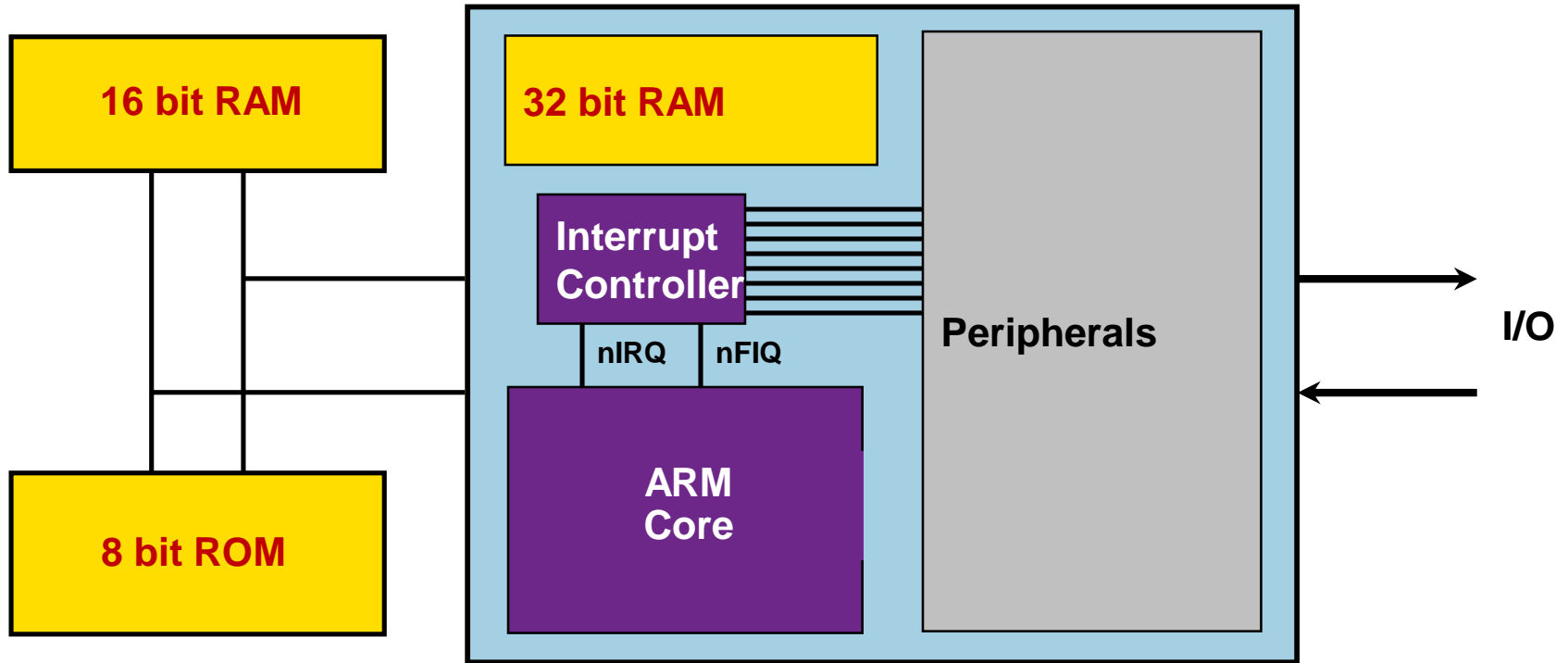
| Cycle | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---------|-----------|---|---|---|----------------|----------------|---|---|---|---|
| Address | Operation | | | | | | | | | |
| 0x8000 | BL 0x8FEC | F | D | E | E _L | E _A | | | | |
| 0x8004 | SUB | | F | D | | | | | | |
| 0x8008 | ORR | | | F | | | | | | |
| 0x8FEC | AND | | | | F | D | E | | | |
| 0x8FF0 | ORR | | | | | F | D | E | | |
| 0x8FF4 | EOR | | | | | | F | D | E | |

F - Fetch D - Decode E - Execute L - Linkret A - Adjust

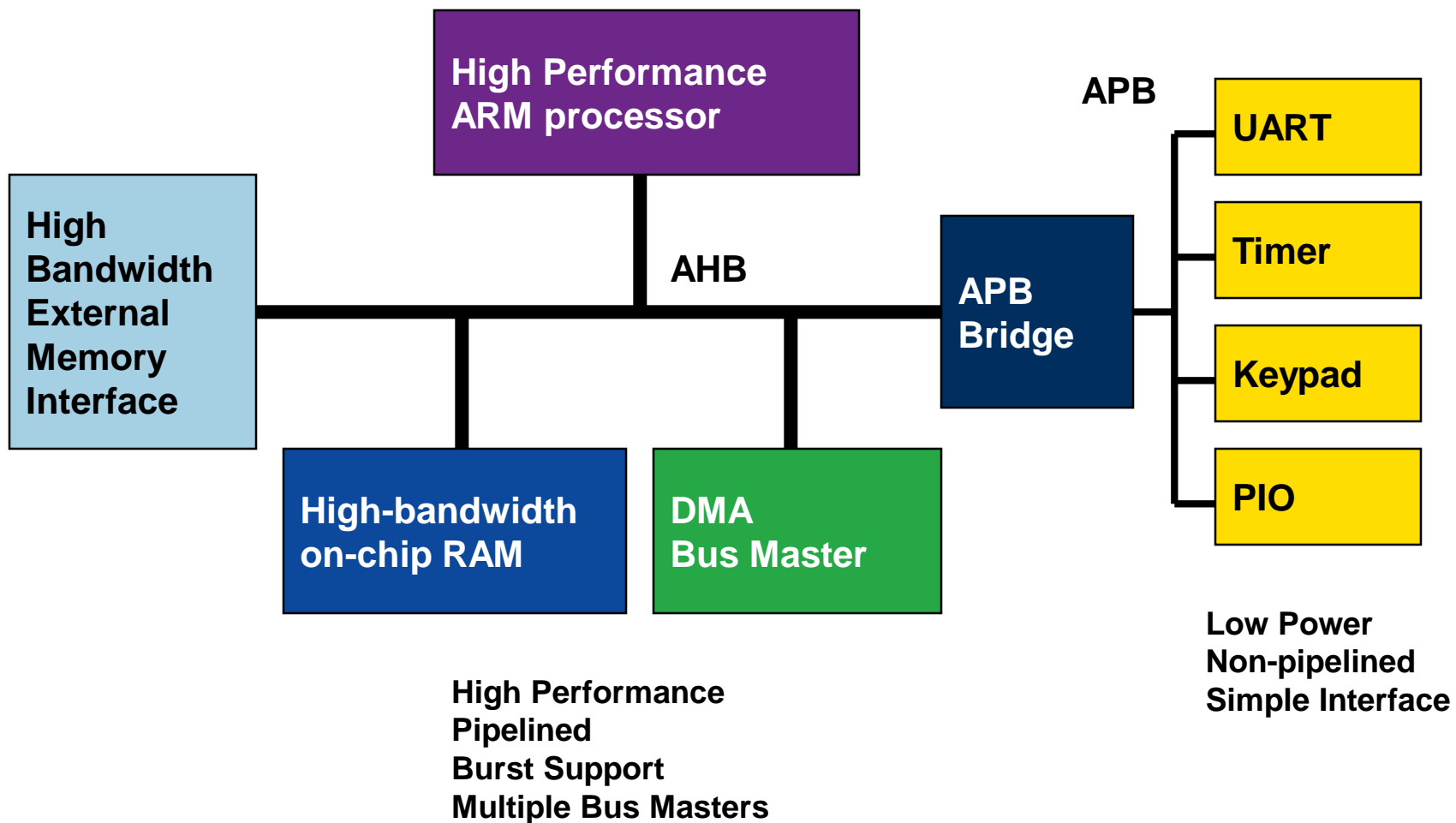
- Breaking the pipeline
- Note that the core is executing in ARM state

AMBA

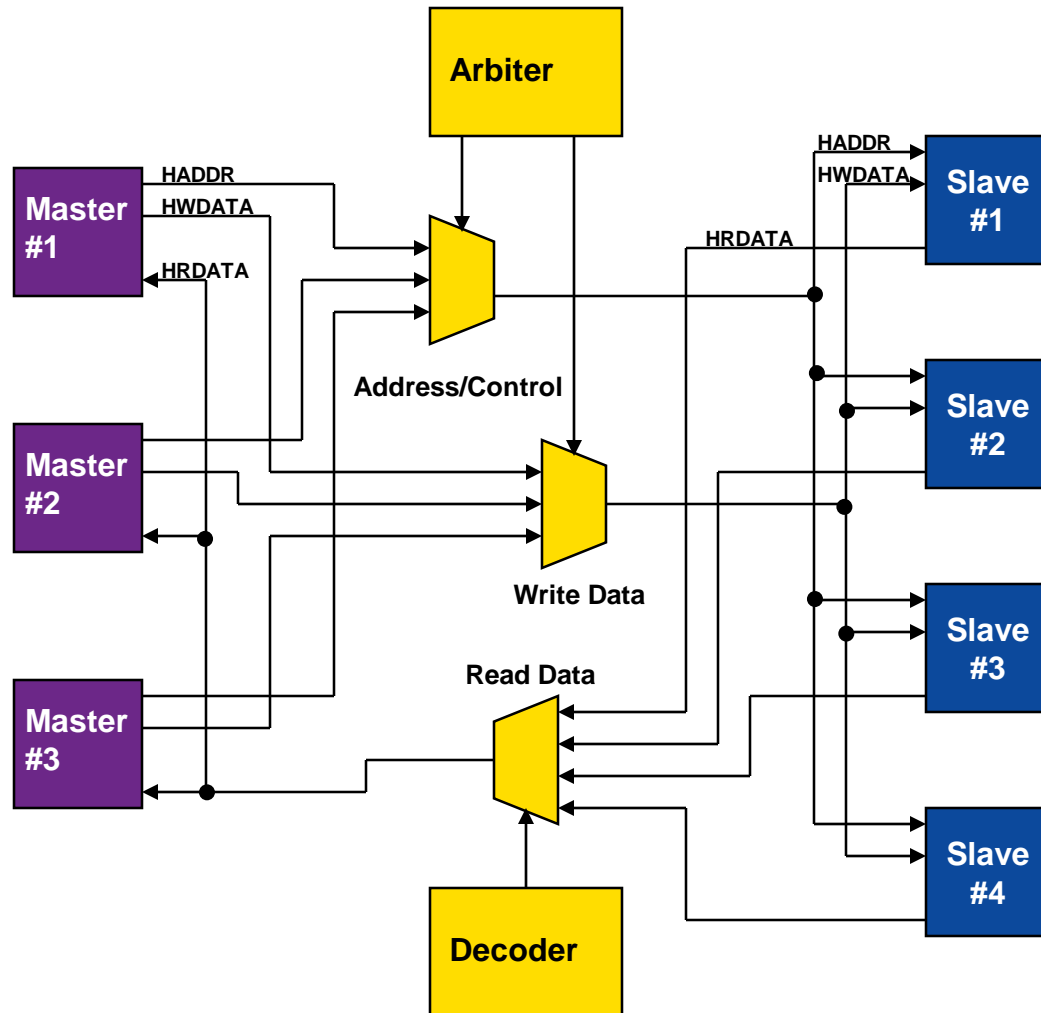
Example ARM-based System



An Example AMBA System

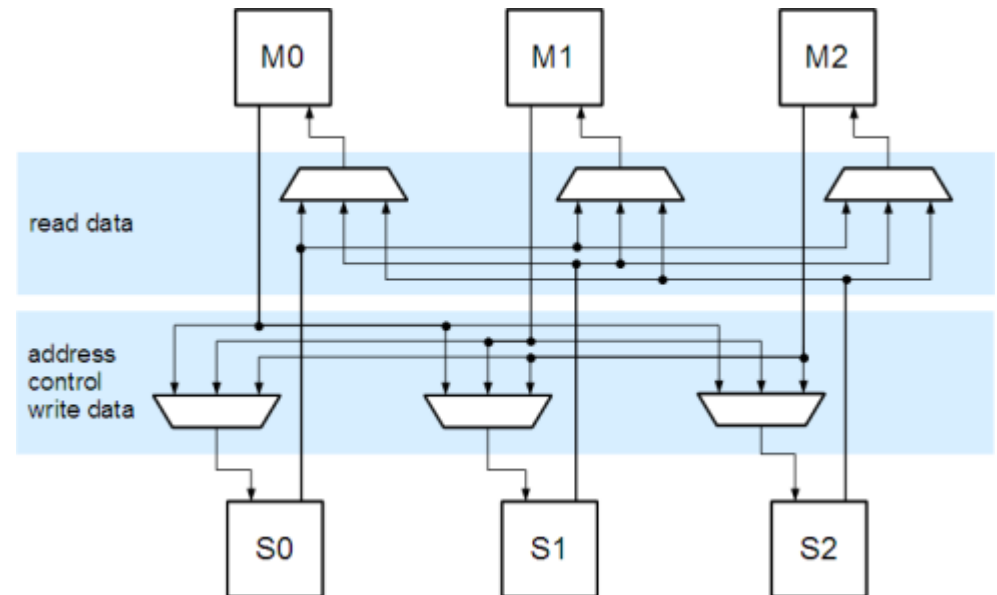


AHB Structure



Multi-layer AHB and AHB-Lite

- Each layer is an independent single master AHB system
- A multi-layer AHB with M masters and S slaves is structured as $M \times 1:S$ multiplexers plus $S \times M:1$ slave multiplexers all connected to separate arbitration and decoding logic
- Multiple masters can talk to multiple slaves concurrently, as long as no two masters don't try to access the same slave at the same time
(e.g. a DMA controller moving data from a receiver into a memory region, while the processor continues to execute code in a different memory region)
- Became AHB-Lite



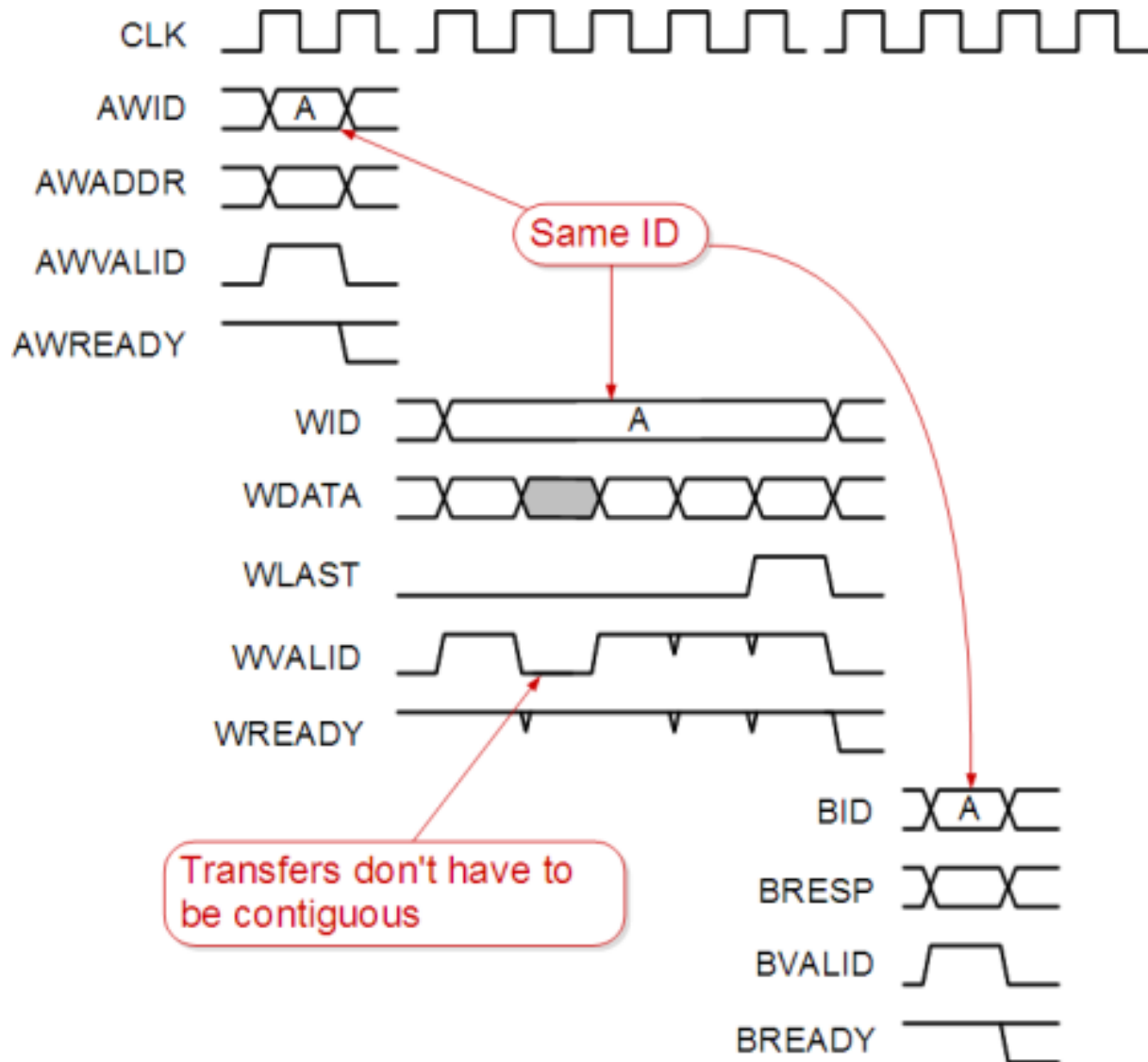
Multi-layer AHB and AHB-Lite

- With modern SoC, the system fabric poses a critical performance bottleneck. The reasons for this include:
- AHB is transfer-oriented:
 - address submitted => a single data item written to or read from the selected slave
 - All transfers initiated by the master. If the slave cannot respond immediately to a transfer request the master will be stalled
 - Each master can have only one outstanding transaction
- Sequential accesses (bursts) consist of consecutive transfers which indicate their relationship by asserting HTRANS/HBURST accordingly
- Although AHB systems are multiplexed and thus have independent read and write data busses, they cannot operate in full-duplex mode.

AXI (Advanced eXtensible Interface)

- **Up to five channels** (write address, write data, write response, read address, read data/response)
- Can operate largely independently of each other
- Each channel uses the same trivial **handshaking** between source and destination => simplifies the interface design
- In AXI3 transactions are **bursts of lengths between 1 and 16**
- Each transaction consists of address, data, and response transfers on their corresponding channels
- Every transfer identifies itself as part of a specific transaction by its **transaction ID tag**
- Transactions **may complete out-of-order** and transfers belonging to different transactions may be **interleaved**. Thanks to the ID that every transfer carries, out-of-order transactions can be sorted out at the destination

Example: AXI Write Burst



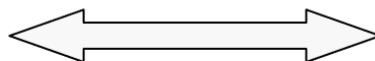
Development Tools

ARM Debug Architecture

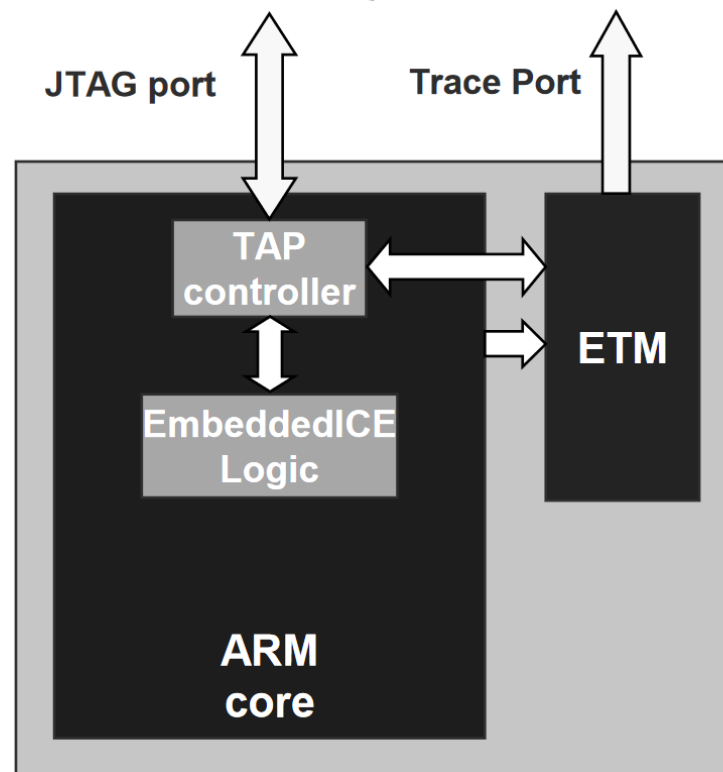
Debugger (+ optional trace tools)



Ethernet



- EmbeddedICE Logic
 - Provides breakpoints and processor/system access
- JTAG interface (ICE)
 - Converts debugger commands to JTAG signals
- Embedded trace Macrocell (ETM)
 - Compresses real-time instruction and data access trace
 - Contains ICE features (trigger & filter logic)
- Trace port analyzer (TPA)
 - Captures trace in a deep buffer



Keil Development Tools for ARM



- Includes ARM macro assembler, compilers (ARM RealView C/C++ Compiler, Keil CARM Compiler, or GNU compiler), ARM linker, Keil uVision Debugger and Keil uVision IDE
- Keil uVision Debugger accurately simulates on-chip peripherals (I²C, CAN, UART, SPI, Interrupts, I/O Ports, A/D and D/A converters, PWM, etc.)
- Evaluation Limitations
 - 16K byte object code limitation
 - Some linker restrictions such as base addresses for code/constants
 - GNU tools provided are not restricted in any way
- <http://www.keil.com/demo/>

Keil Development Tools for ARM

The screenshot displays the Keil Development Tools IDE for an ARM microcontroller. The main window shows a C program named 'Hello.c' for the LPC2100 target. The code includes standard headers and a main function that initializes the serial interface and prints 'Hello World'.

Project Workspace:

| Register | Value |
|----------|------------|
| Current | |
| R0 | 0x0000000c |
| R1 | 0x0000000c |
| R2 | 0x00000020 |
| R3 | 0x0000018d |
| R4 | 0x00000000 |
| R5 | 0x00000000 |
| R6 | 0x00000000 |
| R7 | 0x00000000 |
| R8 | 0x00000000 |
| R9 | 0x00000000 |
| R10 | 0x00000000 |

Symbols:

| Name | Type |
|-----------------|--------|
| Simulator VTREG | |
| Peripheral SFR | |
| ALDOM | uchar |
| ALDOW | uchar |
| ALDOY | ushort |
| ALHOUR | uchar |
| ALMIN | uchar |
| ALMON | uchar |
| ALSEC | uchar |

Code Editor:

```
01 /* *****/
02 /* This file is part of the uVision/ARM development tools */
03 /* Copyright KEIL ELEKTRONIK GmbH 2002-2004 */
04 /* *****/
05 /*
06 /* HELLO.C: Hello World Example
07 /*
08 /* *****/
09
10 #include <stdio.h> /* prototype declarations for I/O functions */
11 #include <LPC21xx.H> /* LPC21xx definitions */
12
13
14 /* *****/
15 /* main program */
16 /* *****/
17 int main (void) { /* execution starts here */
18
19 /* initialize the serial interface */
20 PINSELO = 0x00050000; /* Enable RxD1 and TxD1 */
21 U1LCR = 0x83; /* 8 bits, no Parity, 1 Stop bit */
22 U1DLL = 97; /* 9600 Baud Rate @ 15MHz VPB Clock */
23 U1LCR = 0x03; /* DLAB = 0 */
24
25 printf ("Hello World\n"); /* the 'printf' function call */
26
27 while (1) { /* An embedded program does not stop and */
```

Output Window:

```
MISSING DEVICE (R003: SECURITY KEY NOT FOUND)
Running in Eval Mode
Load "C:\\Keil\\ARM\\Examples\\Hello\\Obj\\Hello.ELF"

*** Restricted Version with 16384 Byte Code Size Limi
*** Currently used: 1980 Bytes (12%)

>
ASSIGN BreakDisable BreakEnable BreakKill BreakList
```

Memory Window:

| Address | Value |
|------------|---|
| 0x00004000 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x0000400D | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x0000401A | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x00004027 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x00004034 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x00004041 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x0000404E | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 0x0000405B | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |

Ready Simulation t1: 0.72642057 sec L:29 C:1