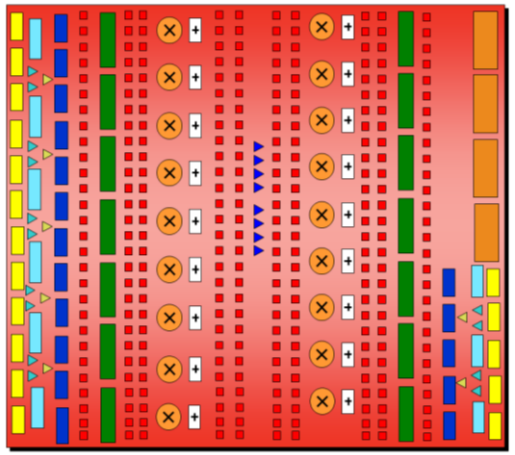
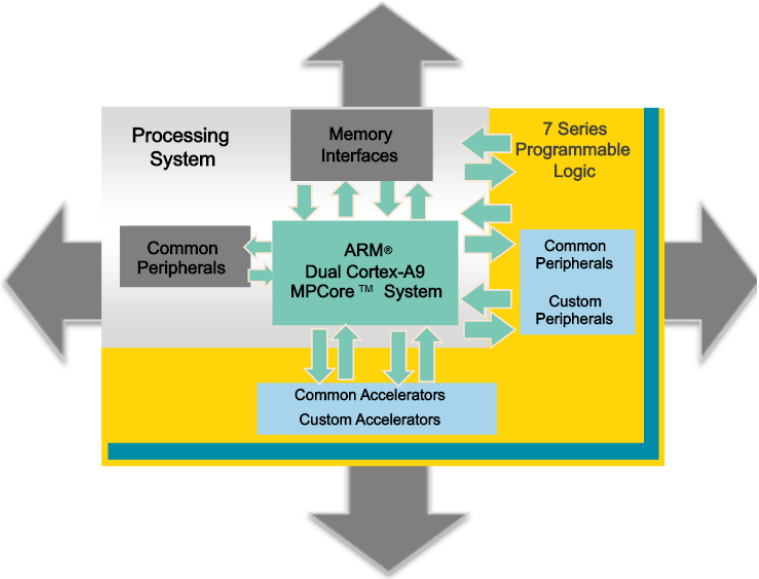
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

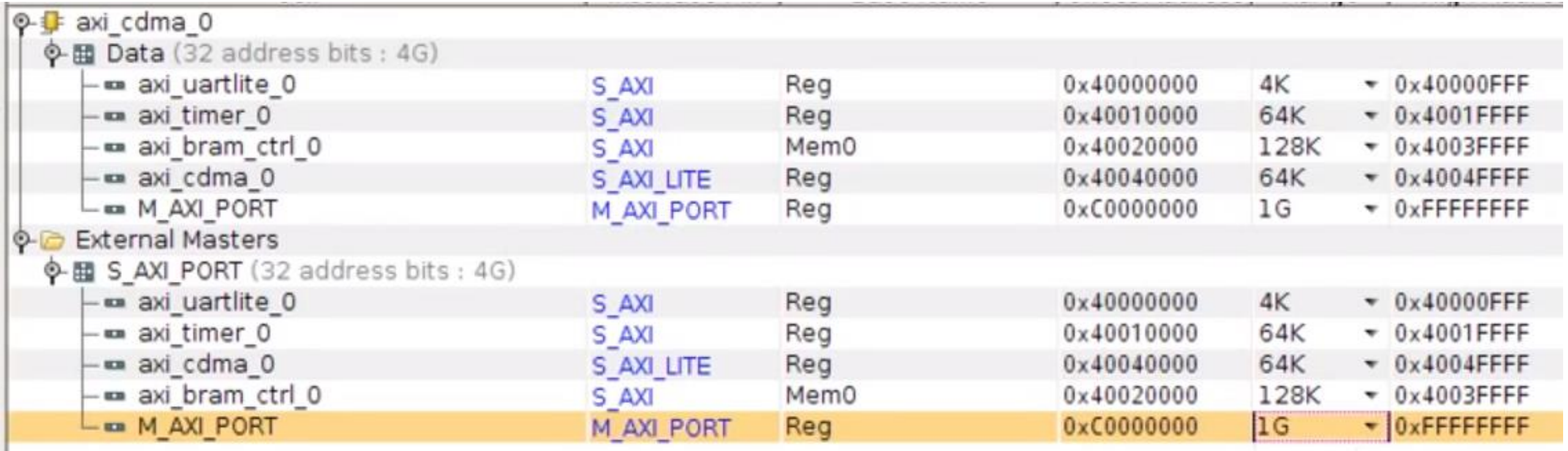
A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design





1KB = 1024 Byte	Kilo	10 address bits
1 MB = 1024 KB	Mega	20 address bits
1 GB = 1024 MB	Giga	30 address bits



Address Allocations

- Consider Zynq SoC with memory mapped IO where the start or base address for any accelerator in PL must be equal to or higher than 0x2000_0010. The example design consists of three accelerators and their memory requirements in Bytes are given in the table below. Assign appropriate starting or base address to each accelerator. Care must be taken to avoid memory wastage.

Accelerator Name	Memory Requirement
AXI GPIO1	4K
AXI GPIO2	512 M
AXI BRAM	256K

- Consider the embedded system deployed for office security purpose.
- After certain interval, system needs to upload all sensor data on the cloud.
- Suppose that the processor is busy doing such transfer and an emergency event occurs. In such cases, how should processor react?

Interrupts

- An interrupt is a signal which is generated to indicate to the processor that its attention is required.
- Interrupts are generated as a specific response to events.
- Interrupts can be generated by hardware processing units and peripherals, and also within the processor itself.
- **Examples:** Internally generated (divide by zero, illegal instruction), software interrupts or exceptions. Externally generated (Arrival of data, triggering event), hardware interrupt.
- Generated asynchronously but read (polling) synchronously

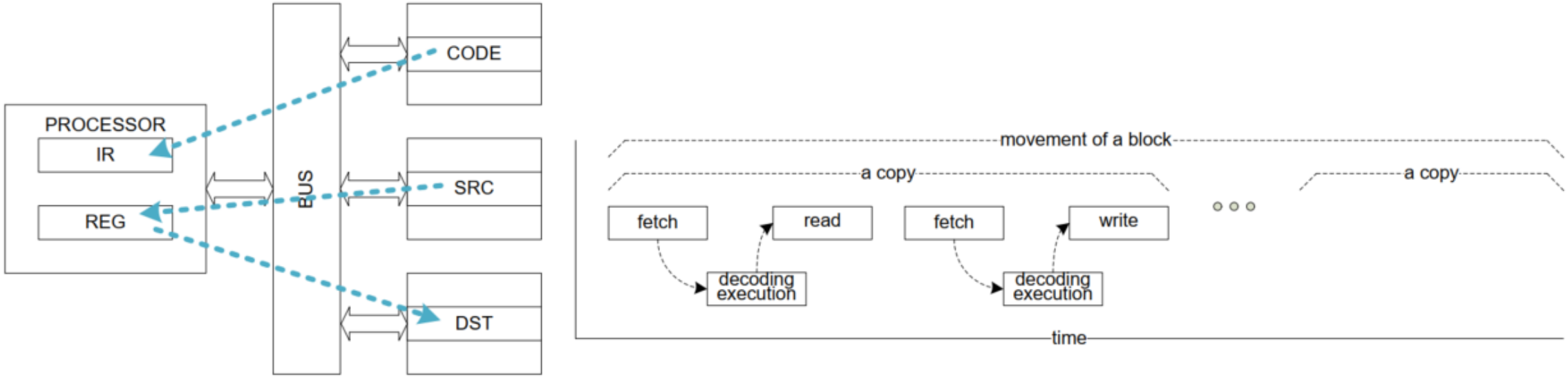
Interrupts

- Interrupts cause a change in program flow.
- First, normal program flow is halted at the end of the current instruction.
- Next, typically a determination is made as to which specific code executes next.
- Then, a special subroutine known as an interrupt handler is called.
- When the interrupt handler completes, normal program flow resumes.

Interrupts

- **Maskable Interrupts (IRQs)** — The trigger event of a masked interrupt is not always important. It is the task of the programmer to decide whether the event should cause the program to jump to the requested execution or not.
- **Non-Maskable Interrupts (NMIs)** — These are interrupts which should never be ignored, and are therefore deemed much more important than maskable interrupts. Events which require NMIs include power-on, external reset (from a physical button) and serious device faults.
- **Inter-Processor Interrupts (IPIs)** — In multiple processor systems, one processor may need to interrupt the operation of another processor. In this situation, an IPI will be generated.
- How to configure interrupts in Zynq SoC: To be discussed later or AELD.

- Consider the embedded system deployed for office security purpose.
- After certain interval, system needs to upload all sensor data on the cloud/memory.
- Usually, memory read and write operations are slower (takes many clock cycles).

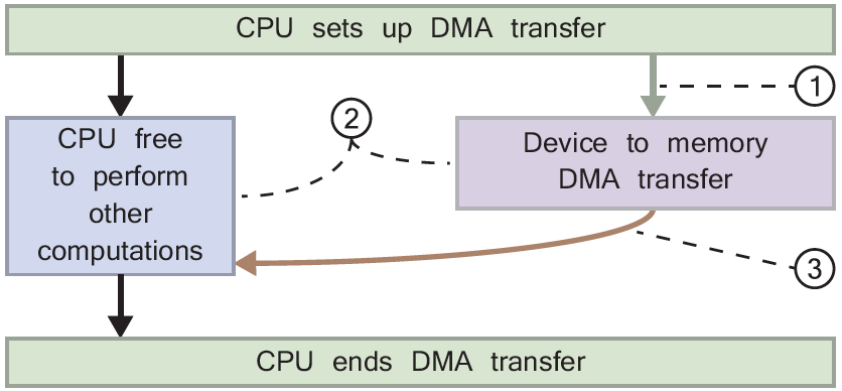


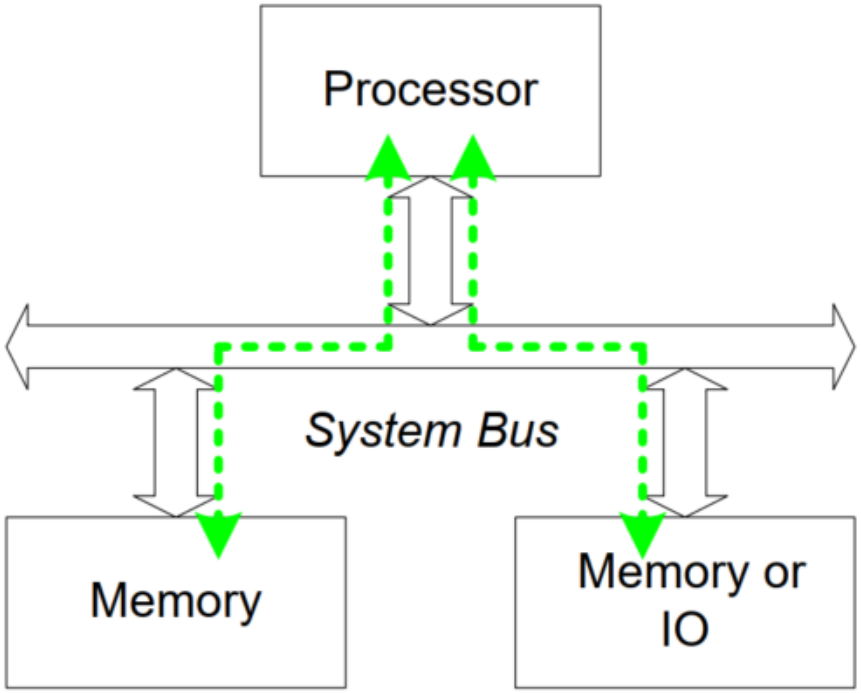
- This means the processor is occupied during this process and can not perform other tasks leading to poor performance. (Have you experience your tablet or mobile phone getting hanged while handling large multi-media files?)
- Data movement takes time -> Performance degradation
- Processors are not designed for burst data transfers -> low efficiency

Direct Memory Access (DMA)

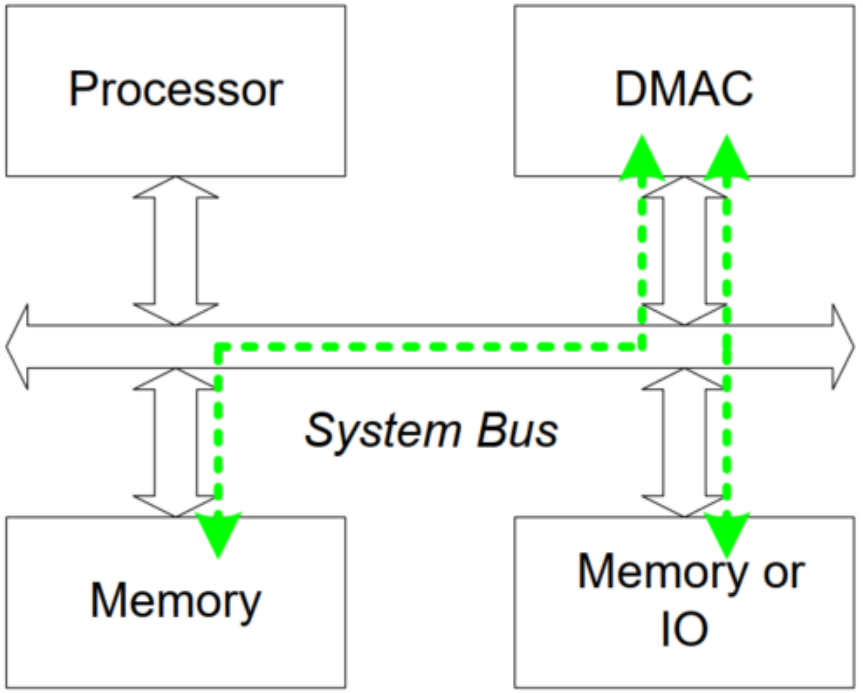
- One way of reducing the burden on the processor is to use Direct Memory Access (DMA) to perform memory transfers.
- Using this approach, the processor issues a memory transfer request to the DMA controller, which will then perform the memory transaction.
- This allows the processor to perform other tasks while the DMA controller performs the transfer.
- In this situation, the DMA controller acts as both a bus master and a bus slave.
- As a master, the DMA controller communicates with the memory controller while arbitrating for the bus.
- While acting as a slave, the DMA controller sets up memory transfers by responding to requests from bus masters (the processor, in most cases).

Direct Memory Access (DMA)





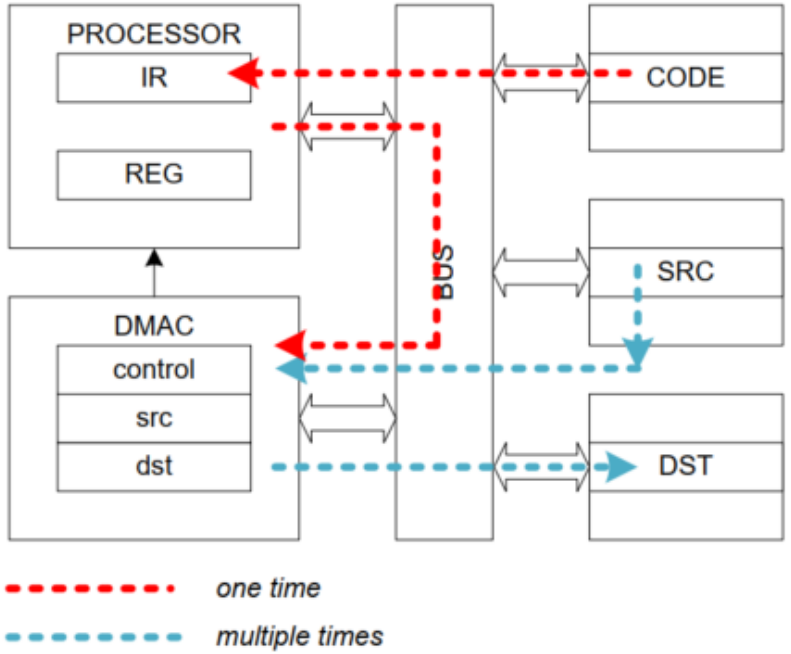
Programmed transfer

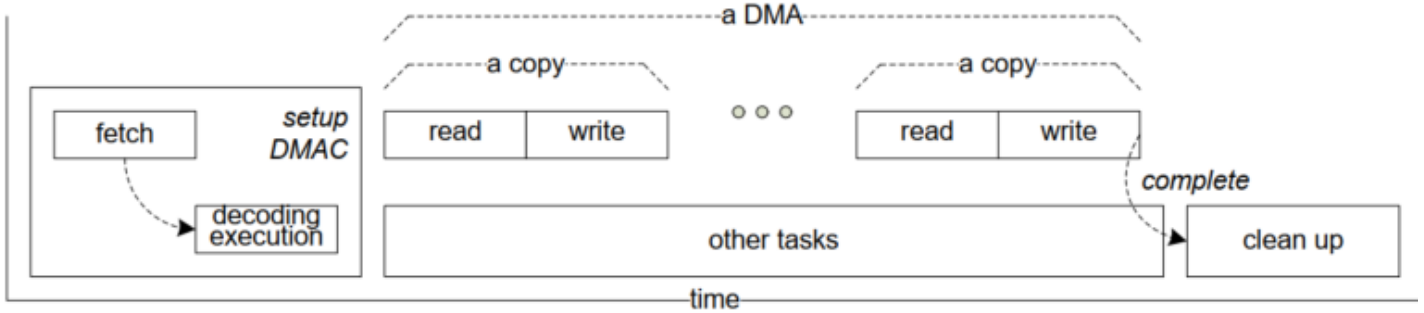


Direct Memory Access transfer

Direct Memory Access (DMA)

- DMA operation involves DMA controller (DMAC) which enables data transfer from one resource to another resource in a computer system without the involvement of the processor
- Data transfer types: Memory to memory, memory to IO, IO to IO and IO to memory





Direct Memory Access (DMA)

- Each DMA cycle will typically result in at least two bus cycles: either a peripheral read followed by a memory write or a memory read followed by a peripheral write, depending on the transfer base addresses.
- The DMA controller itself does no processing on this data. It just transfers the bytes as instructed (by CPU) in its configuration registers.

Direct Memory Access (DMA)

- DMA controllers require initialization by software.
- Typical setup parameters include the base address of the source area, the base address of the destination area, the length of the block, and whether the DMA controller should generate a processor interrupt once the block transfer is complete.

Direct Memory Access (DMA)

- Allow IO devices to access memory directly without involving CPU.
- Useful for block storage devices e.g. disk drives
- DMA permits the peripheral, such as a UART, ethernet to transfer data directly to or from memory without having each byte (or word) handled by the processor.
- Thus DMA enables more efficient use of interrupts, increases data throughput.
- It may lead to reduction in hardware costs by eliminating the need for peripheral-specific FIFO buffers.
- If both CPU and device need bus access, one has to wait for few clock cycles

Direct Memory Access (DMA): Bus Control

- In a typical DMA transfer, some event (such as an incoming data-available signal from a UART, ethernet) notifies a separate device called the *DMA controller* that data needs to be transferred to memory.
- The DMA controller then asserts a *DMA request* signal to the CPU, asking its permission to use the bus.
- The CPU completes its current bus activity, stops driving the bus, and returns a *DMA acknowledge* signal to the DMA controller.
- The DMA controller then reads and writes one or more memory bytes, driving the address, data, and control signals as if it were itself the CPU.
- When the transfer is complete, the DMA controller stops driving the bus and deasserts the DMA request signal.
- The CPU can then remove its DMA acknowledge signal and resume control of the bus.

Direct Memory Access (DMA)

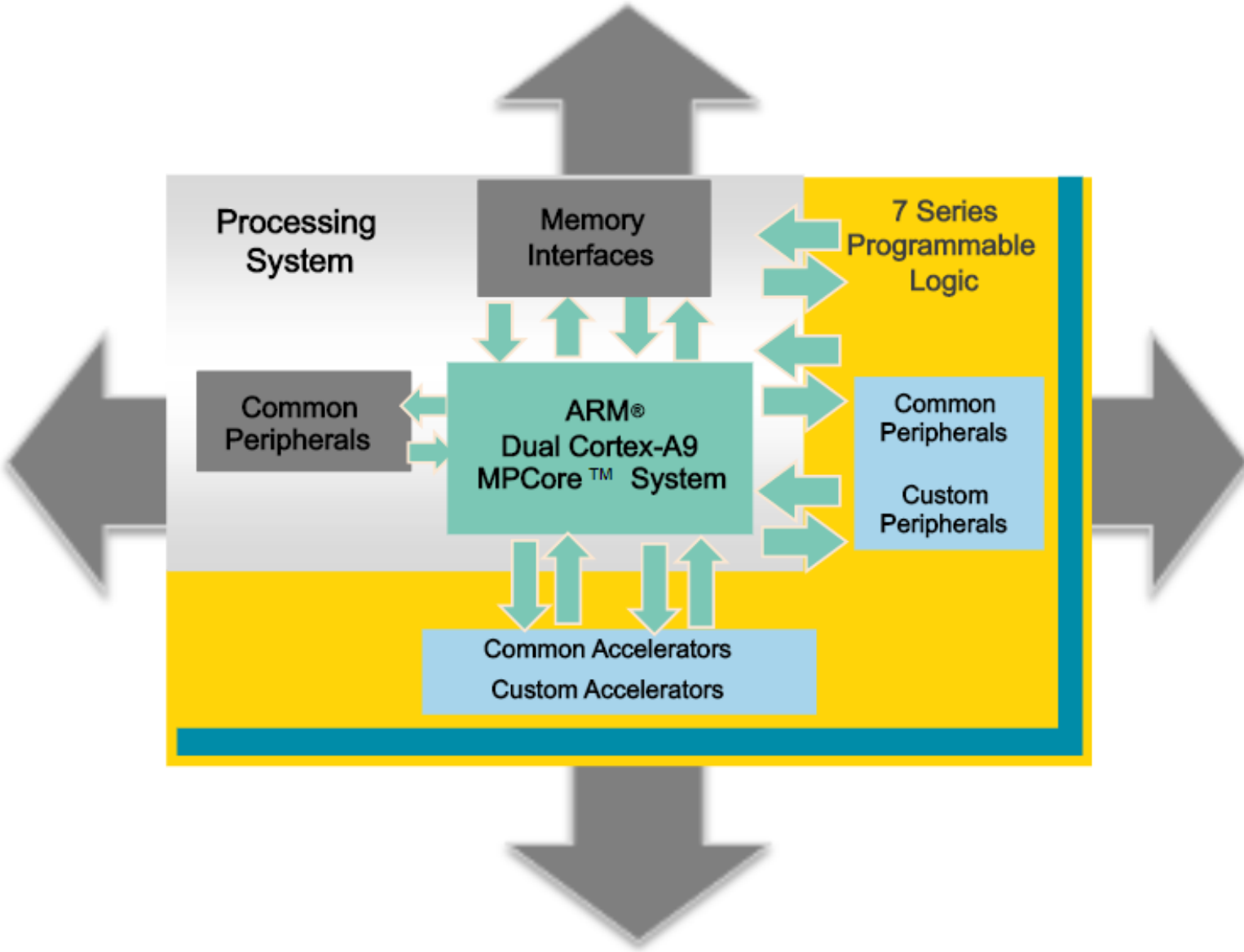
- DMA operations can be performed in either burst or single-cycle mode. Some DMA controllers support both.
- In burst mode, the DMA controller keeps control of the bus until all the data buffered by the requesting device has been transferred to memory (or when the output device buffer is full, if writing to a peripheral).
- In single-cycle mode, the DMA controller gives up the bus after each transfer.
- This minimizes the amount of time that the DMA controller keeps the processor off of the memory bus, but it requires that the bus request/acknowledge sequence be performed for every transfer.
- This overhead can result in a drop in overall system throughput if a lot of data needs to be transferred.

Direct Memory Access (DMA): Model Selection

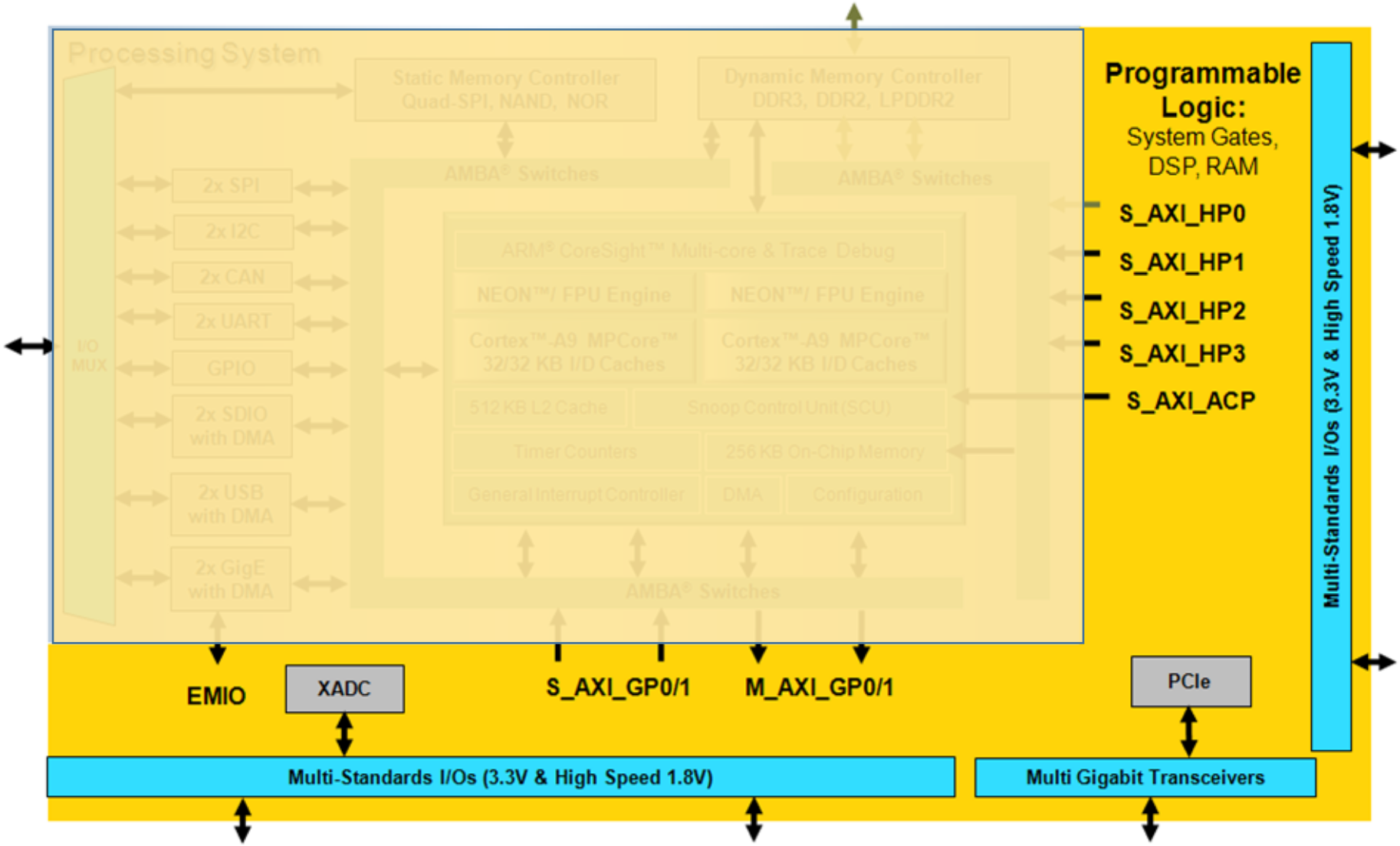
- In most designs, you would use single cycle mode if your system cannot tolerate more than a few cycles of added interrupt latency.
- Likewise, if the peripheral devices can buffer very large amounts of data, causing the DMA controller to tie up the bus for an excessive amount of time, single-cycle mode is preferable.
- How to configure DMA in Zynq SoC: To be discussed in next labs or AELD.

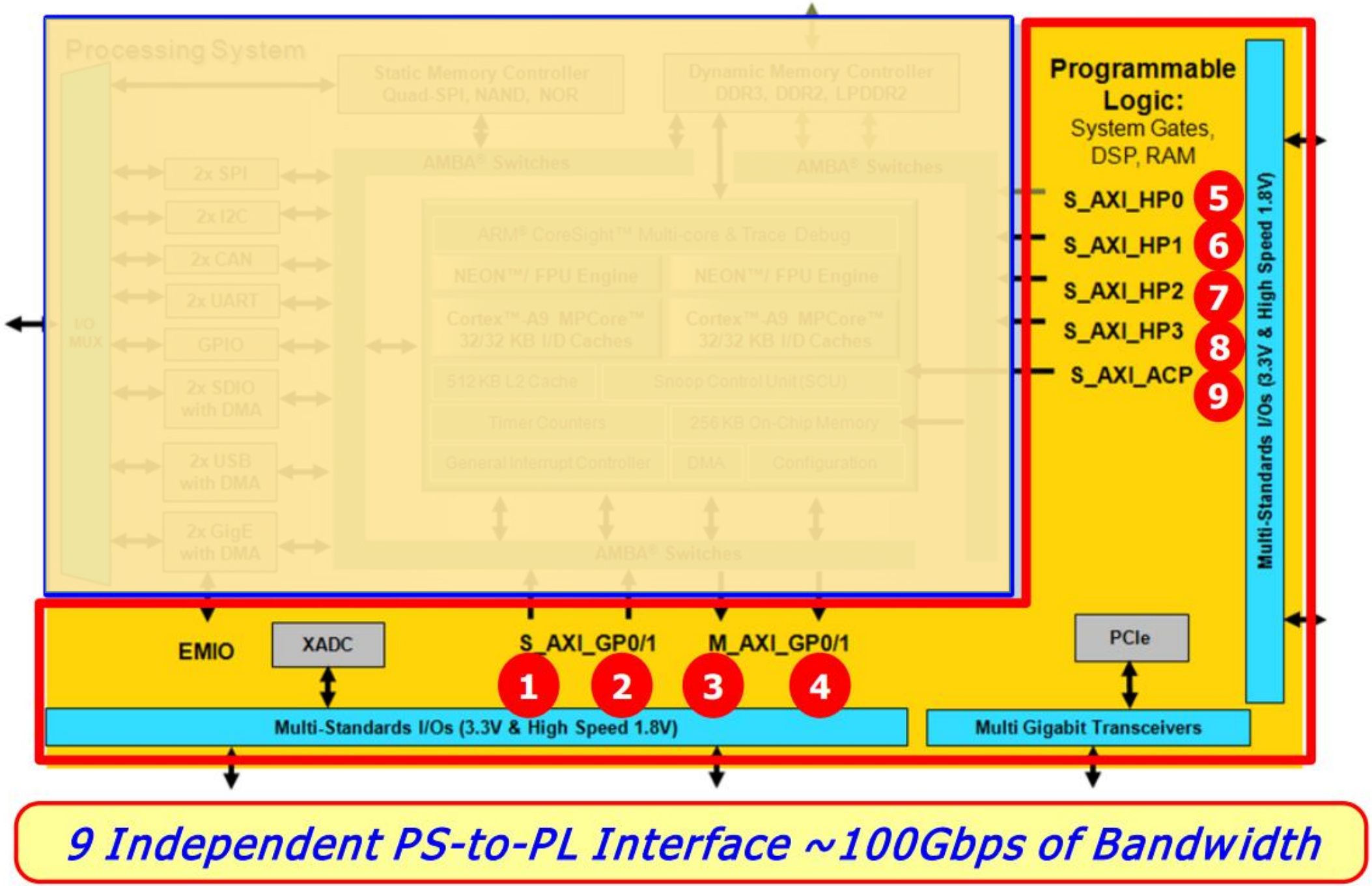
Zynq SoC

Zynq Architecture: PS and PL



Zynq Architecture: PS and PL





9 Independent PS-to-PL Interface ~100Gbps of Bandwidth