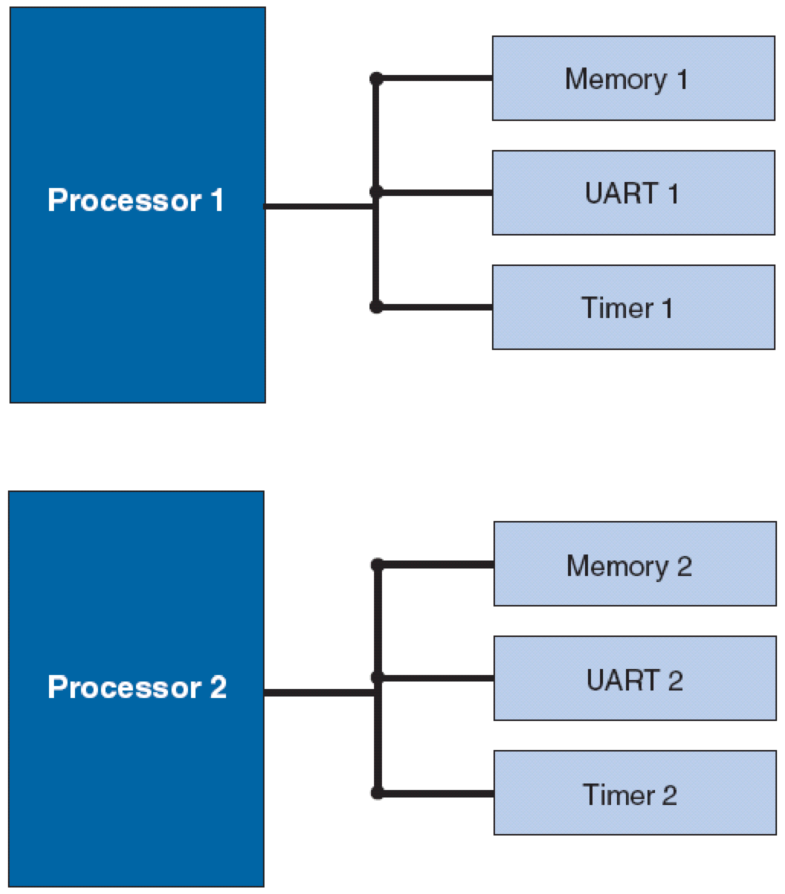
« Real Time Embedded systems » Multi Masters Systems

rene.beuchat@epfl.ch
LAP/ISIM/IC/EPFL
Chargé de cours
rene.beuchat@hesge.ch
LSN/hepia
Prof. HES

Multi Master on Chip

- On a System On Chip, Master can be:
  - A Processor
  - A DMA Master module
  - An accelerator composed of 1-2 DMA units
- Masters can communicate by:
  - Shared common memory
  - Network On Chip
  - Specialized interfaces with local memory as hardware mailbox

Simple Multi processors system

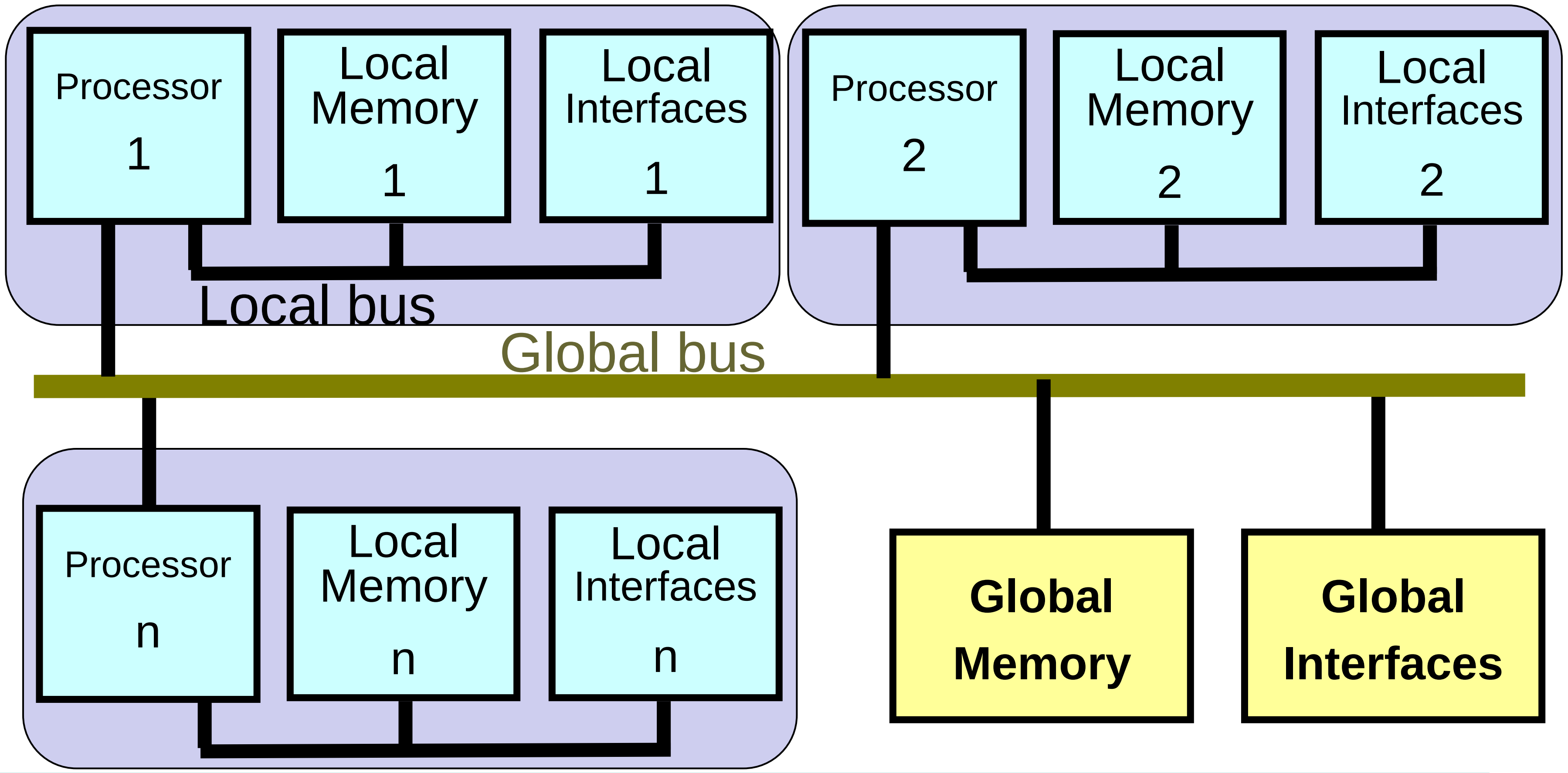


- Independents processors
- Each processor has it's own local memory and programmable interface
- NO communication between processors

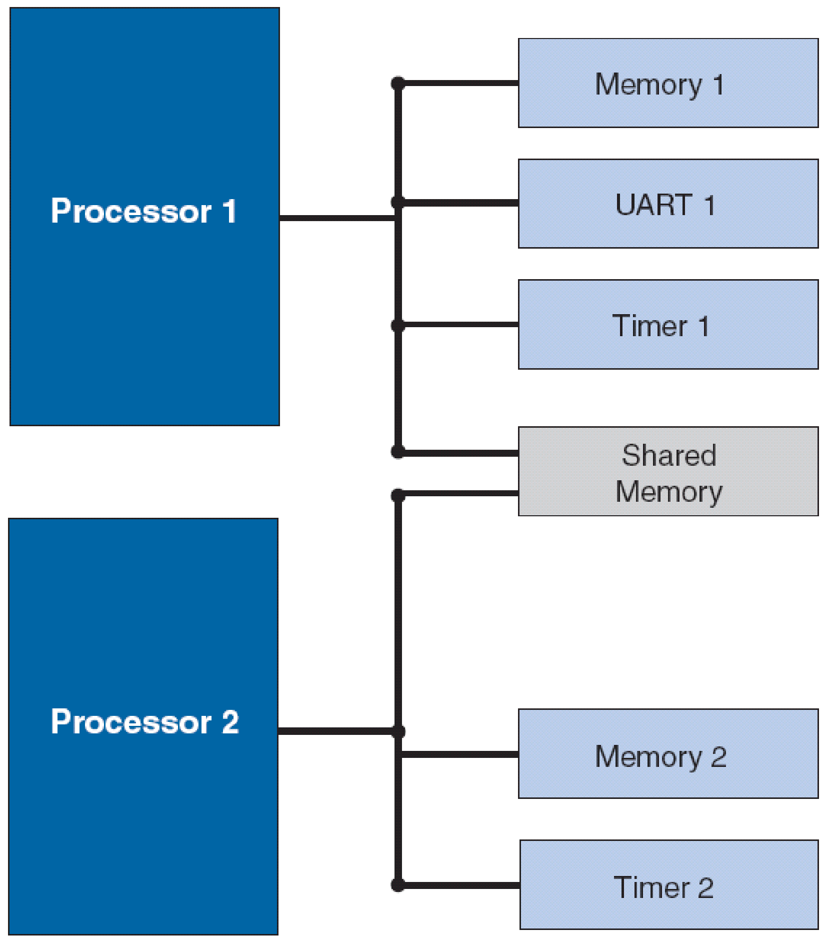
Multi Masters architecture

- Each processor has it's **own local memory** and/or cache memory and/or programmable interfaces
- A **global memory** is shared by all processor
- Mutual exclusion primitives are necessary for access to common resources

Multi Masters architecture



Multi Masters architecture



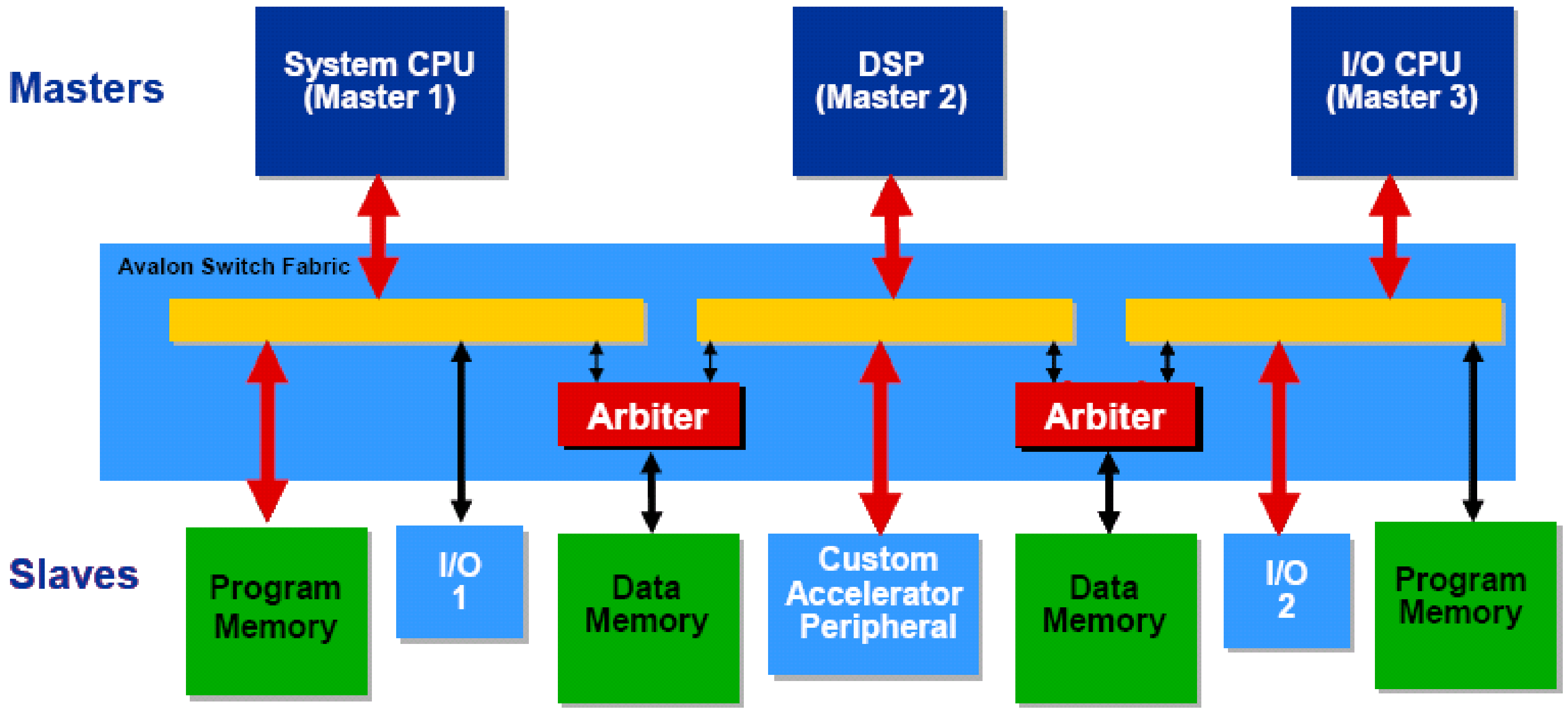
- Example with 2 processors :
  - ➢ Processor 1:
    - ➢ Memory 1
    - ➢ UART1
    - ➢ Timer 1
  - ➢ Processor 2:
    - ➢ Memory 2
    - ➢ Timer 2
  - ➢ Shared Memory

Multiprocessors

- Symmetrical Multi Processors (SMP) are widely used in workstation and high end PC. They are quite complexes. Tasks are send to a free processor.
- On an embedded system, especially FPGA bases embedded systems, **asymmetric** multiprocessor systems are common now. The hardware development is not too difficult.

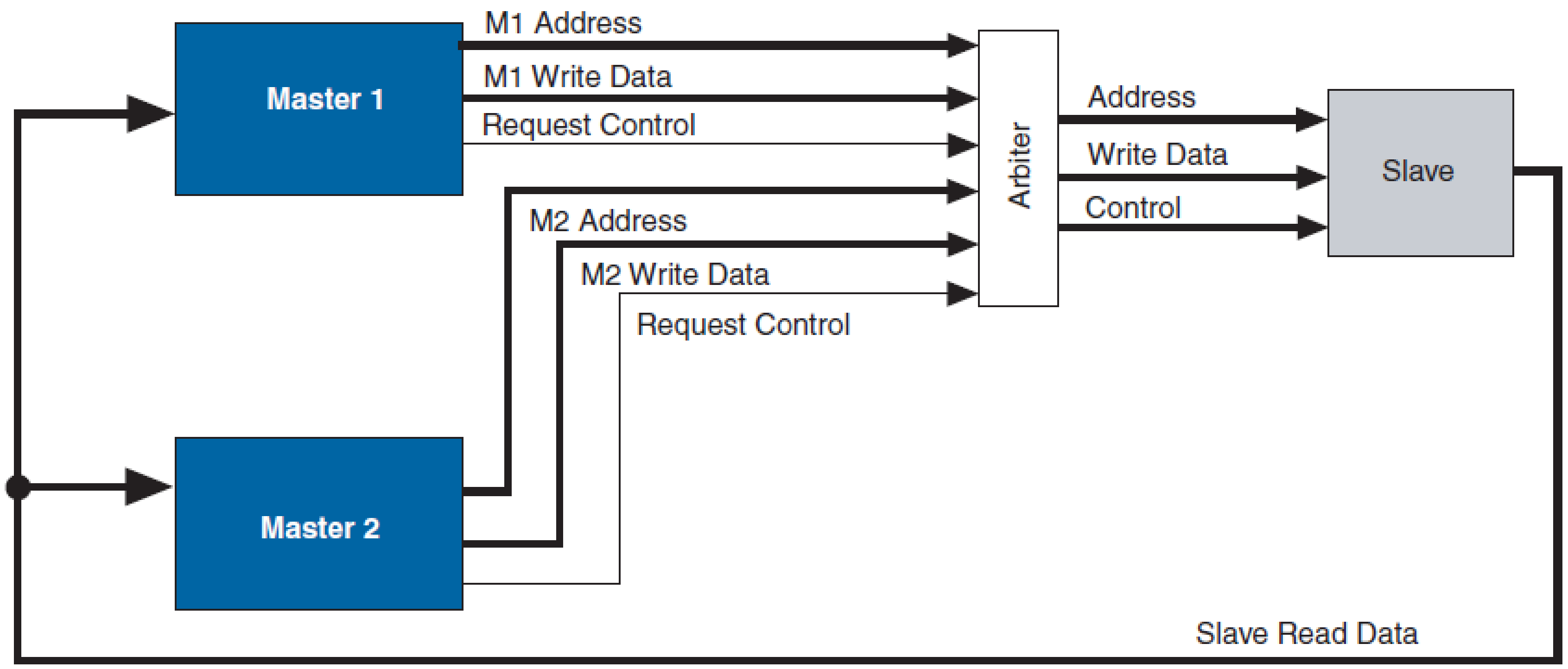
Multiprocessors

- Example of asymmetrical multiprocessor on Avalon system

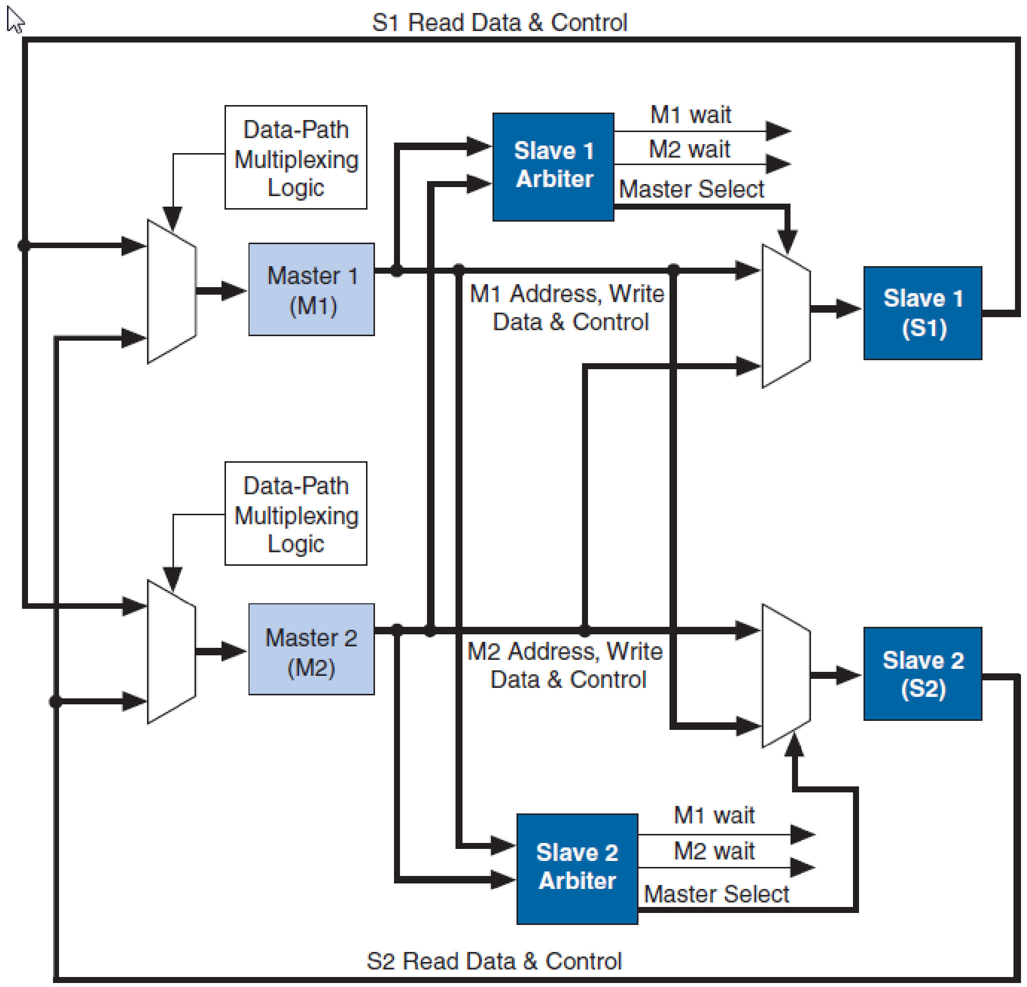


Multi Masters, Avalon interconnection

The arbiter is at the Slave level on the Avalon Bus

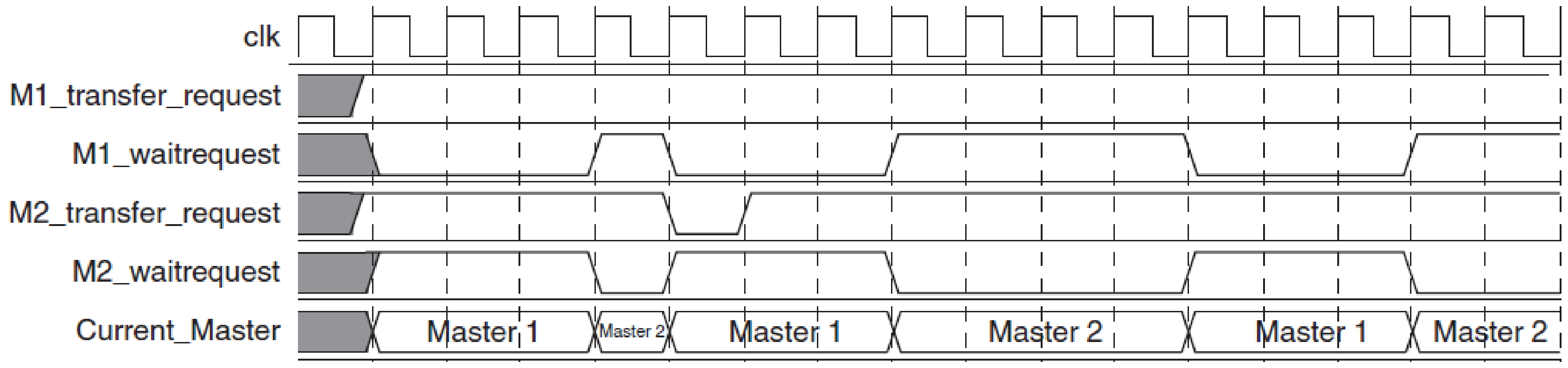


Multi Master, Slave Arbiter view



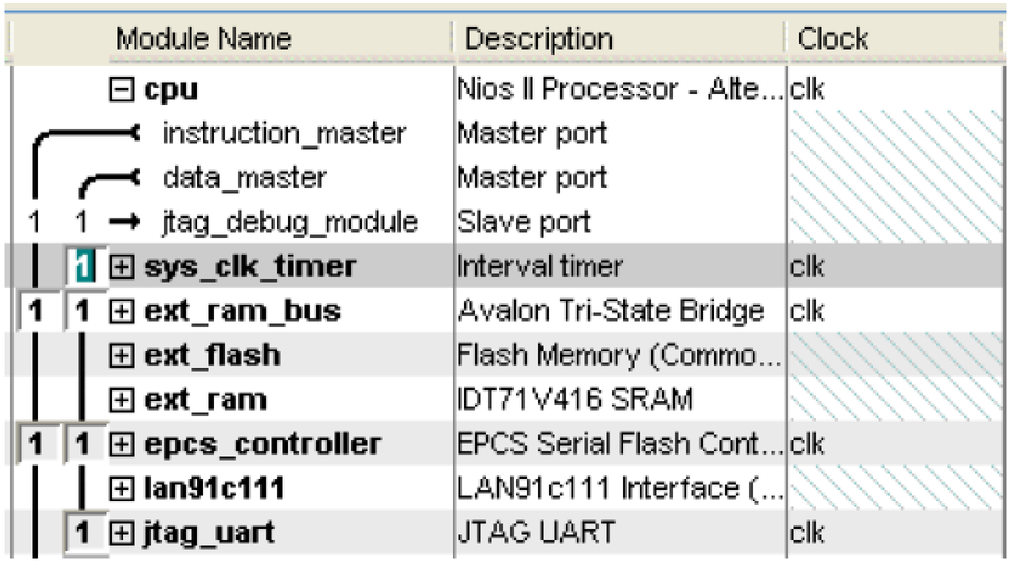
Multi master round robin fairness arbitration

- With SOPC, the number of authorized consecutive cycles for one master can be specified,
- ex: Master 1: 3, Master 2: 4
- If a master don't use all of its *shares*, it will wait until its next slot



Multi master round robin fairness arbitration

- SOPC: arbitration and bus selection
- For each slave, the number of shares between master is configurable
- *View menu → Show Arbitration*



		Module Name	Description	Clock
		⊟ **cpu**	Nios II Processor - Alte...	clk
		instruction_master	Master port	
		data_master	Master port	
1	1	jtag_debug_module	Slave port	
	1	⊞ **sys_clk_timer**	Interval timer	clk
1	1	⊞ **ext_ram_bus**	Avalon Tri-State Bridge	clk
		⊞ **ext_flash**	Flash Memory (Commo...	
		⊞ **ext_ram**	IDT71V416 SRAM	
1	1	⊞ **epcs_controller**	EPCS Serial Flash Cont...	clk
		⊞ **lan91c111**	LAN91c111 Interface (...	
	1	⊞ **jtag_uart**	JTAG UART	clk

Multiprocessors, resource sharing

- Resources sharing with mutual exclusion have to be realized
- A specific hardware interface **Mutex Core** is available on Altera Avalon based systems, the mutex core has the following basic behavior :
  - ➢There are multiple processors accessing a **single mutex core (it's a Global Interface)**
  - ➢Each processor has a unique identifier (ID)

Multiprocessors exclusion primitives

- To access a common resource **mutual exclusion primitive** is needed
- A **indivisible Read-Modify-Write cycle** is necessary
- Depending on the architecture this function can be implemented at:
    - ➢ Bus level with a **lock bus** primitive
    - ➢ Peripheral level with **mutex** programmable interface primitive
    - ➢ **Message passing** synchronization
    - ➢ …similar functionality

Multiprocessors exclusion primitives

- The principle is the same as semaphore primitive in multi-thread on a mono processor system. In this case uninterruptible access to a semaphore (single bit or counter) is needed.
- In multi processor system locked read-modify-write of a semaphore is needed

- Bus level with a lock bus primitive
  - ➢ Possible if the bus provides this primitive
  - ➢ The master needs to be able to use this primitive

Multiprocessors (mutex)

- **Peripheral level with mutex primitive:**

- A special register has 2 fields:
  - **Processor ID** and **Value**
- When value is 0, the mutex is unlocked (free)
- Otherwise, the mutex is locked (busy)
- The mutex register can always be read by any processor
- When the mutex is locked, the Processor ID contain the value of the locking processor

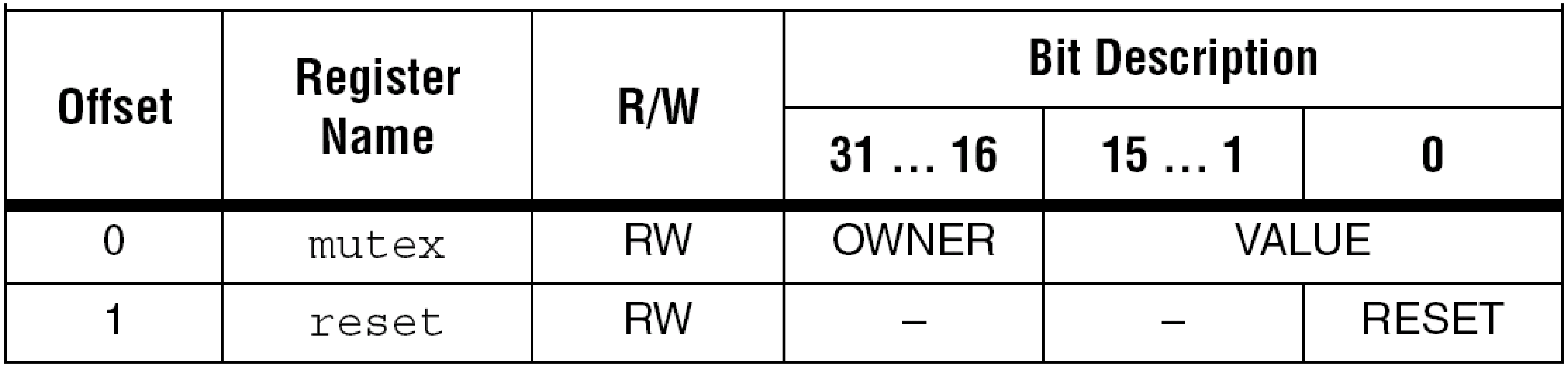
Multiprocessors (mutex)

- A write operation changes the mutex register only if one or both of the following conditions is true:
  - ➢ The VALUE field of the mutex register is zero.
  - ➢ The OWNER field of the mutex register matches the OWNER field in the data to be written.

Multiprocessors (mutex)

- A processor attempts to acquire the mutex by writing its ID to the OWNER field, and writing a non-zero value to VALUE.
- The processor then checks if the acquisition succeeded by verifying the OWNER field.

Multiprocessors (mutex)

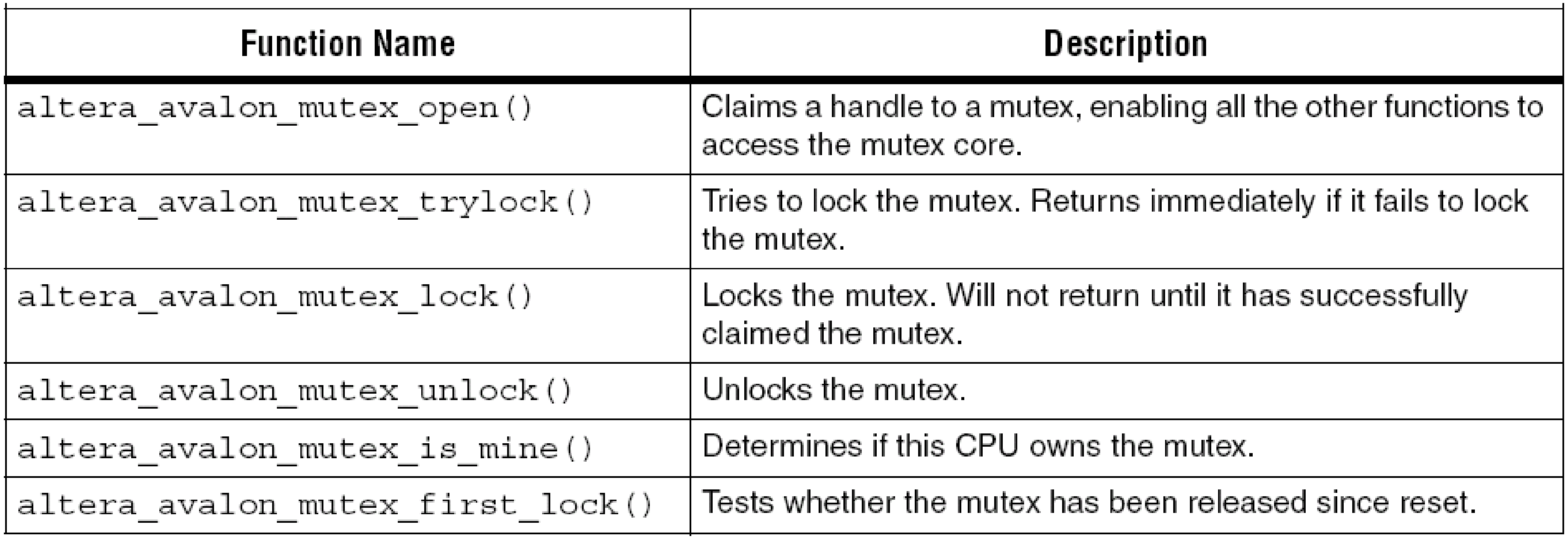


Offset	Register Name	R/W	Bit Description		
			31 … 16	15 … 1	0
0	`mutex`	RW	OWNER	VALUE	
1	`reset`	RW	–	–	RESET

- Mutex register,
  - ➢ Owner: CPU ID
  - ➢ Value : can be modified by CPU owner or every processor if Owner = 0
- Reset = 1 at reset and must be cleared
- An Owner can be initialized at FPGA compile time

Multiprocessors (mutex)

- Avalon Mutex functions



Function Name	Description
`altera_avalon_mutex_open()`	Claims a handle to a mutex, enabling all the other functions to access the mutex core.
`altera_avalon_mutex_trylock()`	Tries to lock the mutex. Returns immediately if it fails to lock the mutex.
`altera_avalon_mutex_lock()`	Locks the mutex. Will not return until it has successfully claimed the mutex.
`altera_avalon_mutex_unlock()`	Unlocks the mutex.
`altera_avalon_mutex_is_mine()`	Determines if this CPU owns the mutex.
`altera_avalon_mutex_first_lock()`	Tests whether the mutex has been released since reset.

Multiprocessors (mutex)

Example: Opening and locking a mutex

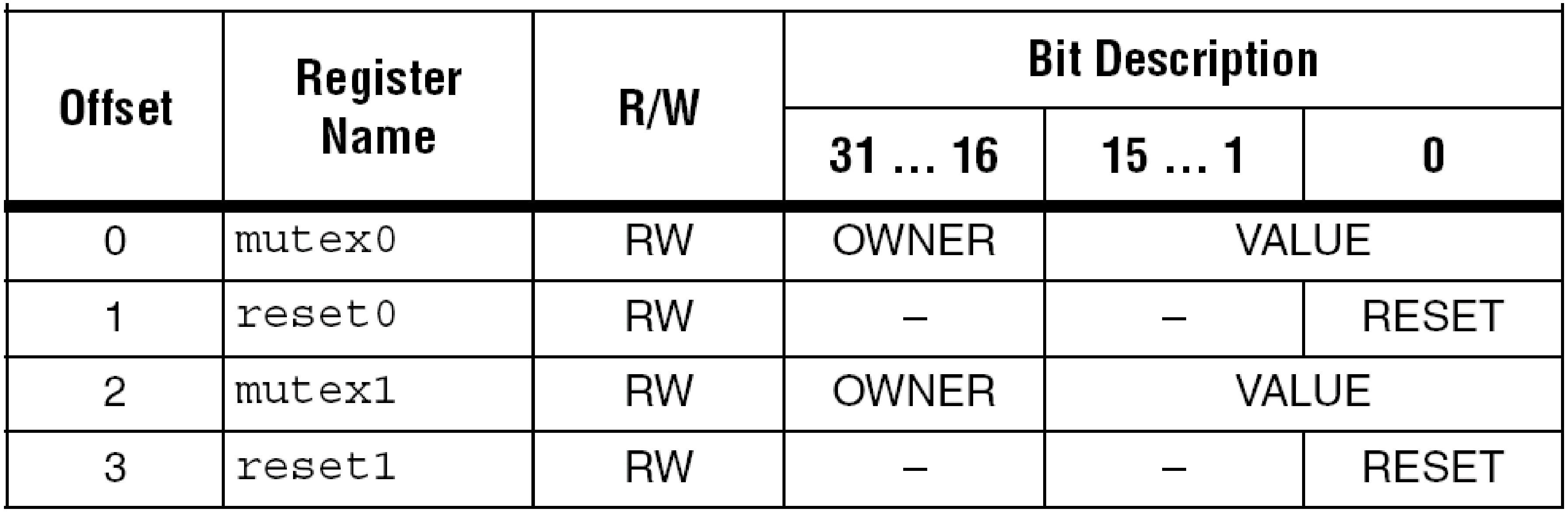
```
#include <altera_avalon_mutex.h>

/* get the mutex device handle */
alt_mutex_dev* mutex = altera_avalon_mutex_open( "/dev/mutex" );

/* acquire the mutex, setting the value to one */
altera_avalon_mutex_lock( mutex, 1 );

/*
 * Access a shared resource here.
 */

/* release the lock */
altera_avalon_mutex_unlock( mutex );
```

Multiprocessors (Mailbox)

- Multiprocessor environments can use the **mailbox core** with Avalon® interface (the mailbox core) to send messages between processors.
- The mailbox core contains mutexes to ensure that only one processor modifies the mailbox contents at a time. The mailbox core must be used in conjunction with a **separate shared memory** which is used for storing the actual messages.
- The mailbox core is designed for use in Avalon-based processor systems, such as a Nios® II processor system.
- The mailbox core is SOPC Builder-ready and integrates easily into any SOPC Builder-generated system.

Multiprocessors (Mailbox) deprecated

- Mailbox functionality using the mutexes and memory is implemented entirely in software.
- 2 Mutexes for the mailbox interface:
  - 1 mutex for **write** access to the software mailbox
  - 1 mutex for **read** access to the software mailbox



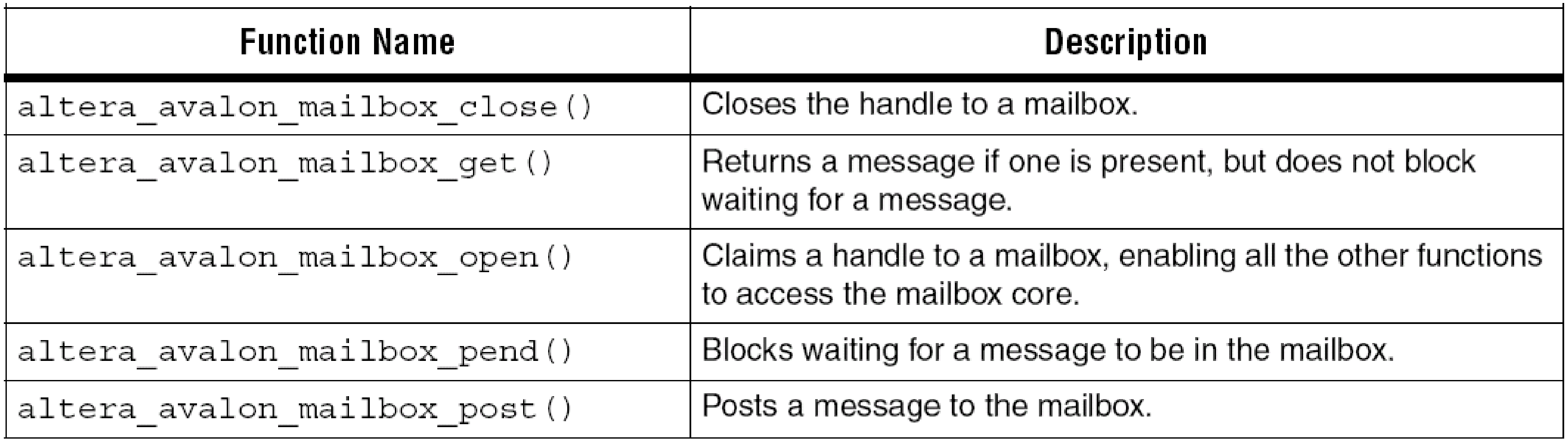
Offset	Register Name	R/W	Bit Description		
			31 … 16	15 … 1	0
0	mutex0	RW	OWNER	VALUE	
1	reset0	RW	–	–	RESET
2	mutex1	RW	OWNER	VALUE	
3	reset1	RW	–	–	RESET

Multiprocessors (Mailbox) deprecated

- A complete mailbox for message passing need:
  - ➢ The mailbox interface
  - ➢ A shared memory
- Both of them need to be available for all of the processors in the systems who need to communicate together

Multiprocessors (Mailbox) deprecated

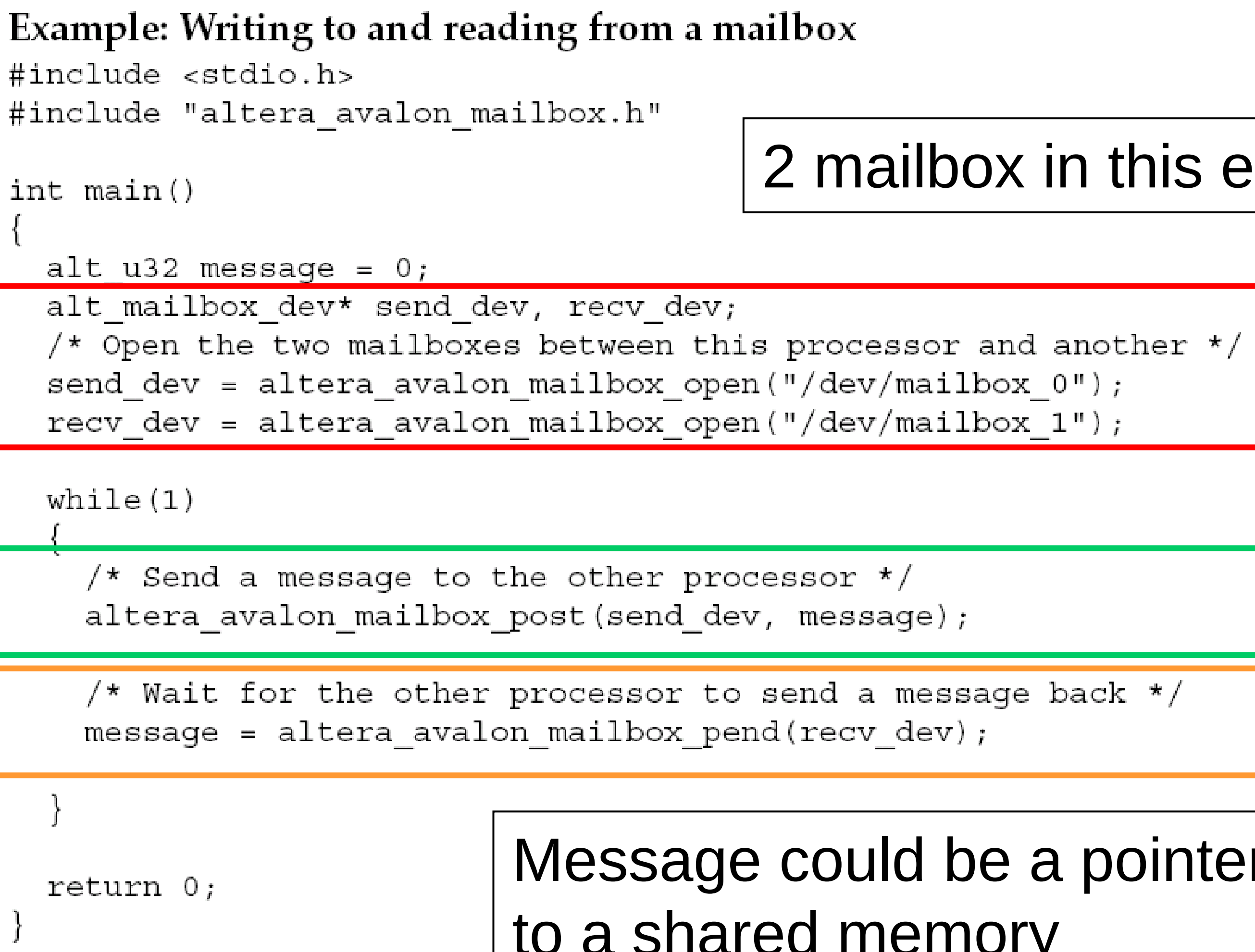
- Some part of function are available to manage and use the mailbox



Function Name	Description
`altera_avalon_mailbox_close()`	Closes the handle to a mailbox.
`altera_avalon_mailbox_get()`	Returns a message if one is present, but does not block waiting for a message.
`altera_avalon_mailbox_open()`	Claims a handle to a mailbox, enabling all the other functions to access the mailbox core.
`altera_avalon_mailbox_pend()`	Blocks waiting for a message to be in the mailbox.
`altera_avalon_mailbox_post()`	Posts a message to the mailbox.

26

Multiprocessors (Mailbox) deprecated



Example: Writing to and reading from a mailbox

```
#include <stdio.h>
#include "altera_avalon_mailbox.h"

int main()
{
  alt_u32 message = 0;
  alt_mailbox_dev* send_dev, recv_dev;
  /* Open the two mailboxes between this processor and another */
  send_dev = altera_avalon_mailbox_open("/dev/mailbox_0");
  recv_dev = altera_avalon_mailbox_open("/dev/mailbox_1");

  while(1)
  {
    /* Send a message to the other processor */
    altera_avalon_mailbox_post(send_dev, message);

    /* Wait for the other processor to send a message back */
    message = altera_avalon_mailbox_pend(recv_dev);

  }

  return 0;
}
```

2 mailbox in this example

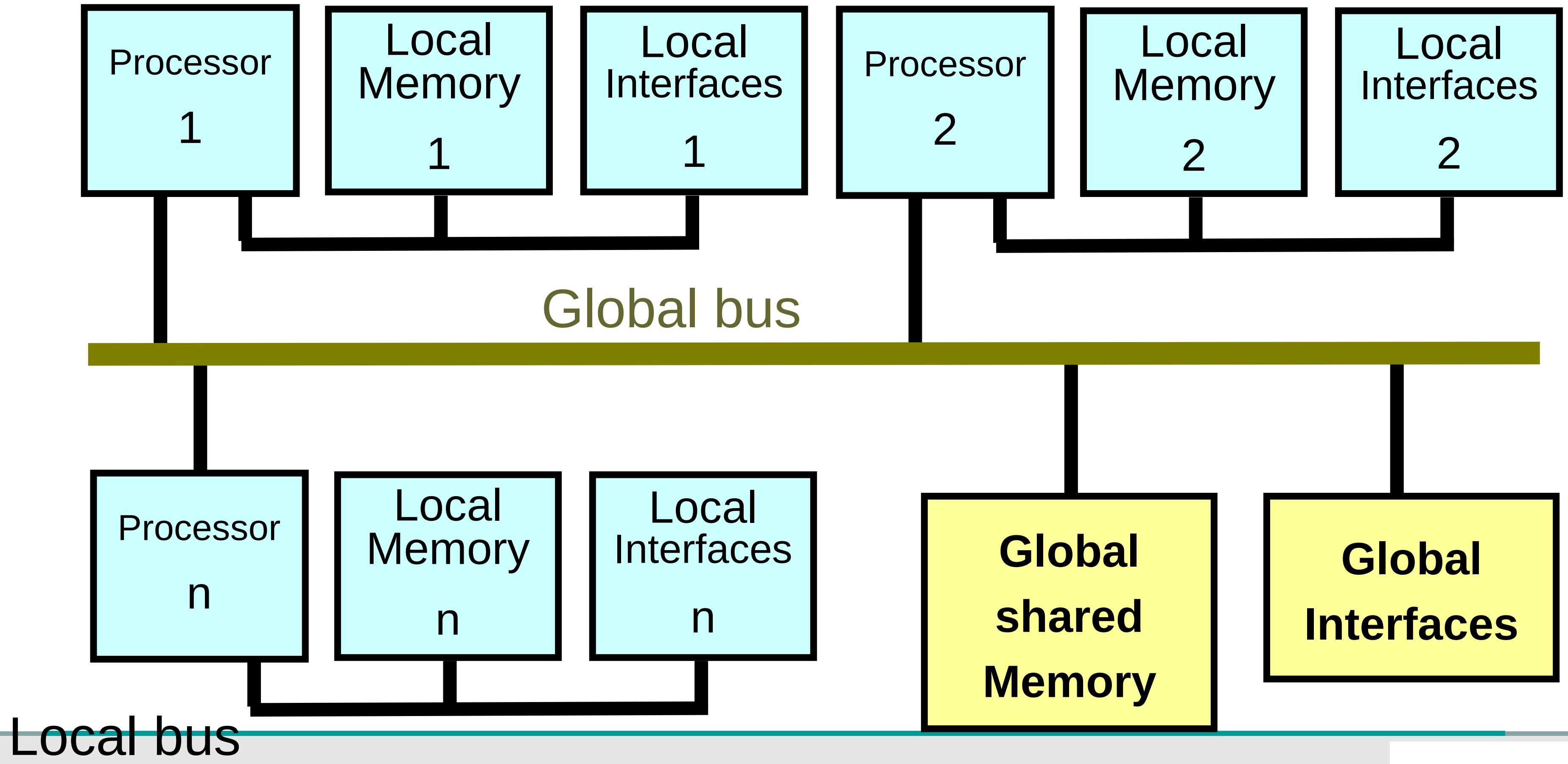
Message could be a pointer to a shared memory

Multiprocessor cache memories

- In multiprocessor with data cache memories a big problem is cache **coherency** for shared common memories
- If a write access is done by a master, bus snooping by other caches need to invalidate them if necessary
- Or shared memory access could by-pass the cache memory (i.e. bit 31=1 on Avalon bus)
- Or flush of local data cache needs to be done before shared memory access

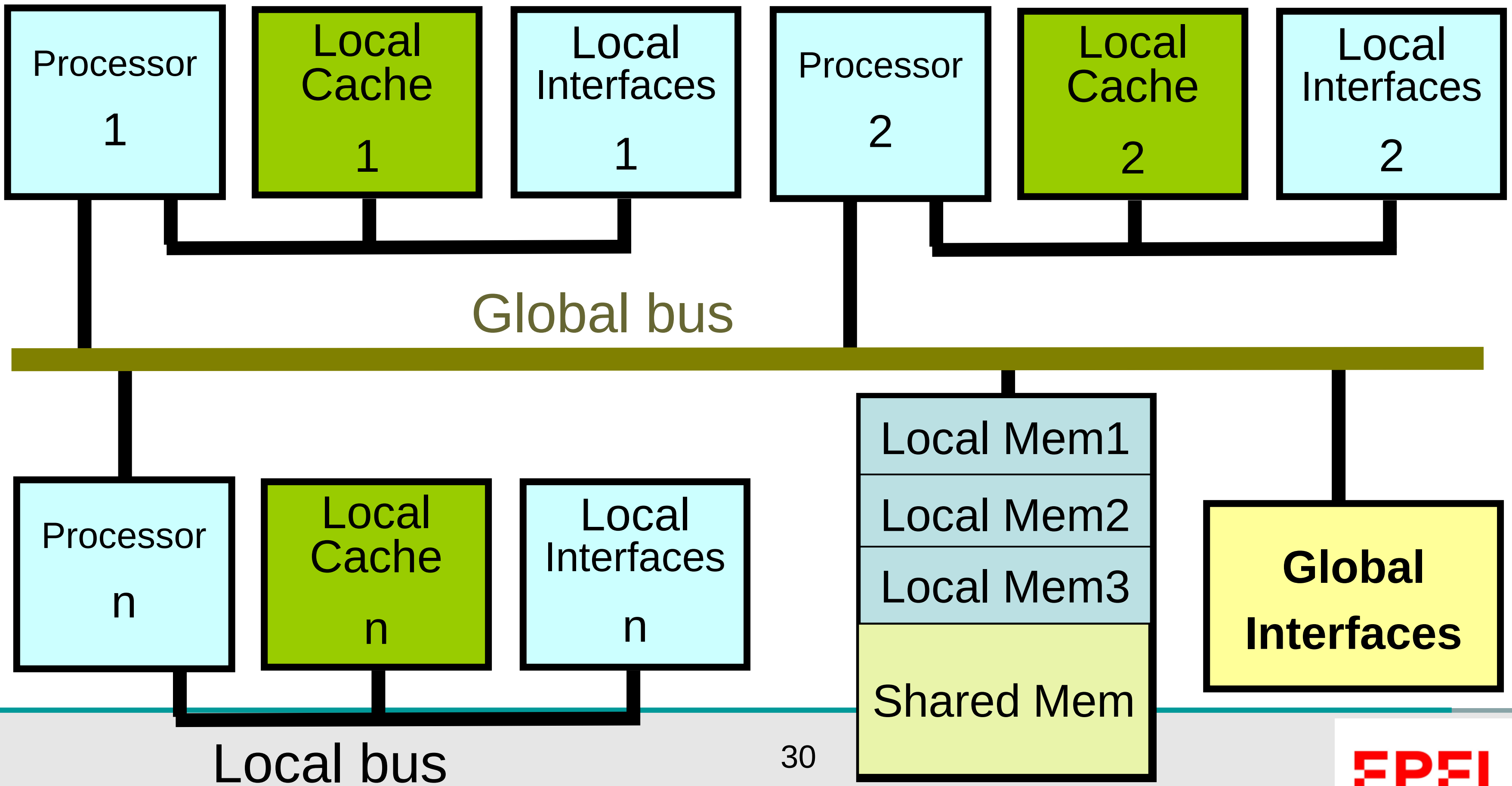
Multi Masters architecture

Local memory for each processor
Global memory for shared data/instr



Multi Masters architecture

"***Local memories***" can be in separate area on the global physical memory
Processors could have local cache memories (instr/data)

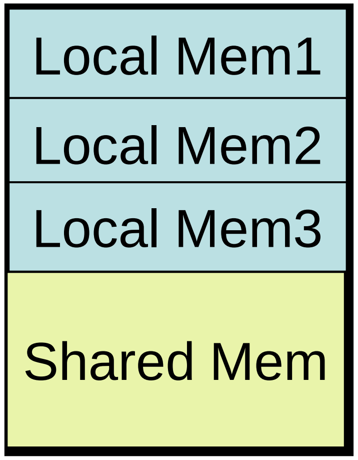


Multiprocessor instruction memories

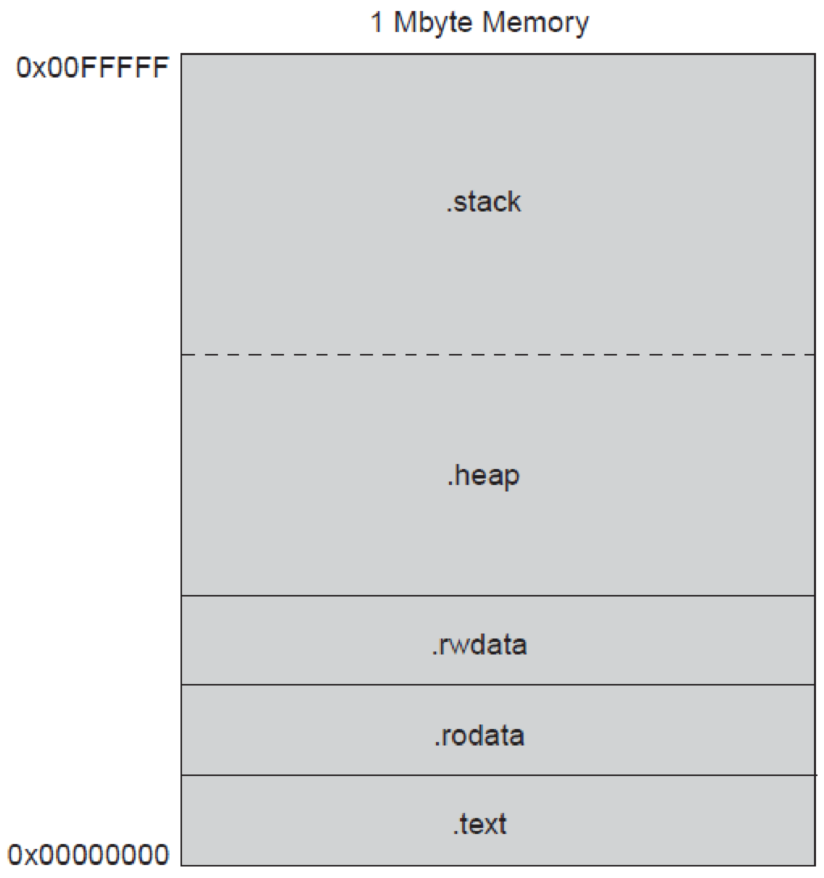
- Instruction cache memories need to be **flushed** after a new modification of the code (code loading).
- The problem is to be able to signal that fact to the corresponding processor ! i.e. by interruption.

Multiprocessors on Avalon, memory mapping

- Each processor can have a reserved local access to the main common memory.
- NIOS IDE use the Exception address specified in SOPC from the NIOS description



Memory Map on a typical processor (ex. NIOSII)

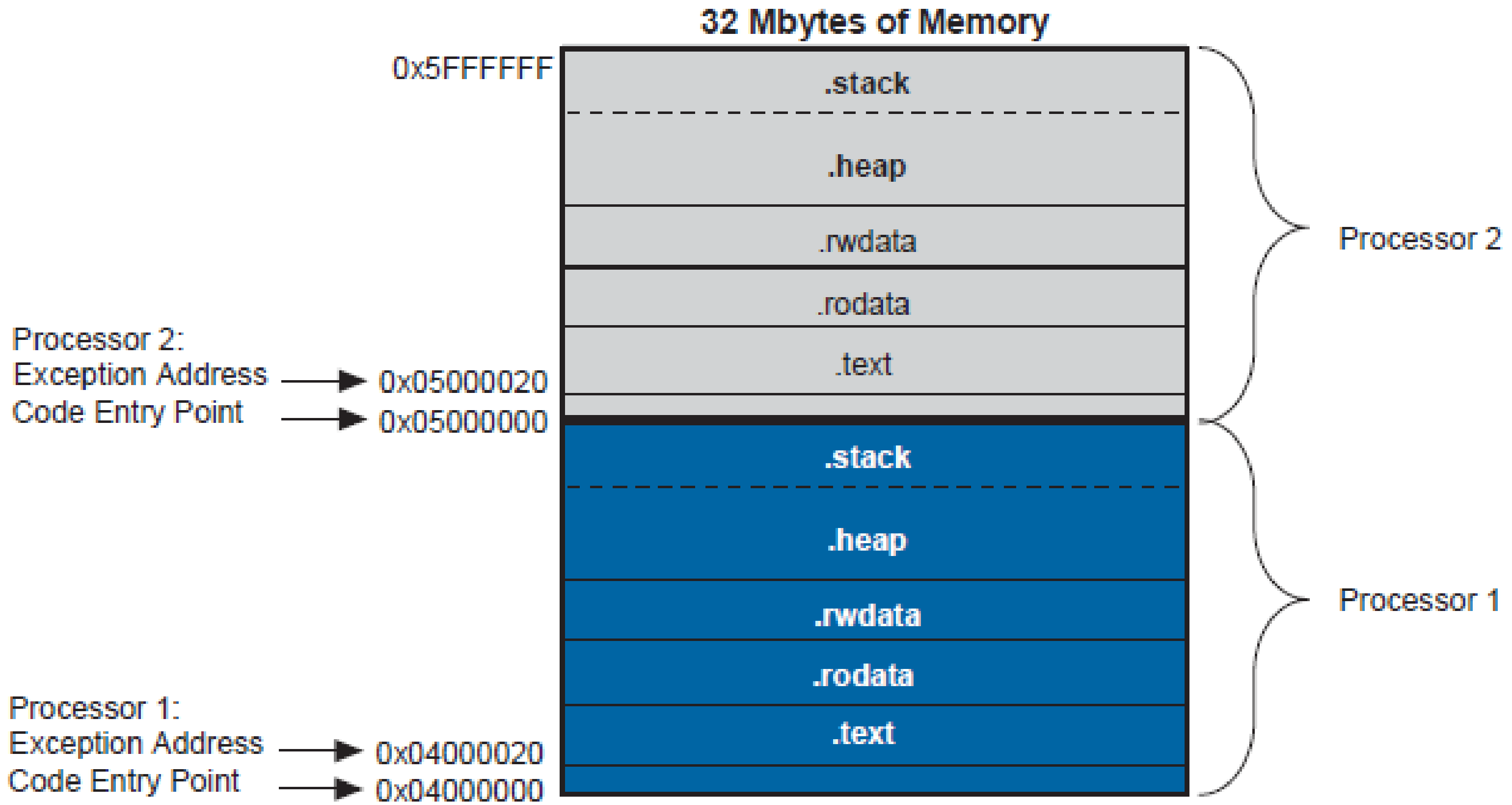


- Areas used by the linker:
- **.stack**— where function-call parameters and other temporary data is stored
- **.heap**— where dynamically allocated memory is located
- **.rwdata**— where read-write variables and pointers are stored
- **.rodata**— any read-only data used in the execution of the code
- **.text**— the actual executable code

- The memory used for each area is specified in NIOS IDE through System Library Settings

Mapping for two processors

- Memory Map Specification for 2 processors



Memory configuration rules

- Be careful as NO barrier are available between the two memory maps
- Left 0x20 between Reset and Exception addresses: it corresponds to 1 instruction cache line
- The end of a processor area is the starting address of the next area
- For common memory, allocate it on one processor area and pass the pointer by message passing or use an separate ***internal SRAM*** available to all processors

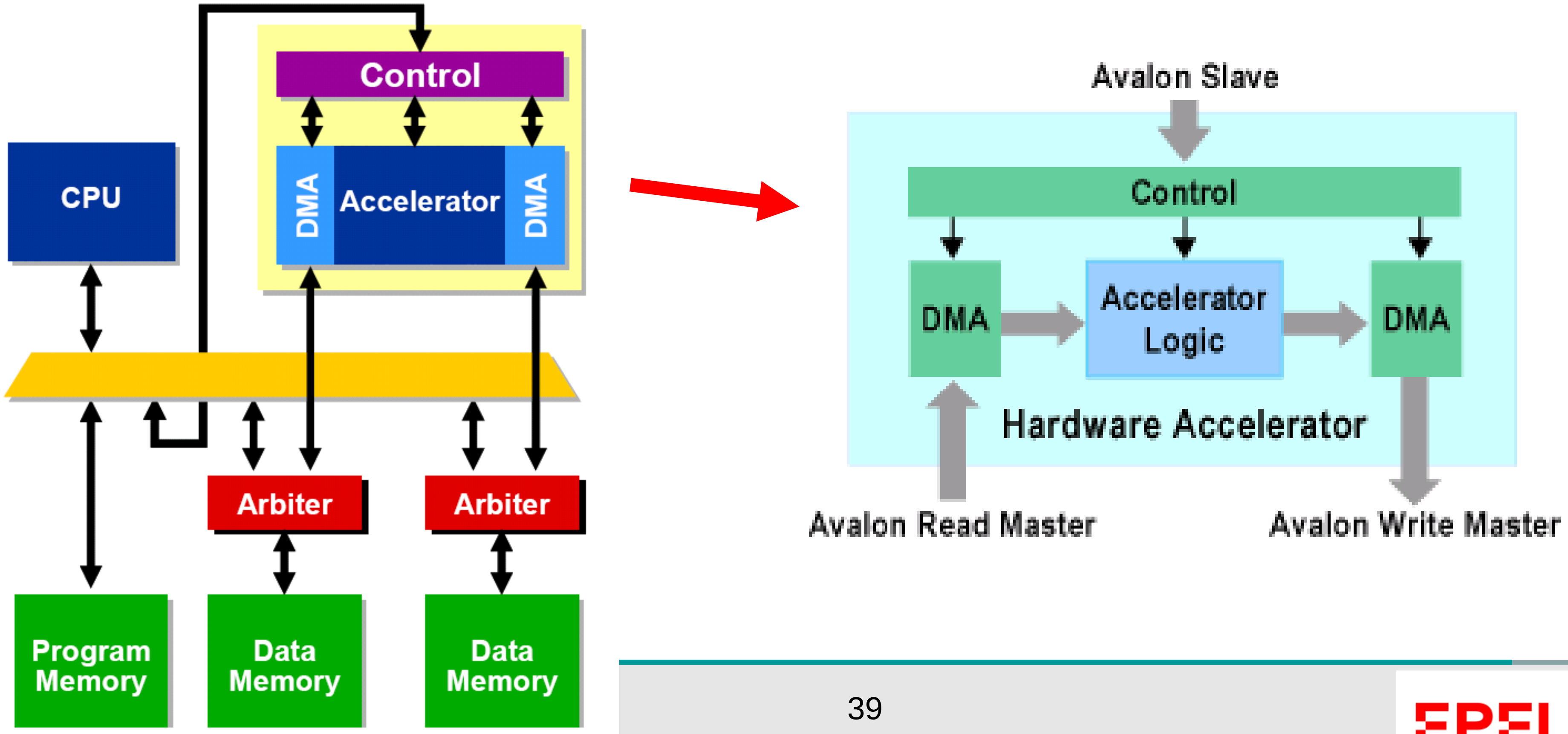
Debugging with NIOS IDE

- With NIOS IDE, multiprocessors can be debugged as n single processors through processor ID specification for each NIOS program.
- A ***collection*** can be specified to group n processor and run their program automatically, !! They are NOT synchronized to start at the same time, it's the user responsibility to do that.

MULTI MASTER DMA ACCELERATOR

Multi master DMA accelerator

- An accelerator is based on a DMA unit:
- Direct Memory Access unit capable of reading/Writing in memory as an Avalon Master (altera sopc system)

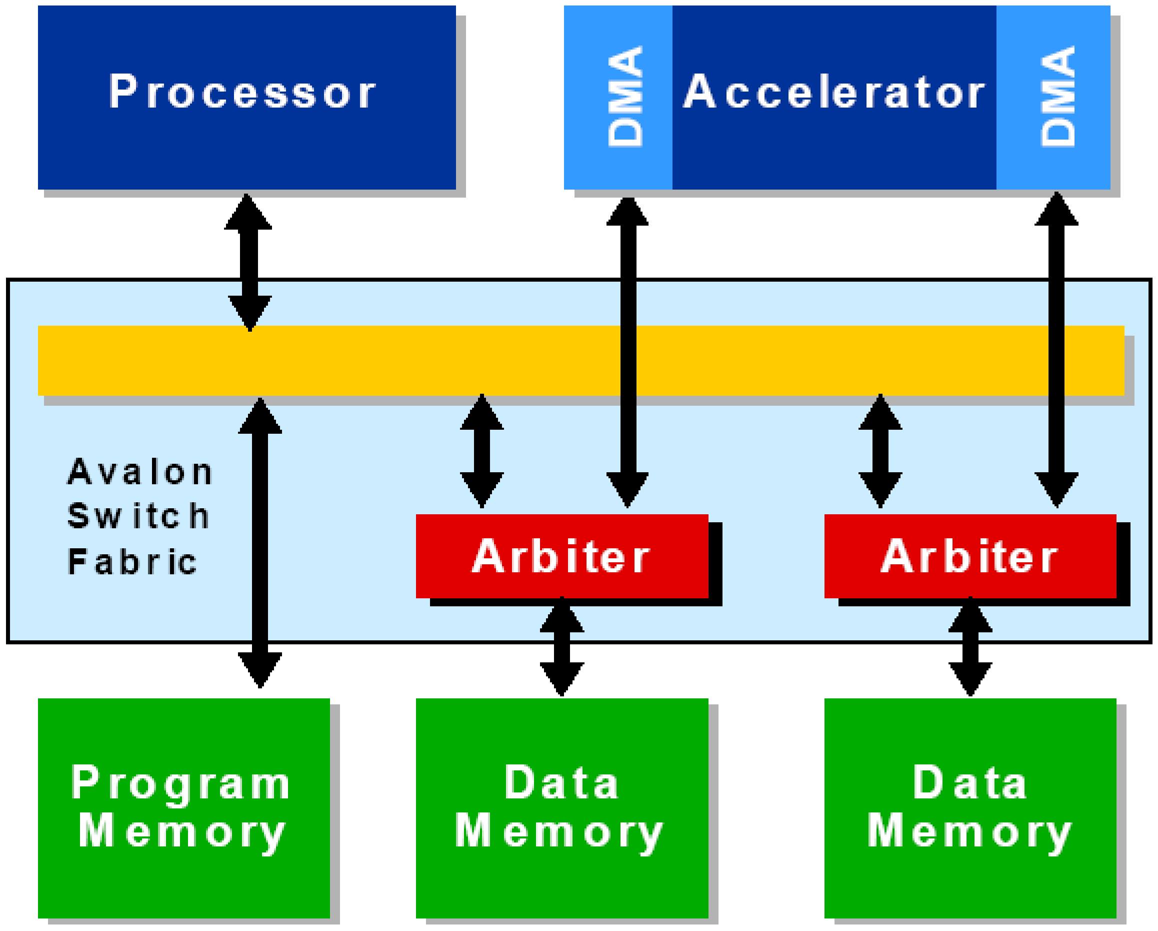


EPFL

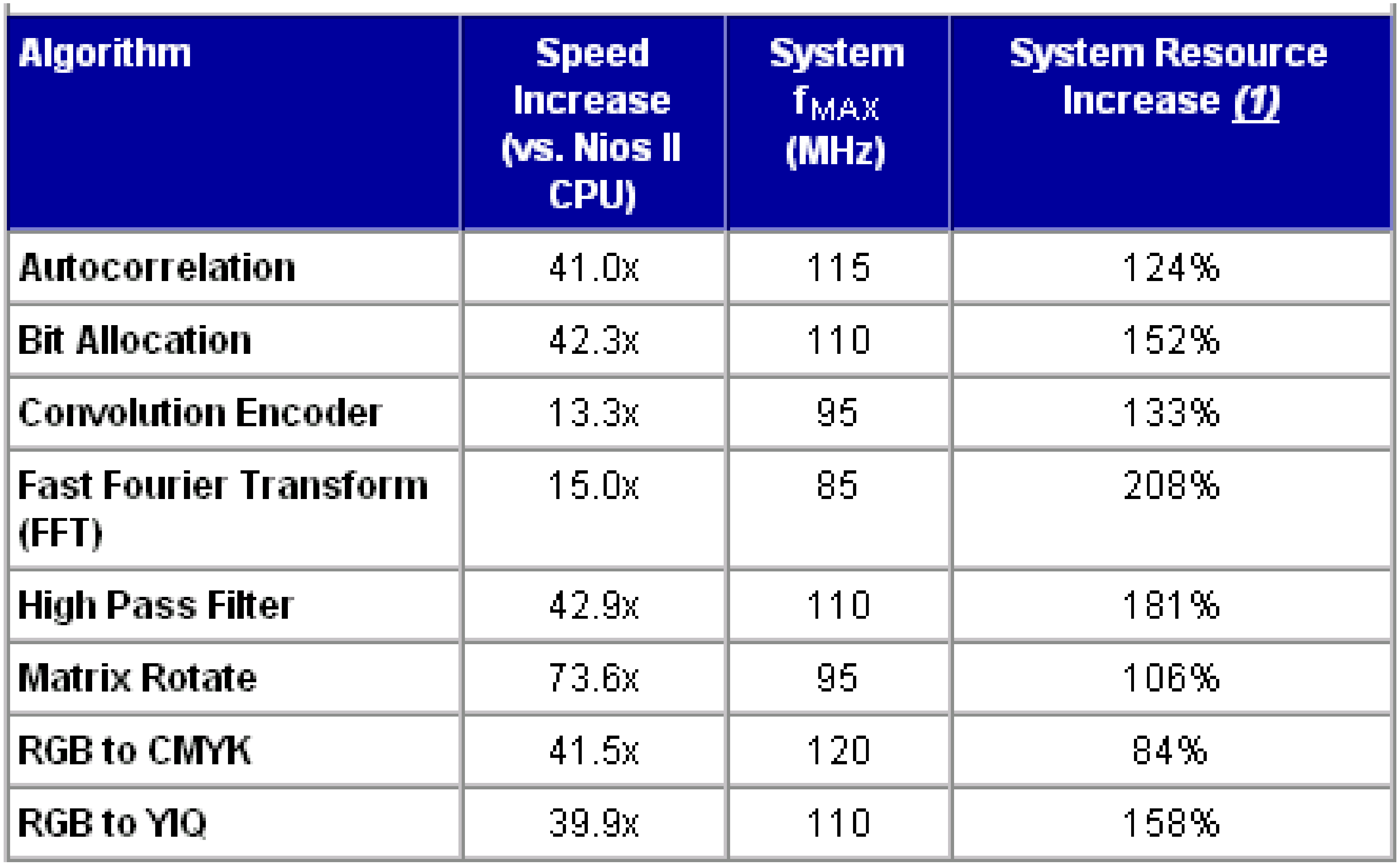
NIOS II Processor, hardware accelerator

- To assist the processor efficiency in some tasks, a specialized **accelerator** can be provide.
- It has the capacity of reading and writing in memory by itself.
- It can make (complex) operations between reading and writing,
- The processor has to initialize the source/destination addresses, the length and some parameters depending on the accelerator

NIOS II Processor, hardware accelerator



NIOS II Processor, hardware accelerator, examples (Stratix II, from Altera)



Algorithm	Speed Increase (vs. Nios II CPU)	System f_{MAX} (MHz)	System Resource Increase (1)
Autocorrelation	41.0x	115	124%
Bit Allocation	42.3x	110	152%
Convolution Encoder	13.3x	95	133%
Fast Fourier Transform (FFT)	15.0x	85	208%
High Pass Filter	42.9x	110	181%
Matrix Rotate	73.6x	95	106%
RGB to CMYK	41.5x	120	84%
RGB to YIQ	39.9x	110	158%

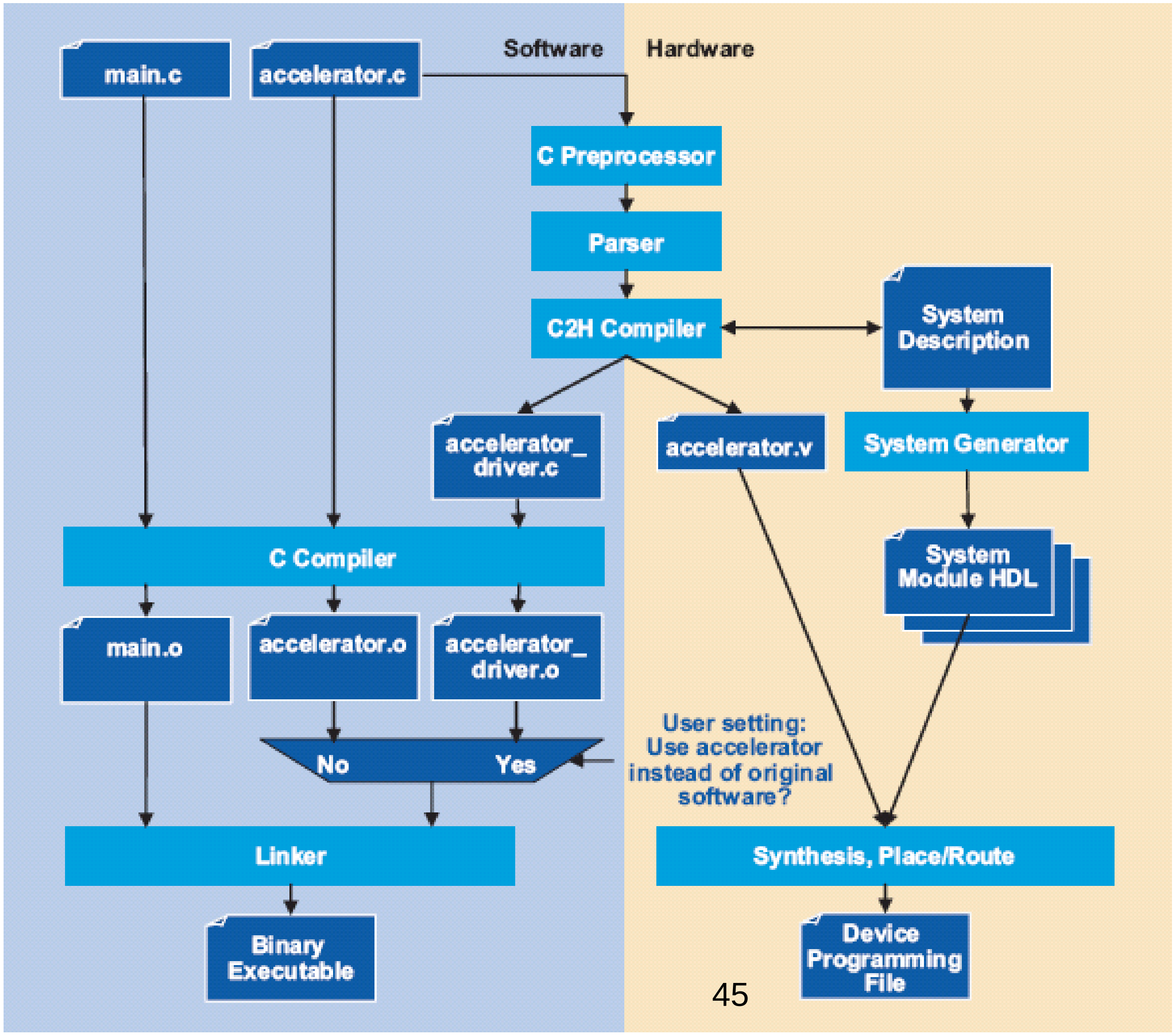
NIOS II Processor, hardware accelerator

- The accelerator can be design in HDL language as **VHDL** or **Verilog**
- It can be extracted directly from **C code** an translated in HDL language with initializing and API functions in C automatically
- It was the **C2H** compiler tool (deprecated)

NIOS II Processor, hardware accelerator

- More “universal” language **OpenCL** C code:
  - ➢Kernel modules translated:
    - ➢for multiprocessor
    - ➢for GPU
    - ➢for FPGA

NIOS II Processor, hardware accelerator design flow



Profiling

- To determine which part of code is useful to accelerate, code profiling is the best tool.
- Code with :
  - ➢Loop
  - ➢specific algorithm between source and destination
  - ➢a large amount of data
- is a good candidate to accelerate

Profiling, GNU Profiler

- Minimal source code changes are required to take measurements for analysis with the GNU profiler. The only changes needed are as follows:

1. Add the profiler library via a checkbox in the Nios II IDE.
2. Change the main() function to call exit().
3. Rebuild the project.

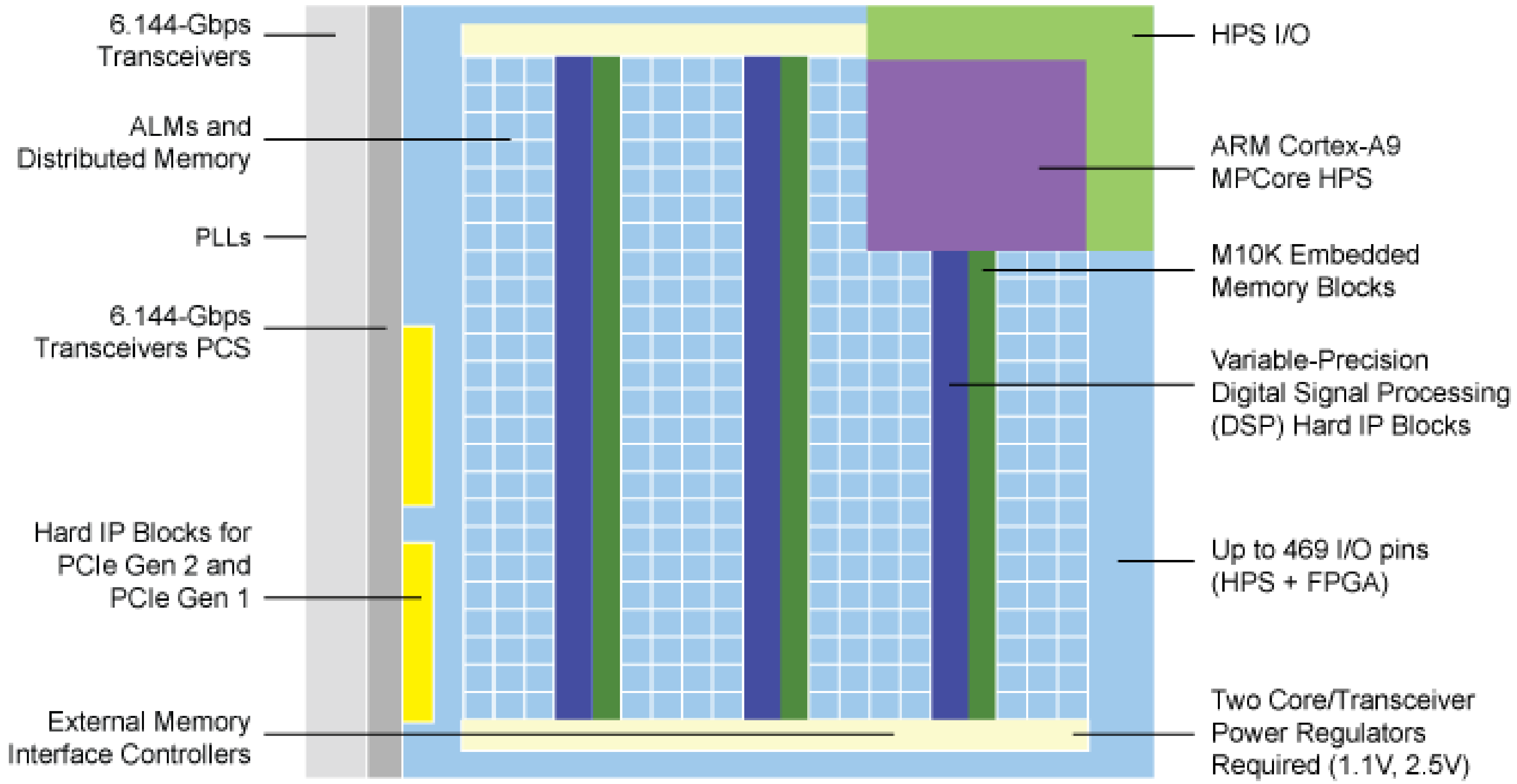
Profiling, GNU Profiler Advantage

- The major advantage to measuring with the profiler is that it provides an **overview of the entire system**. Although there is some overhead, it is distributed evenly throughout the system.
- The functions that are identified as consuming the most CPU time will still consume the most CPU time when run at full speed without profiler instrumentation.

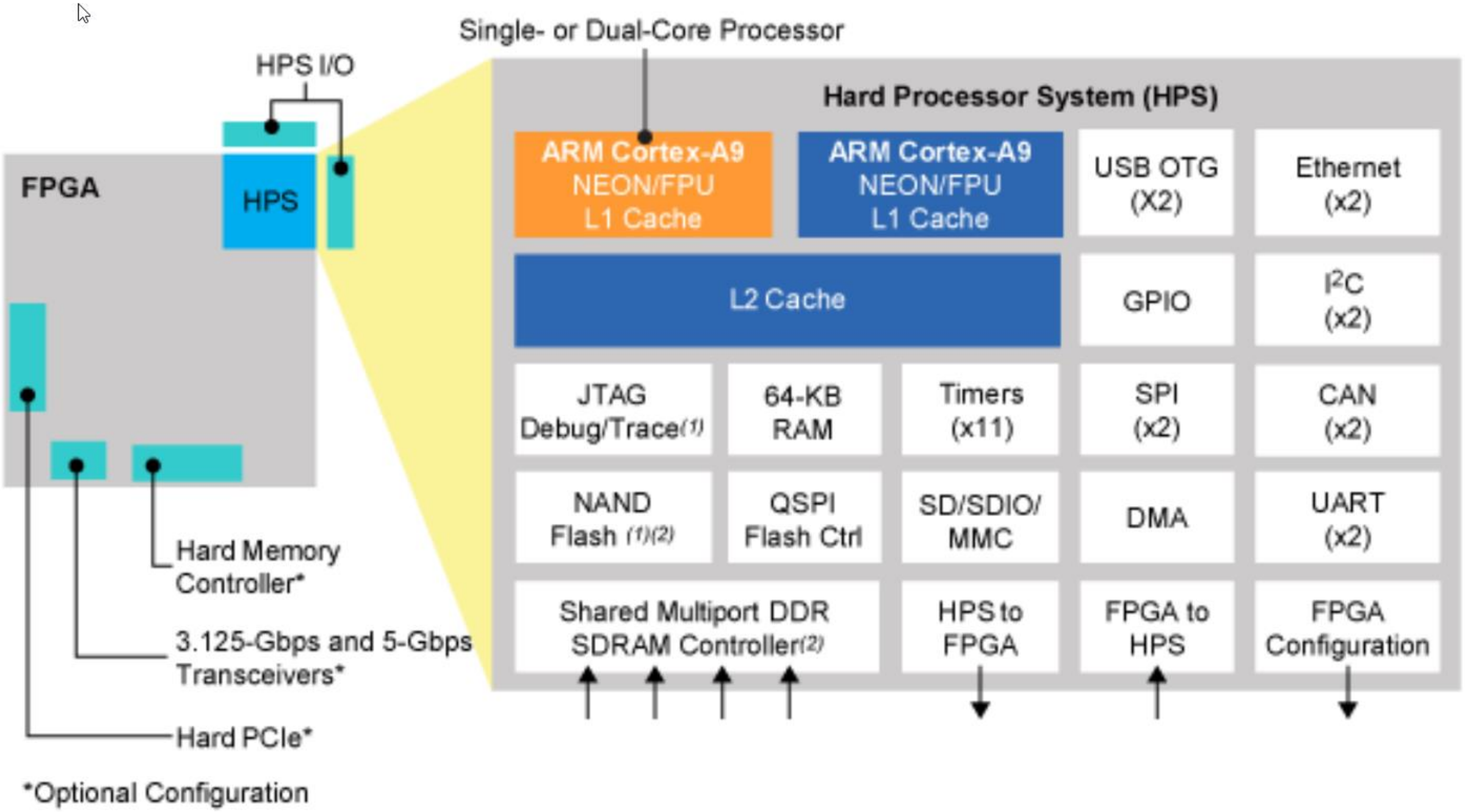
Profiling, GNU Profiler Drawback

- Adding instructions to each function call for use with the profiler **affects the code’s behavior**.
- Each function is slightly larger. Each function calls another function to collect profiling information.

SOC + FPGA (ex.CycloneV)


6.144-Gbps Transceivers
ALMs and Distributed Memory
PLLs
6.144-Gbps Transceivers PCS
Hard IP Blocks for PCIe Gen 2 and PCIe Gen 1
External Memory Interface Controllers
HPS I/O
ARM Cortex-A9 MPCore HPS
M10K Embedded Memory Blocks
Variable-Precision Digital Signal Processing (DSP) Hard IP Blocks
Up to 469 I/O pins (HPS + FPGA)
Two Core/Transceiver Power Regulators Required (1.1V, 2.5V)

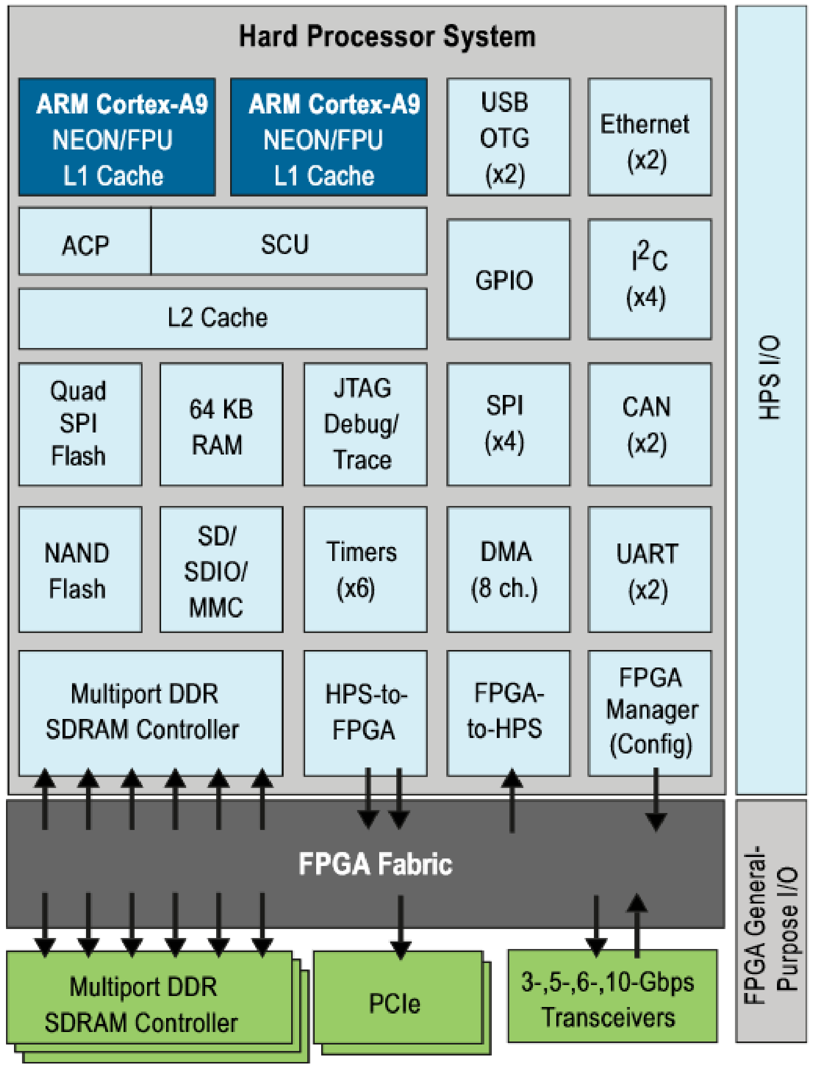
SOC + FPGA (ex.CycloneV)



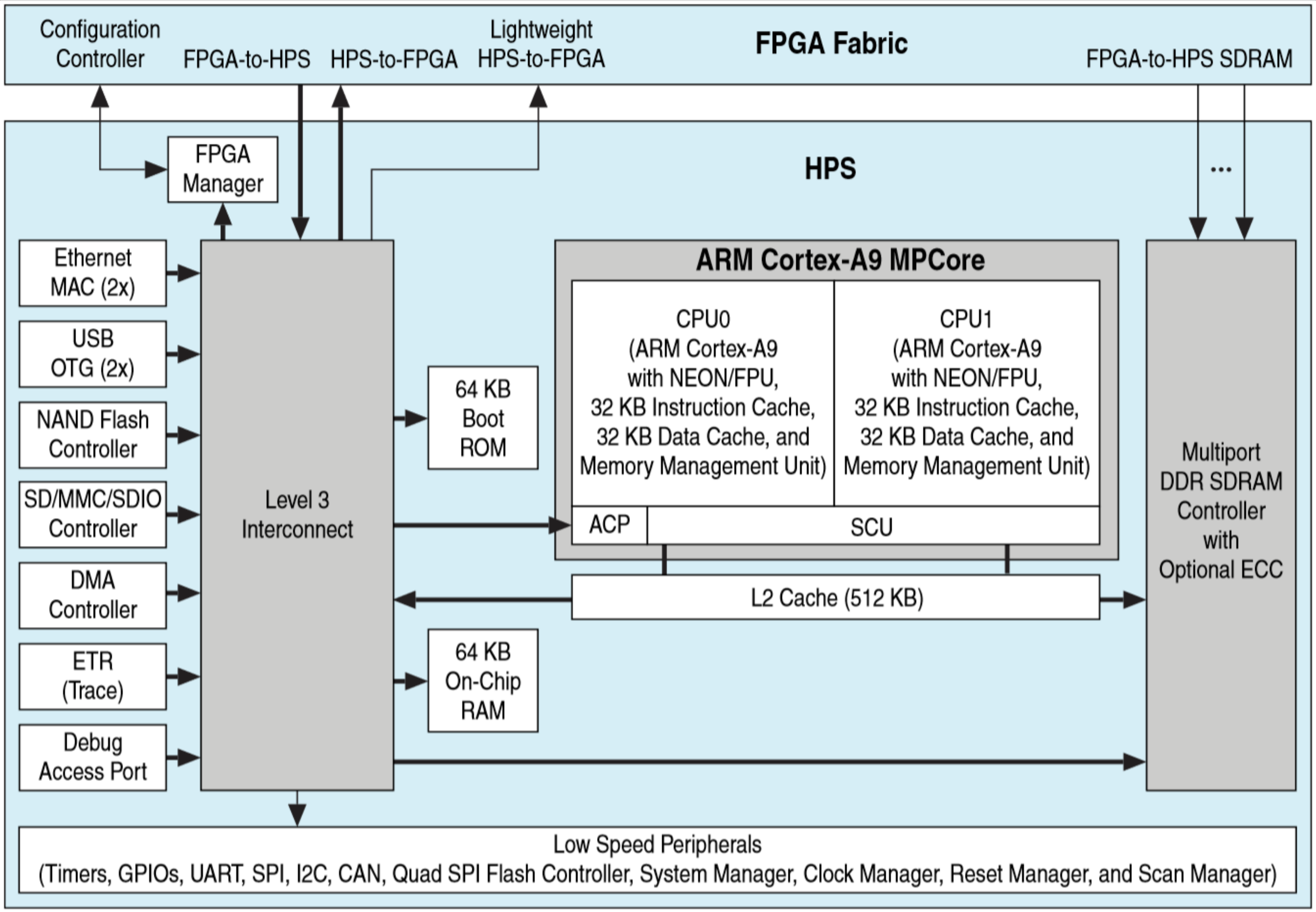
SOC + FPGA (ex.CycloneV)

SCU: Snoop Control Unit

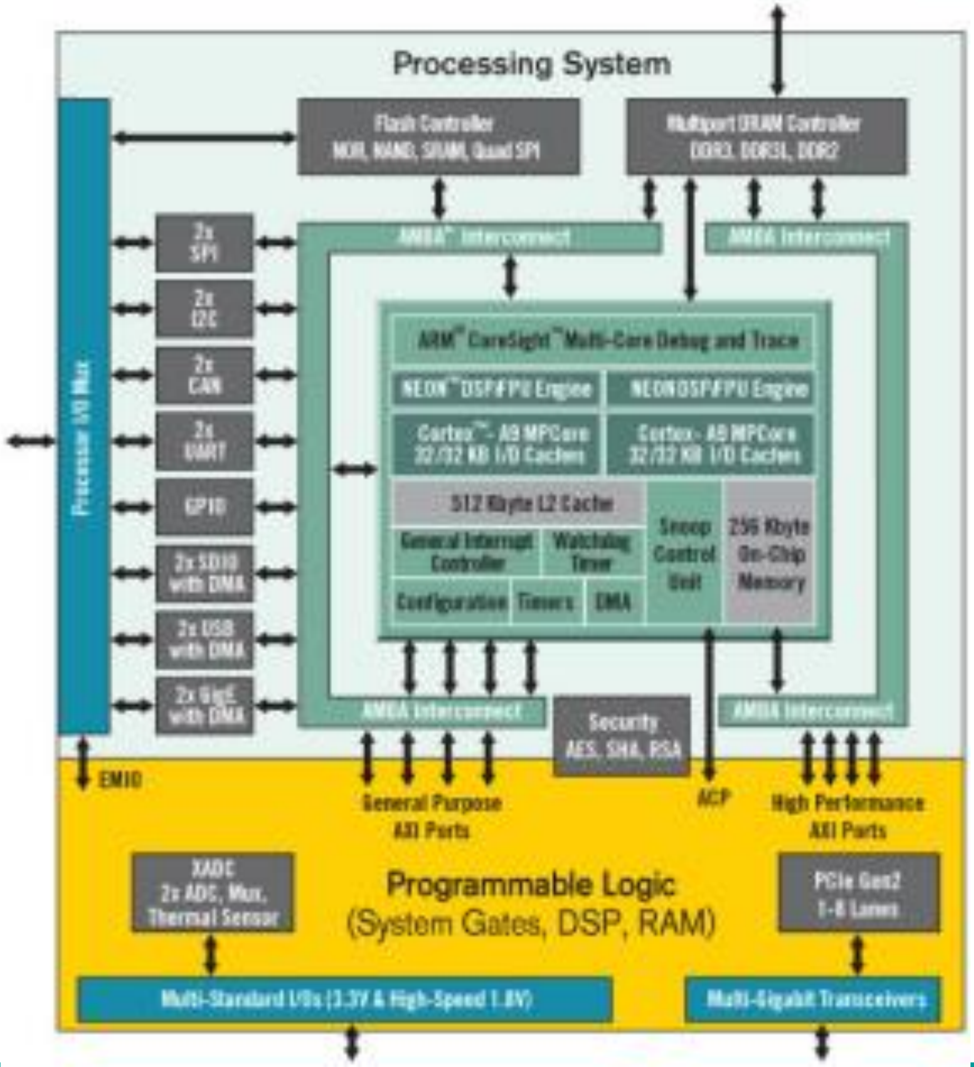
ACP: Accelerator Coherency Port



SOC + FPGA (ex.CycloneV)


Configuration Controller
FPGA-to-HPS
HPS-to-FPGA
Lightweight HPS-to-FPGA
FPGA Fabric
FPGA-to-HPS SDRAM
FPGA Manager
HPS
...
Ethernet MAC (2x)
USB OTG (2x)
NAND Flash Controller
SD/MMC/SDIO Controller
DMA Controller
ETR (Trace)
Debug Access Port
Level 3 Interconnect
64 KB Boot ROM
64 KB On-Chip RAM
ARM Cortex-A9 MPCore
CPU0 (ARM Cortex-A9 with NEON/FPU, 32 KB Instruction Cache, 32 KB Data Cache, and Memory Management Unit)
CPU1 (ARM Cortex-A9 with NEON/FPU, 32 KB Instruction Cache, 32 KB Data Cache, and Memory Management Unit)
ACP
SCU
L2 Cache (512 KB)
Multiport DDR SDRAM Controller with Optional ECC
Low Speed Peripherals
(Timers, GPIOs, UART, SPI, I2C, CAN, Quad SPI Flash Controller, System Manager, Clock Manager, Reset Manager, and Scan Manager)

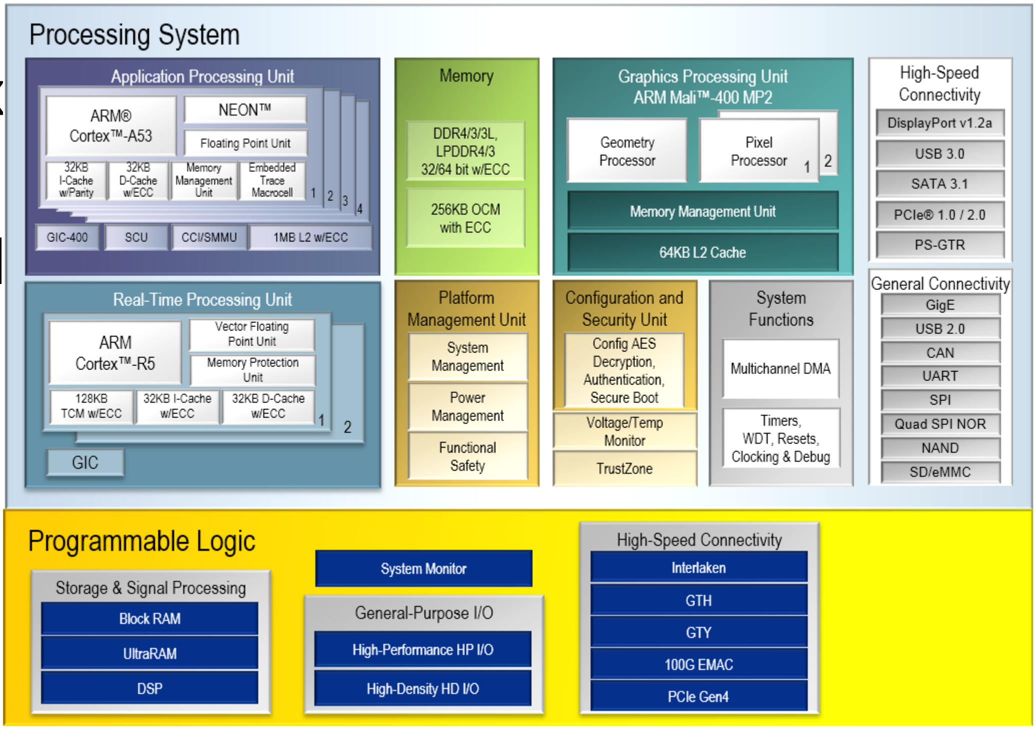
SOC + FPGA (ex.Zynq Xilinx)



Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA Interconnect
AMBA Interconnect
ARM CoreSight Multi-Core Debug and Trace
NEON DSP/FPU Engine
NEON DSP/FPU Engine
Cortex - A9 MPCore 32/32 KB I/D Caches
Cortex - A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

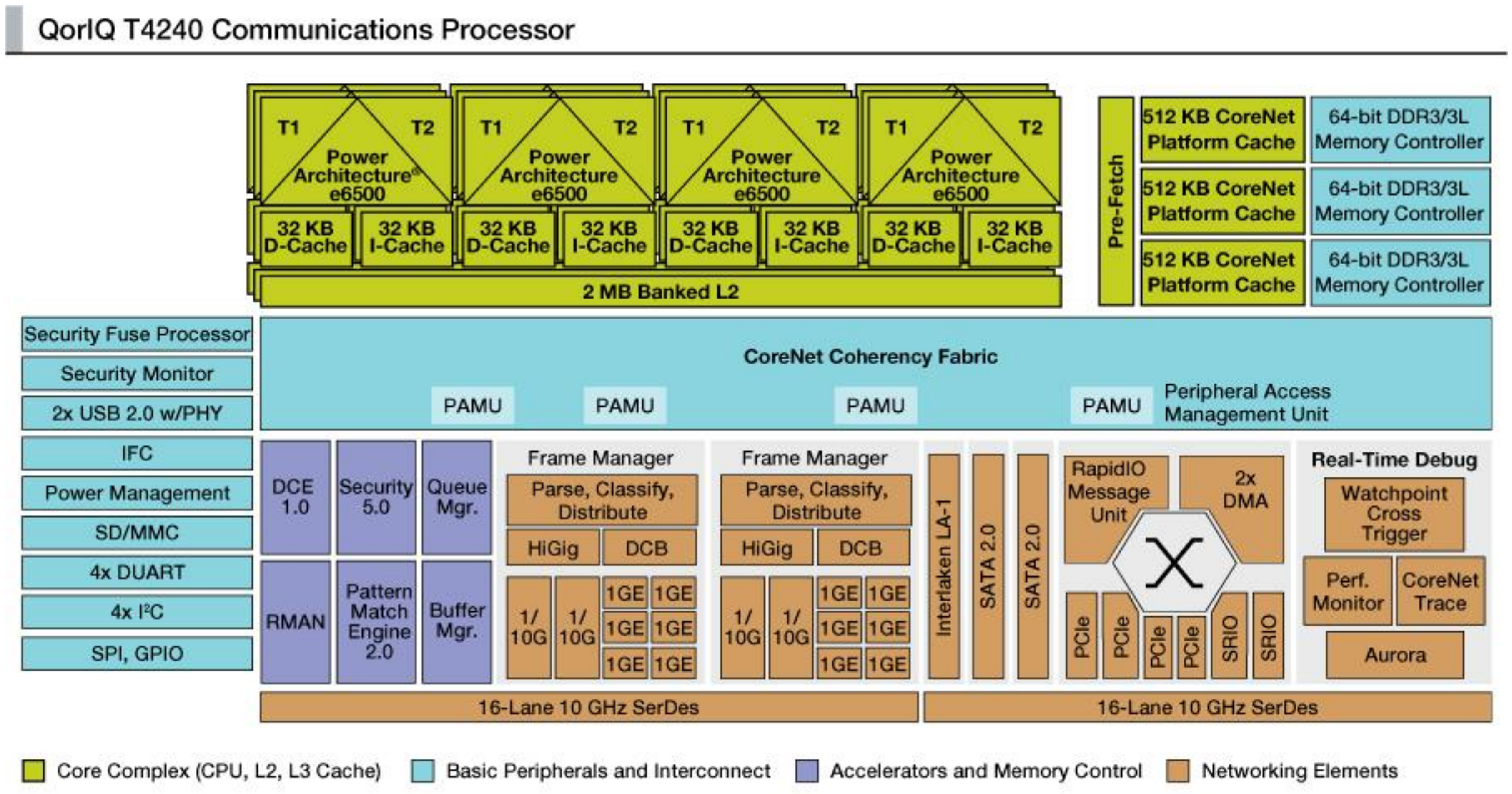
Ultra-Scale Psoc Architecture (ex: Zynq UltraScale+ EG)

- quad-core ARM® Cortex A53 up to 1.5GHz.
- dual-core Cortex-R5 real time processors,
- Mali-400 MP2 graphics processing unit,
- and 16nm FinFET+ programmable logic

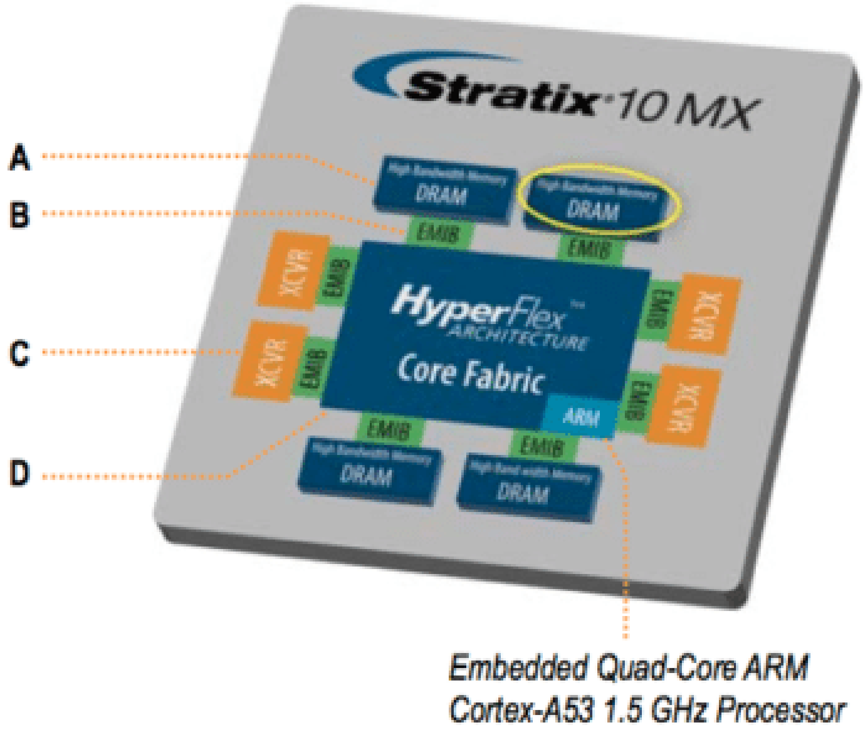


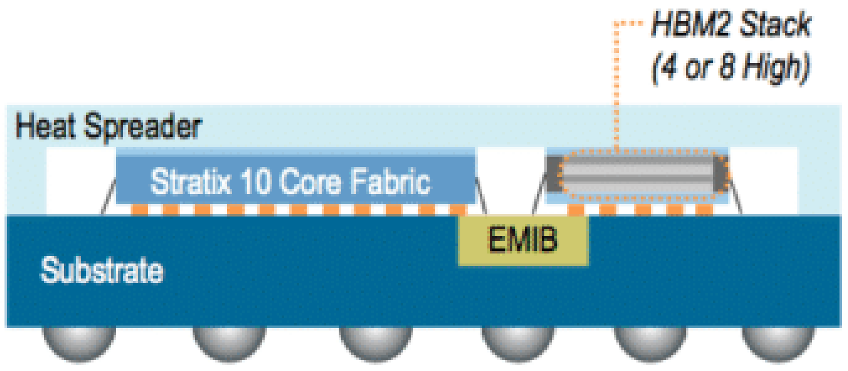
https://www.xilinx.com/content/dam/xilinx/imgs/products/zynq/zynq-eg-block.PNG

Multiprocessor, ex. T4240 Freescale QorIQ serie

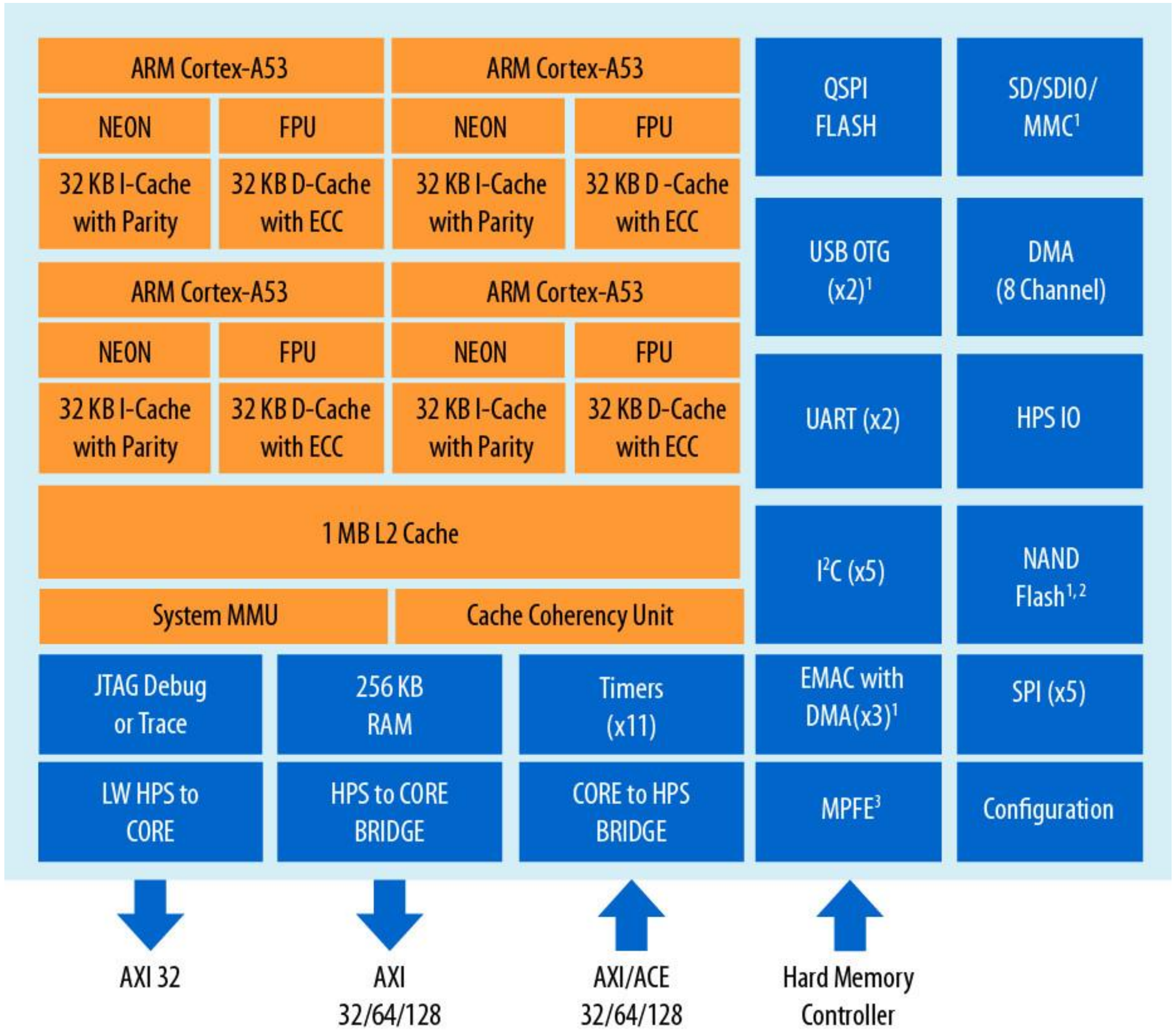


Stratix-10 Global view


Stratix 10 MX
A
B
C
D
High Bandwidth Memory DRAM
EMIB
XCVR
HyperFlex ARCHITECTURE
Core Fabric
ARM
Embedded Quad-Core ARM Cortex-A53 1.5 GHz Processor

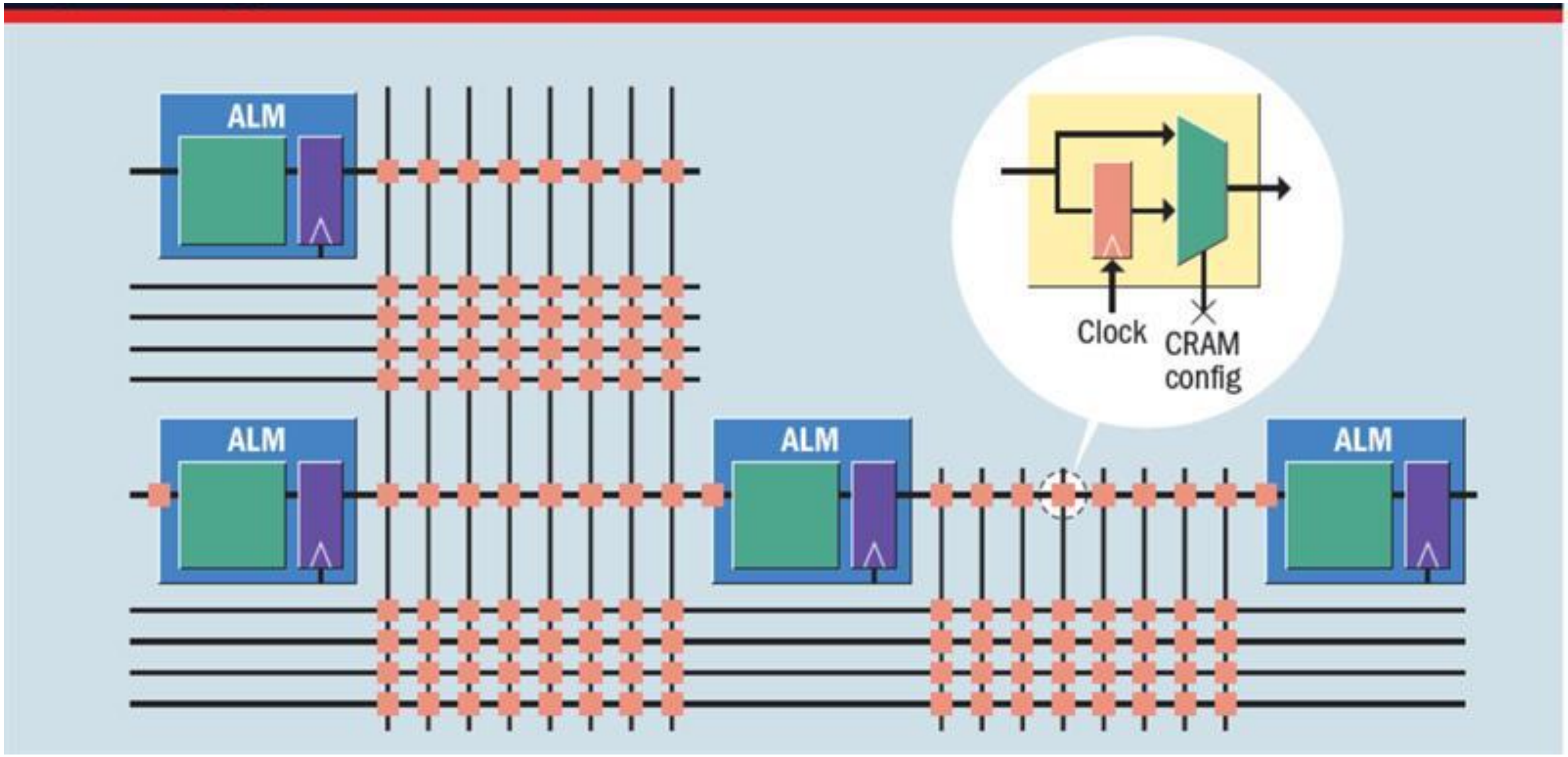


Startix-10, SoC architecture with FPGA



4x ARM-A53 64 bits
AXI 32/64/128 bits

Stratix 10: HyperFlex Architecture
Synchronisation on routing signals


ALM
ALM
ALM
ALM
Clock
CRAM config