Bus, DMA and Accelerator

Chixiao Chen

Announcement

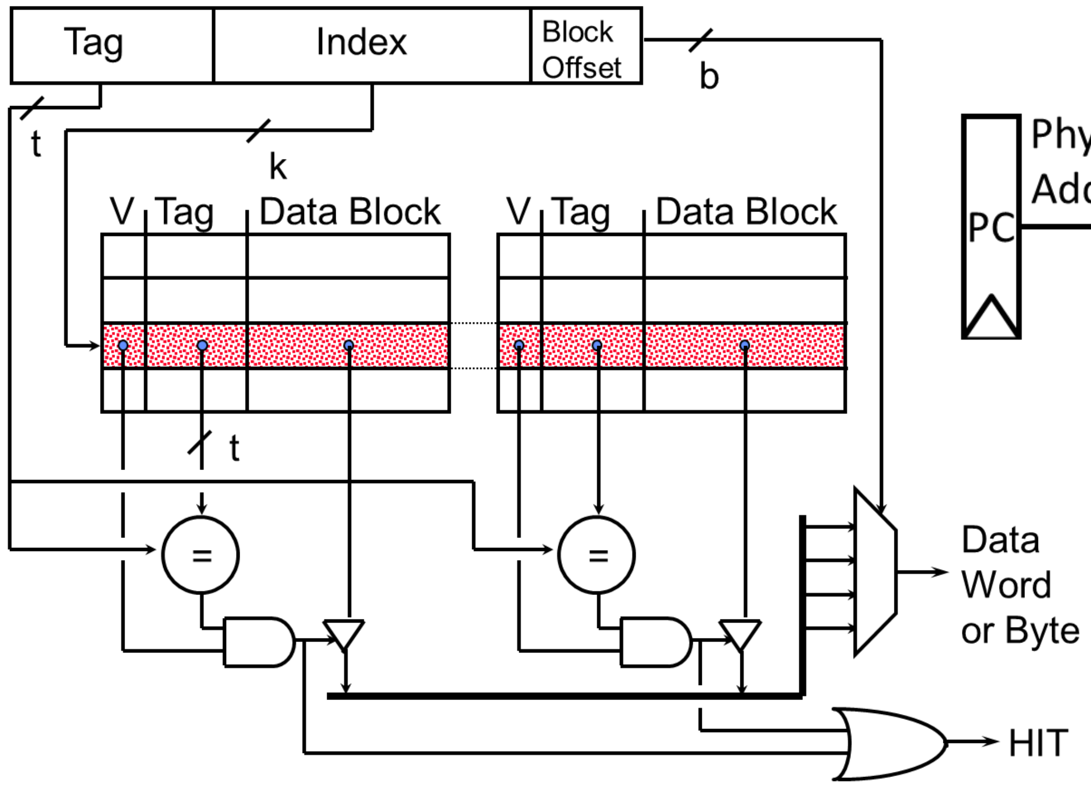
- HW 2 is online

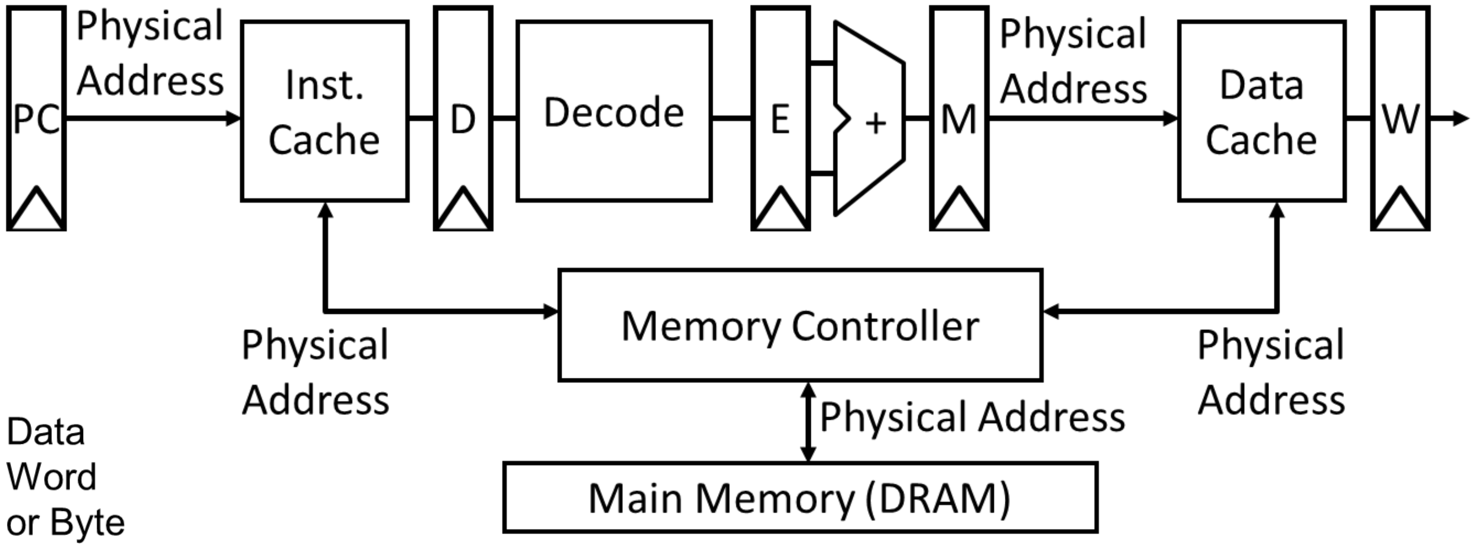
- We will not have classes next week.

- Complete HW2 next week.

Last time Review

- Memory Hierarchy : Using cache instead of direct access on Memory





Outline

- IO Devices
- Bus Introduction (AMBA and AXI4)
- DMA
- Accelerator
- NOC

A Computer is Useless without I/O

- Storage (DRAM/SSD)
- Interface(keyboard, mouse, downloader)
- Networks with other processors

How to control/access them ?

Treat all devices as memory

- A memory-mapped interface
  - Processor can write & read to control registers to tell the device what to do
  - Processor can write & read data to the device

- The "device driver" runs in the operating system
  - Provides a device independent abstraction to the rest of the OS
  - And user programs have to ask the OS for permission
- Programmed I/O:
  - The CPU is responsible to transfer data to the devices as well as commands

Reading register bits with MMIO

- How?
  - Let’s say that red, green, and blue pushbuttons are mapped to memory address 0xB0000003 at bits 0, 1, and 2, respectively
  - When pushed, we will read a ‘1’ from the corresponding bit
  - When released, we will read a ‘0’ from the corresponding bit

```
Address \ Bit    31     24 23     16 15      8  7    210
0xB0000003-------- -------- -------- -----⊙⊙⊙
```

```
#define  PBS_REG 0xB0000003

int isBlueButtonPressed() {
  return (*((uint32_t *)PBS_REG) & 0x00000004) >> 2;
}
```

Writing register bits with MMIO

- How?
  - Let’s say that red, green, and blue LEDs are mapped to memory address 0xB0000004 at bits 0, 1, and 2, respectively
  - Writing a ‘1’ (‘0’) to the corresponding bit turns on (off) the LED

```
Address \ Bit    31      24 23      16 15      8 7     210
0xB0000004-------- -------- -------- -----🟦🟩🟥
```

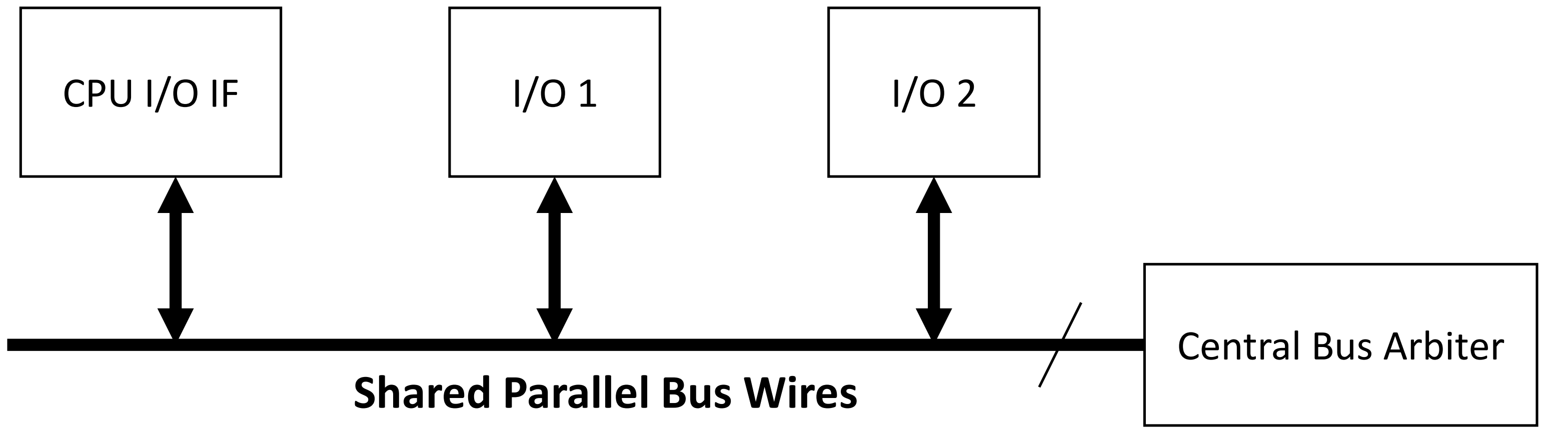
  - When updating (set/clear) bits, must preserve remaining contents
  - Use bitmask to select target bits for manipulation, e.g.

```
#define  LED_REG 0xB0000004

void setRedLed() {
  *((uint32_t *)LED_REG) |= 0x1;
}

void setGreenLed() {
 *((uint32_t *)LED_REG) |= 0x2;
}
```

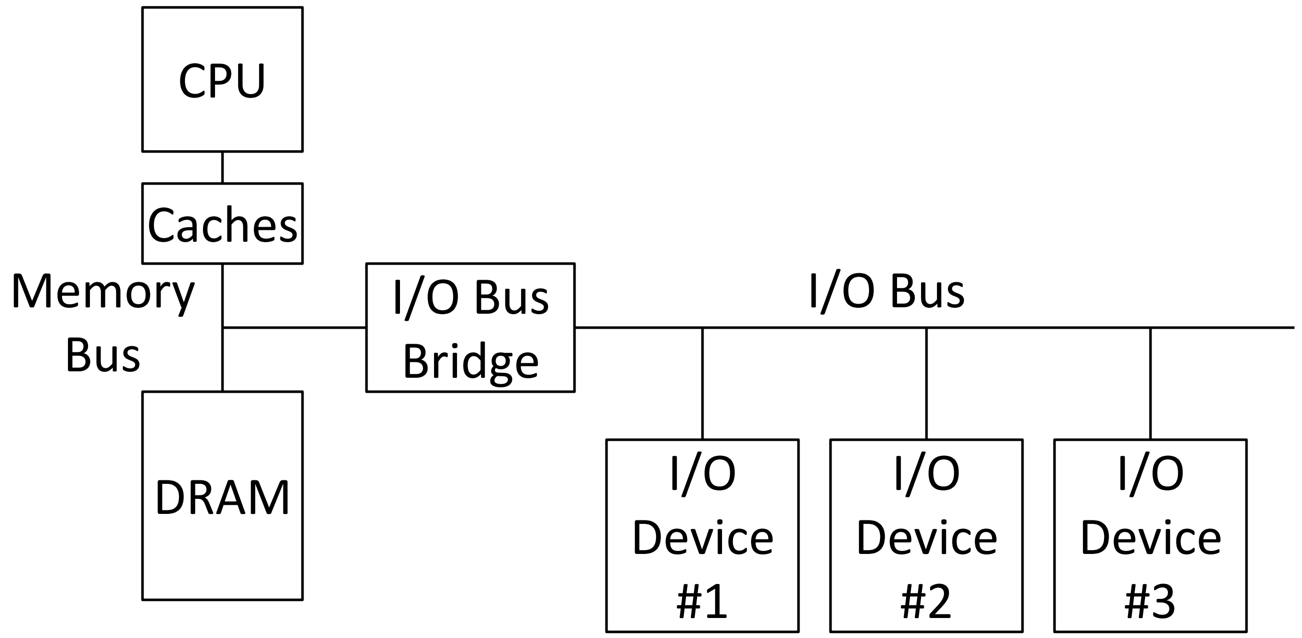
Shared Parallel Bus



- Parallel bus clock rate limited by clock skew across long bus (~100MHz)
- High power to drive large number of loaded bus lines
- Central bus arbiter adds latency to each transaction, sharing limits throughput
- Expensive parallel connectors and backplanes/cables (all devices pay costs)
- Examples: VMEbus, Sbus, ISA bus, PCI, SCSI, IDE

Simple I/O Bus Structure

- Some range of physical addresses map to I/O bus devices
- I/O bus bridge reduces loading on critical CPU-DRAM bus
- Devices can be “slaves”, only responding to I/O bus requests
- Devices can be “masters”, initiating I/O bus transfers



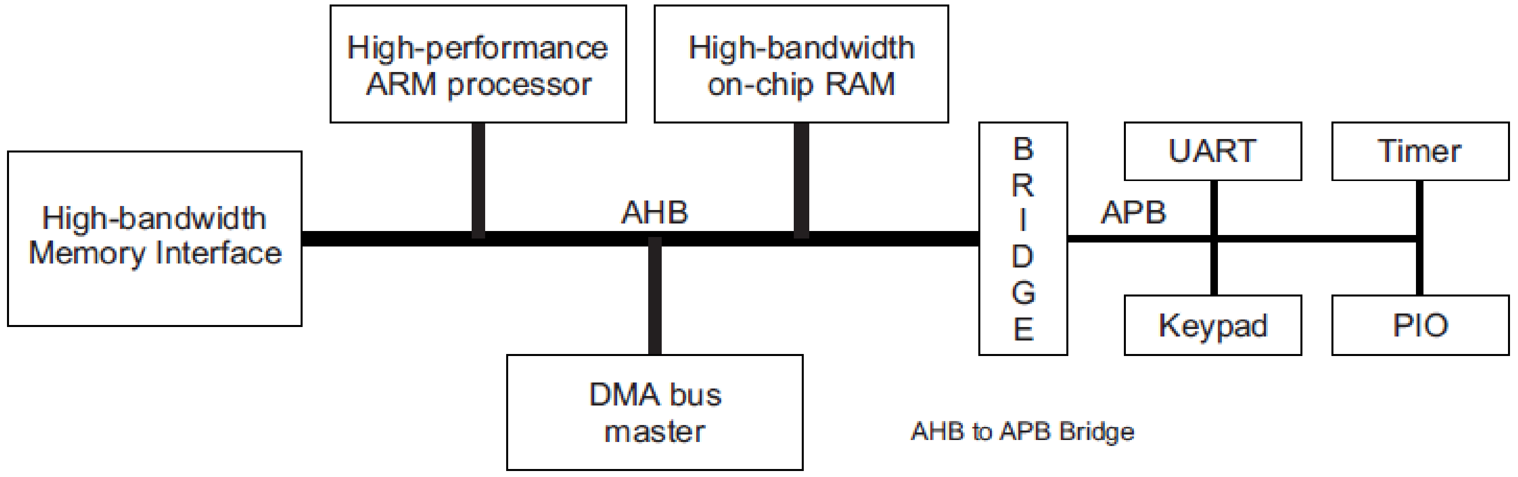
Advanced Microcontroller Bus Architecture (AMBA)

Advanced High-performance Bus (AHB)

- High performance
- Pipelined operation
- Burst transfers
- Multiple bus masters
- Split transactions

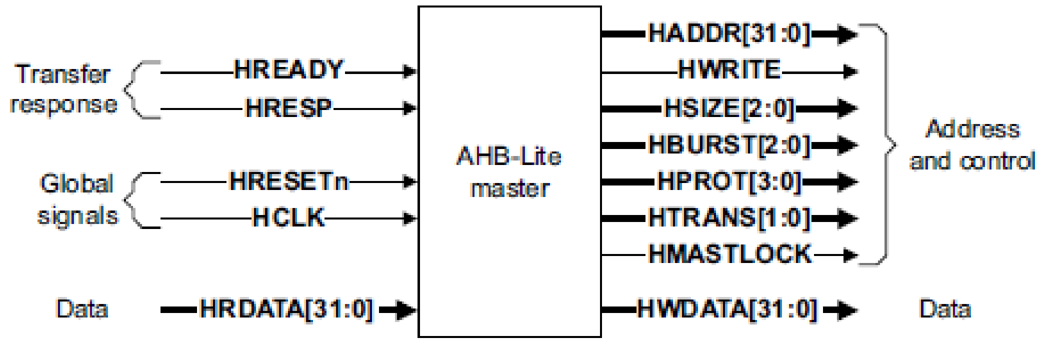
Advanced Peripheral Bus (APB)

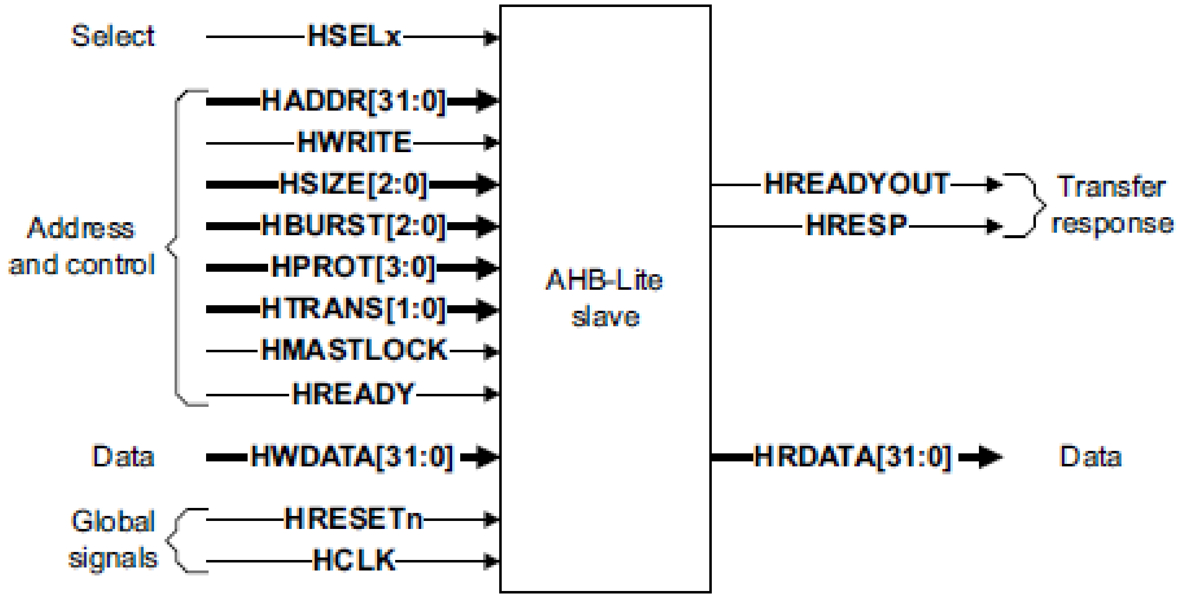
- Low power
- Latched address/control
- Simple interface
- Suitable of many peripherals


High-performance ARM processor
High-bandwidth on-chip RAM
High-bandwidth Memory Interface
AHB
DMA bus master
BRIDGE
APB
UART
Timer
Keypad
PIO
AHB to APB Bridge

AHB-lite Master / Slave Interface

- Global signals :
  - HCLK
  - HRESETn
- Master out/slave in
  - HADDR (address)
  - HWDATA (write data)
  - Control
    - HWRITE
    - HSIZE
    - HBURST
    - HPROT
    - HTRANS
    - HMASTLOCK
- Slave out/master in
  - HRDATA (read data)
  - HREADY
  - HRESP

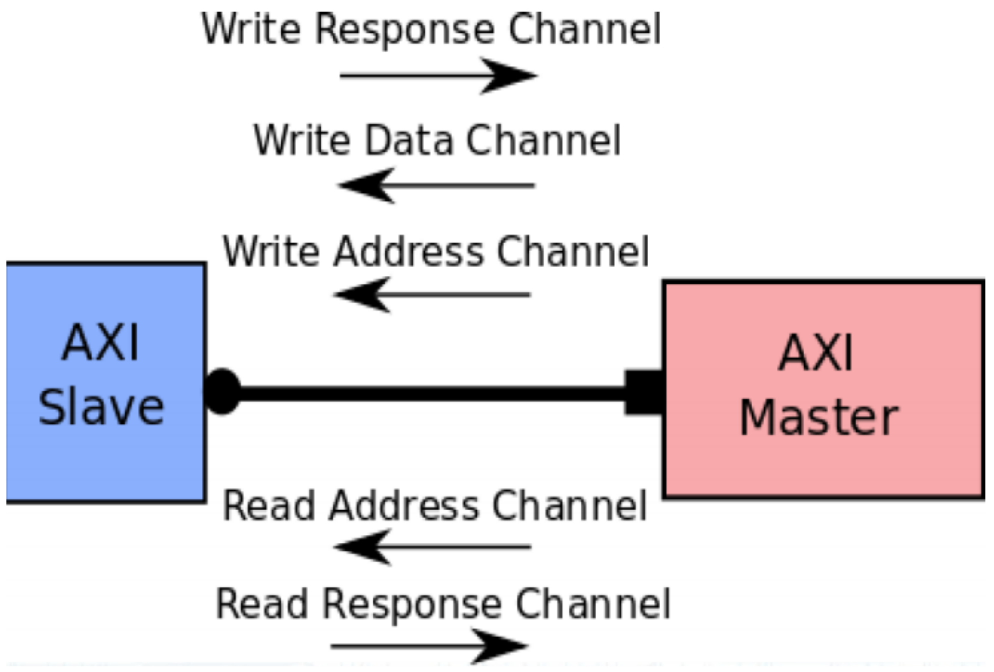




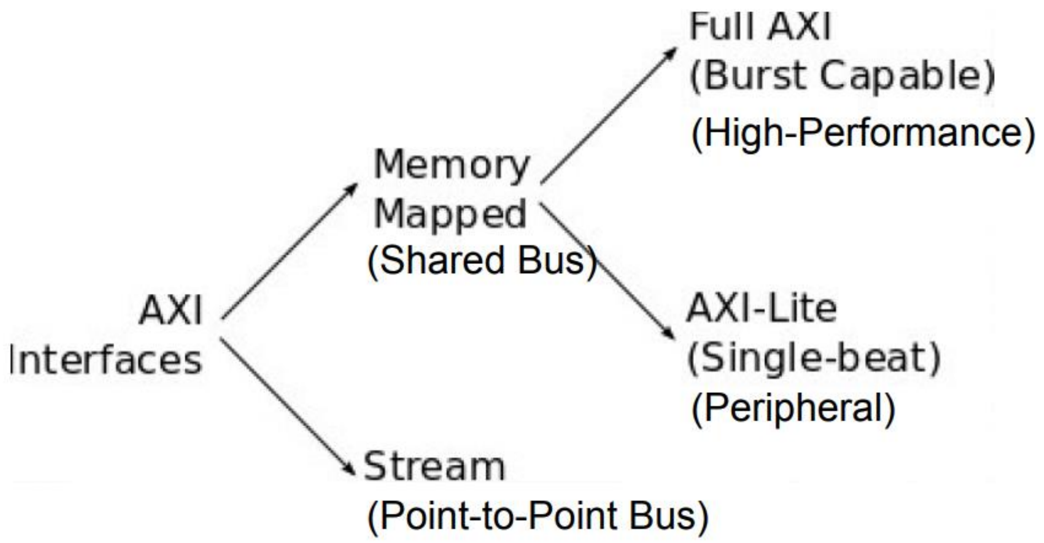
AMBA Advanced eXtensible Interface 4 (AXI4)

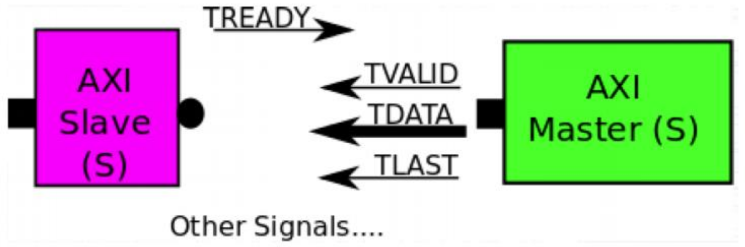
- The fourth generation of AMBA interface defined in the AMBA 4 specification, targeted at high performance, high clock frequency systems. Introduced by ARM in 2010.

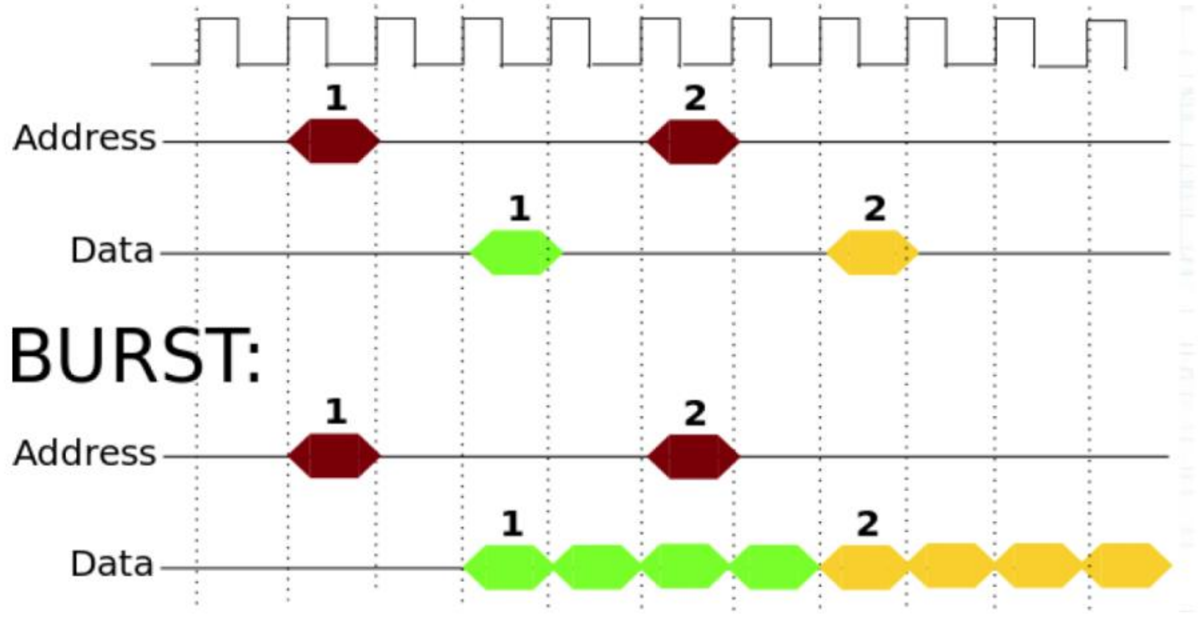
- AXI 4 use 5 channels!



More Choices on AXI Interfaces



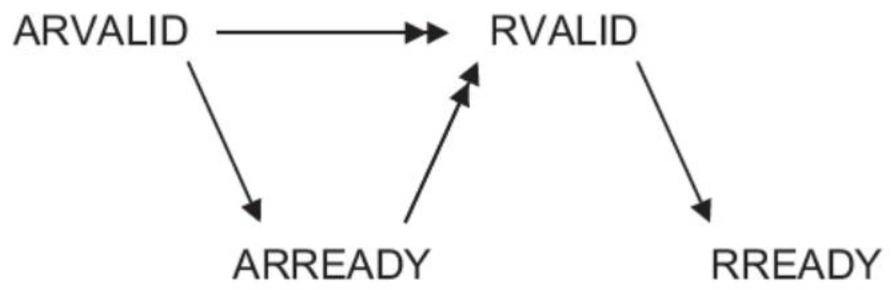




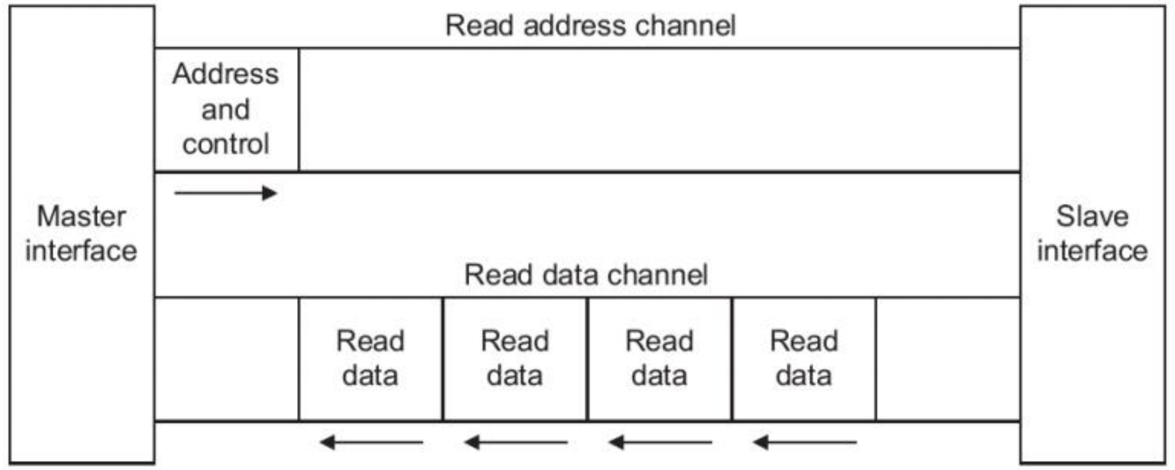
Dedicated Point-to-point Serial Links

- Point-to-point links run at multi-gigabit speed using advanced clock/signal encoding (requires lots of circuitry at each end)
- Lower power since only one well-behaved load
- Multiple simultaneous transfers
- Cheap cables and connectors (trade greater endpoint transistor cost for lower physical wiring cost), customize bandwidth per device using multiple links in parallel
- Examples: Ethernet, Infiniband, PCI Express, SATA, USB, Firewire, etc.

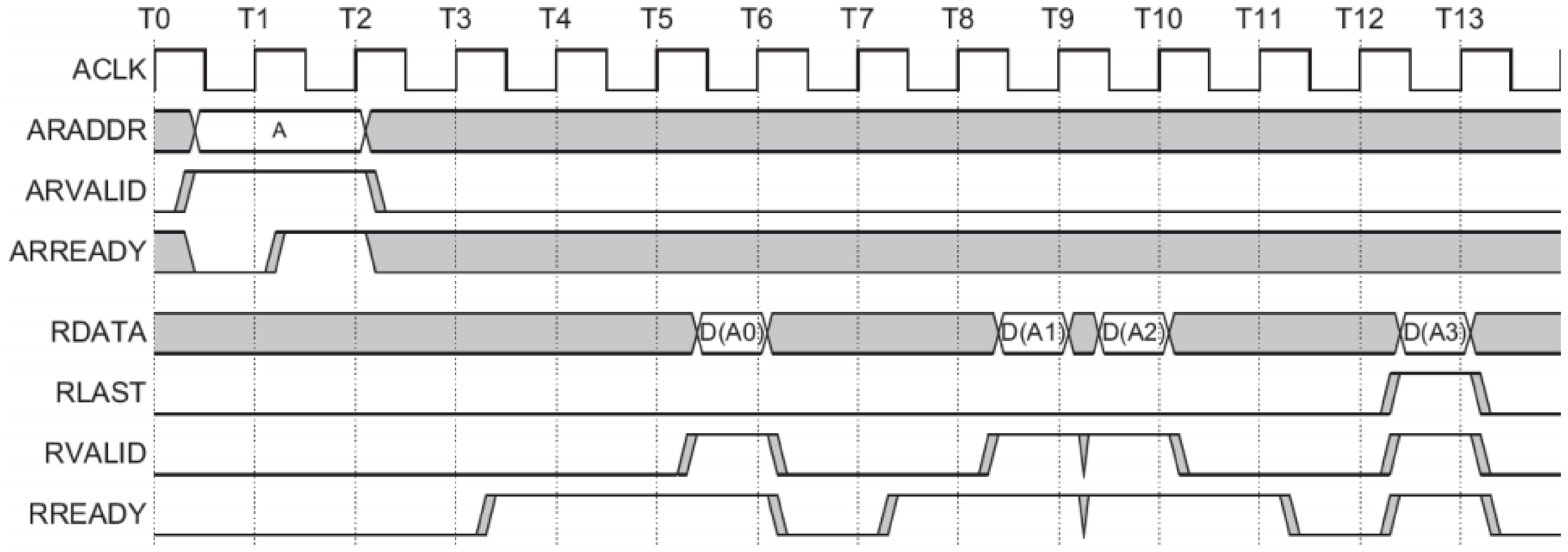
AXI4 Read



ARVALID
RVALID
ARREADY
RREADY

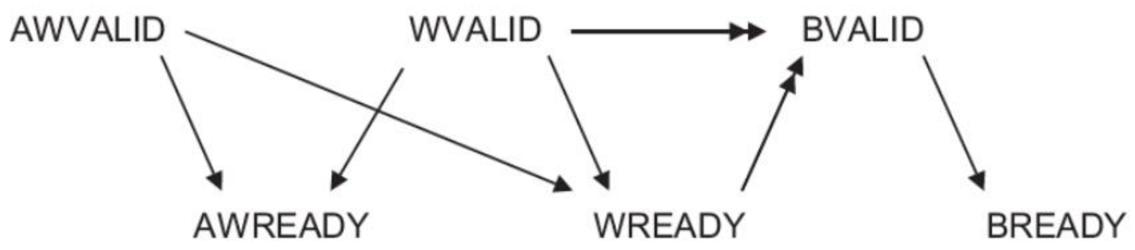


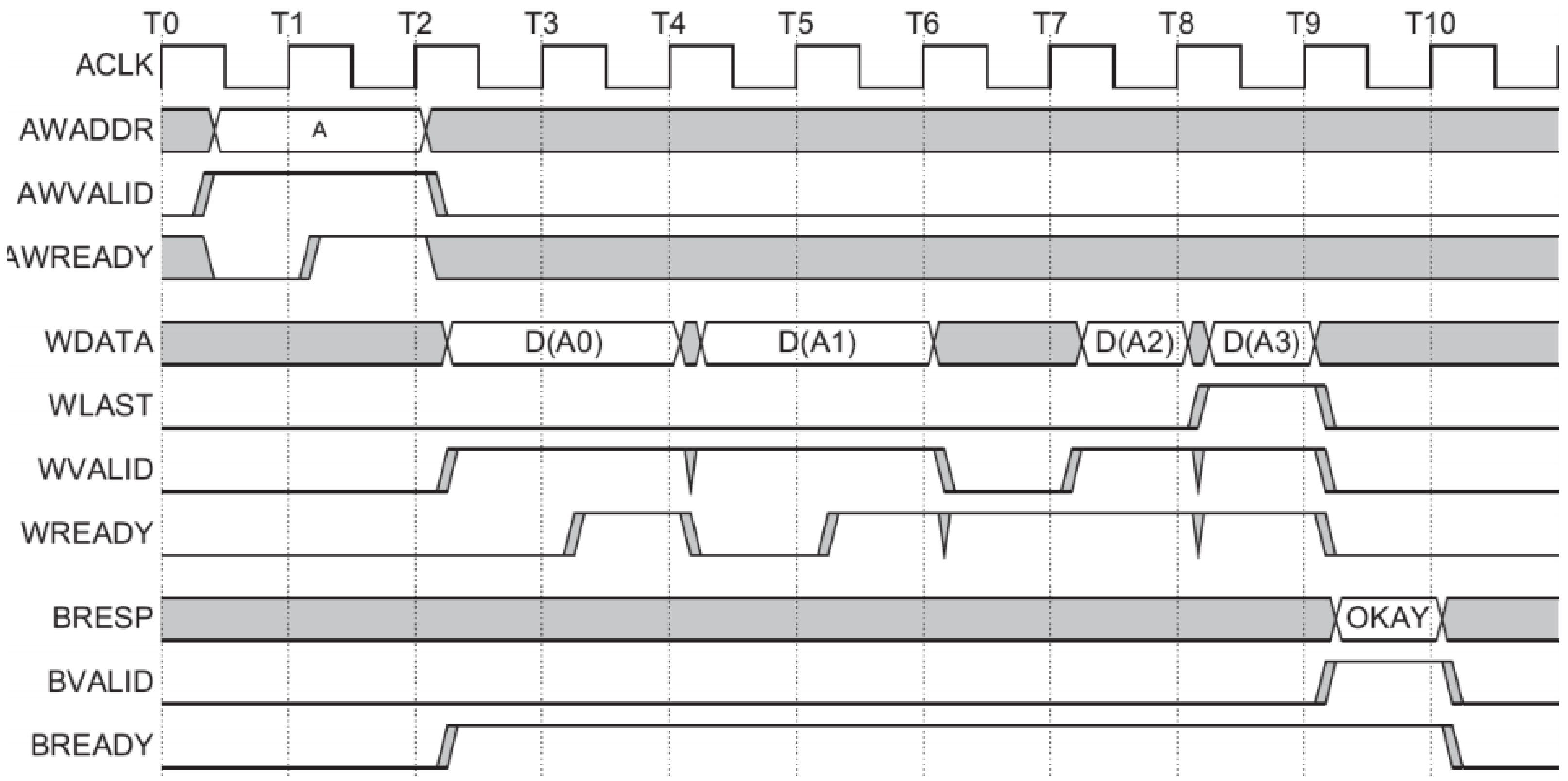
Read address channel
Address and control
Master interface
Slave interface
Read data channel
Read data
Read data
Read data
Read data

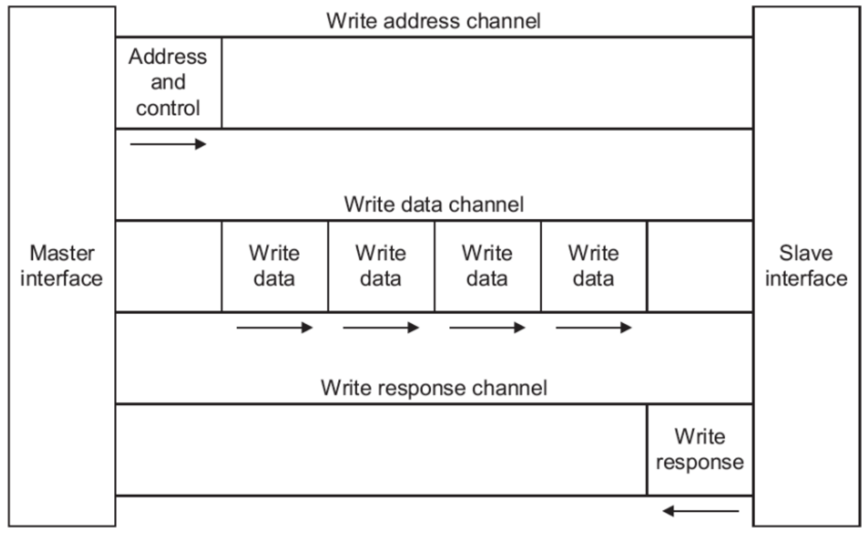


T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
T13
ACLK
ARADDR
A
ARVALID
ARREADY
RDATA
D(A0)
D(A1)
D(A2)
D(A3)
RLAST
RVALID
RREADY

AXI4 Write


AWVALID
WVALID
BVALID
AWREADY
WREADY
BREADY

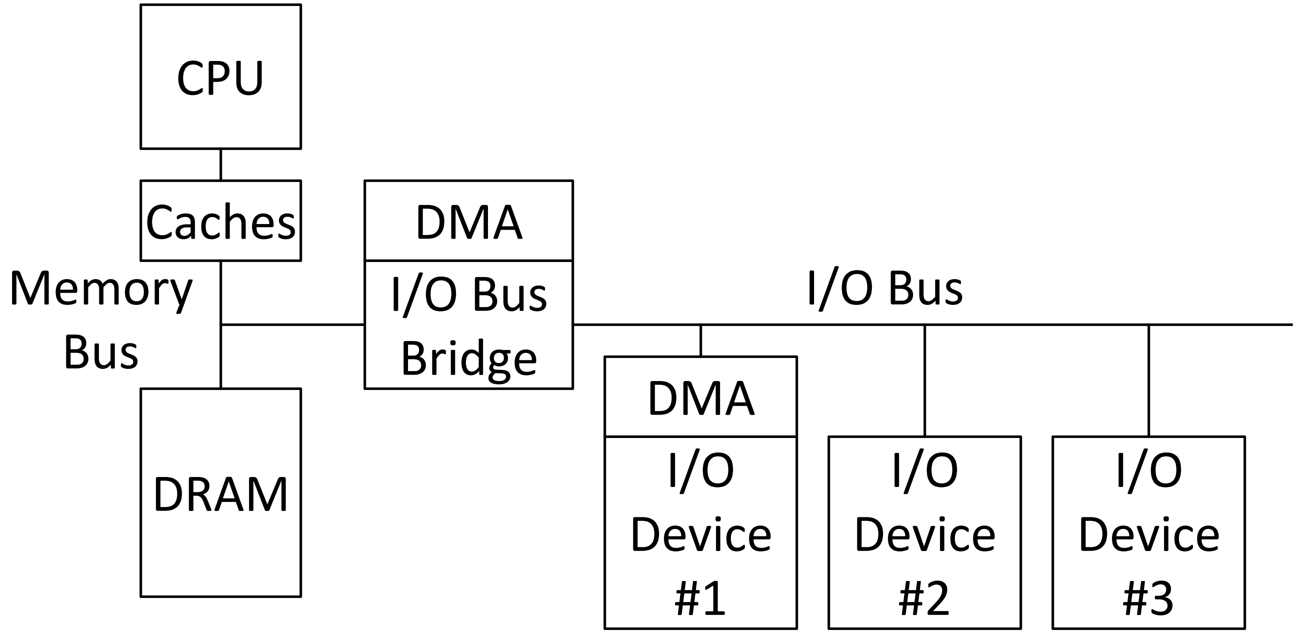

T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
ACLK
AWADDR
A
AWVALID
AWREADY
WDATA
D(A0)
D(A1)
D(A2)
D(A3)
WLAST
WVALID
WREADY
BRESP
OKAY
BVALID
BREADY


Write address channel
Address and control
Write data channel
Master interface
Write data
Write data
Write data
Write data
Slave interface
Write response channel
Write response

Direct Memory Access (DMA)

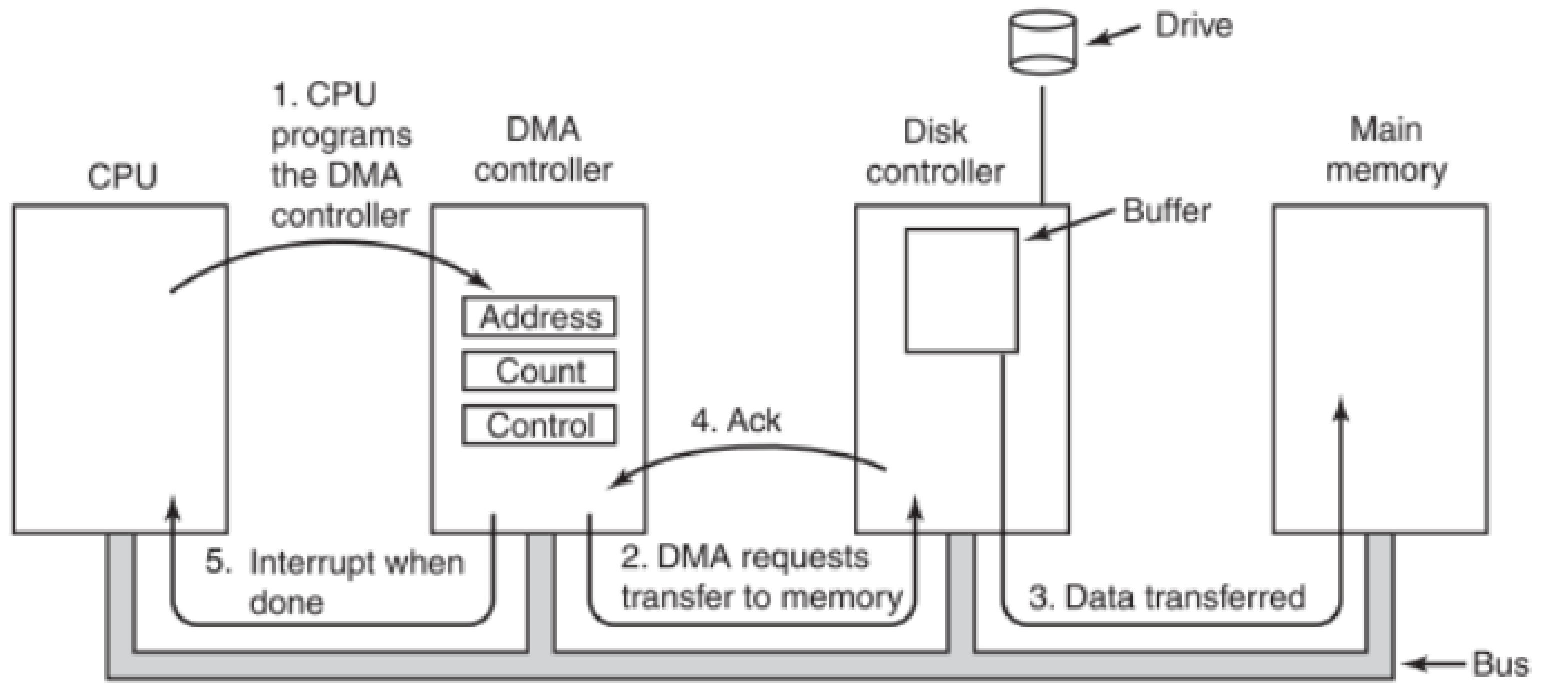
- DMA engines offload CPU by autonomously transferring data between I/O device and main memory. Interrupt/poll for done

- Often, many separate DMA engines in modern systems:
  - Centralized in I/O bridge (usually supporting multiple concurrent channels to different devices), works on slave-only I/O busses



Operation of a DMA Transfer

- DMA engine contains registers written by CPU:
  - Memory address to place data , # of bytes, I/O device #, direction of transfer unit of transfer, amount to transfer per burst



DMA: Some new problems - I

- Where in the memory hierarchy do we plug in the DMA engine?

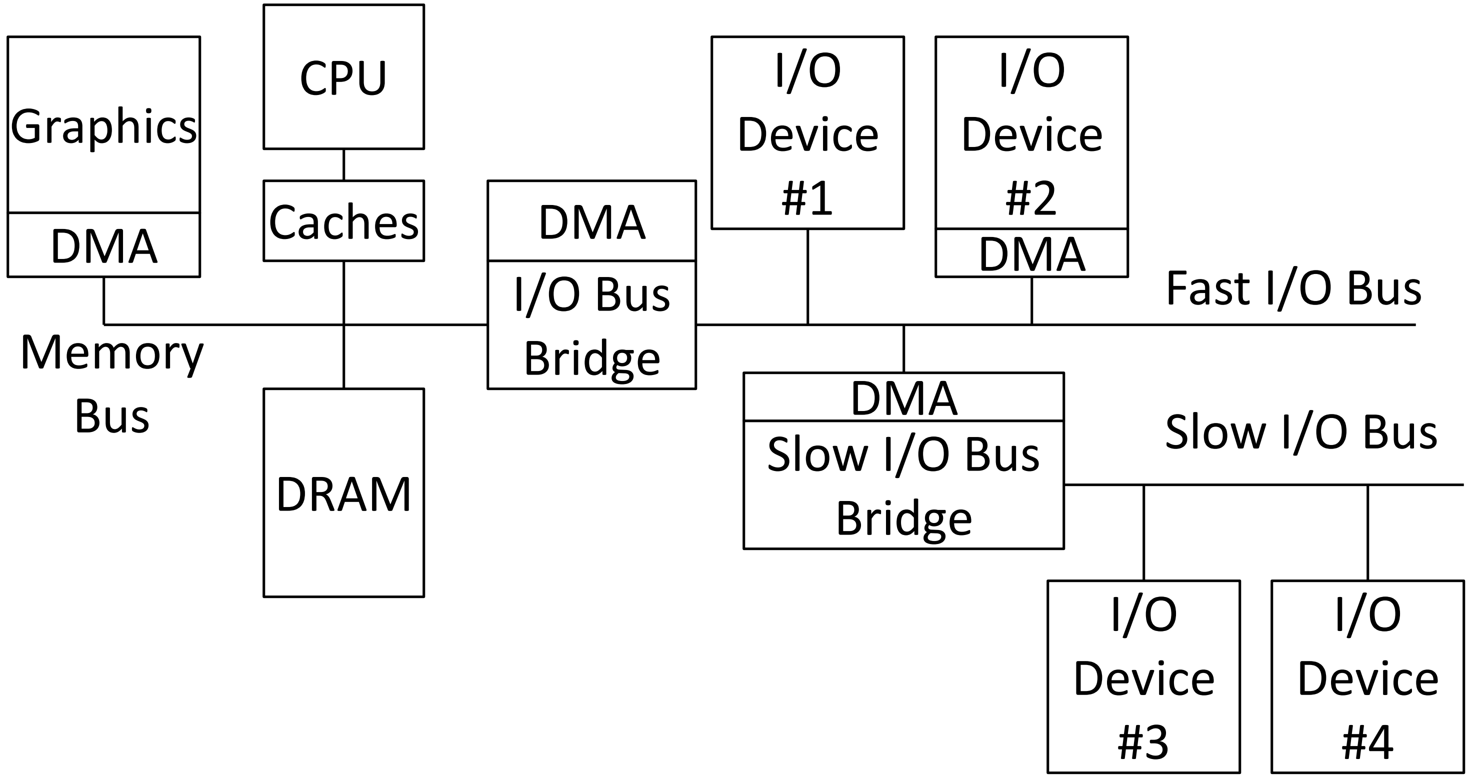
- Answer 1: Between L1 and CPU:
  - Pro: Free coherency
  - Con: Trash the CPU’s working set with transferred data
- Answer 2: Between Last-level cache and main memory
  - Pro: Don’t mess with caches
  - Con: Need to explicitly manage coherency

DMA: Some new problems - II

- How do we arbitrate between CPU and DMA Engine/Device access to memory? Three options:
- Burst Mode : Start transfer of data block, CPU cannot access memory in the meantime
- Cycle Stealing Mode : DMA engine transfers a byte, releases control, then repeats - interleaves processor/ DMA engine accesses
- Transparent Mode: DMA transfer only occurs when CPU is not using the system bus

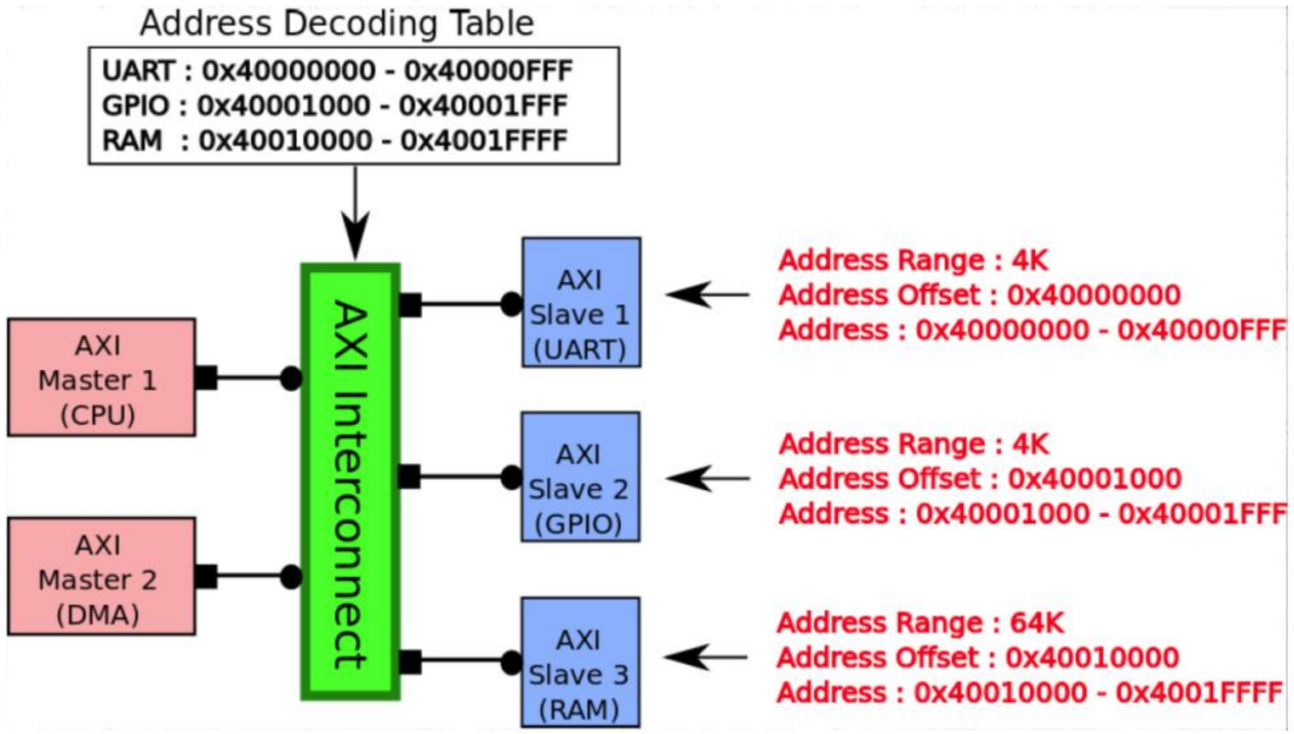
More Complex Bus Structures

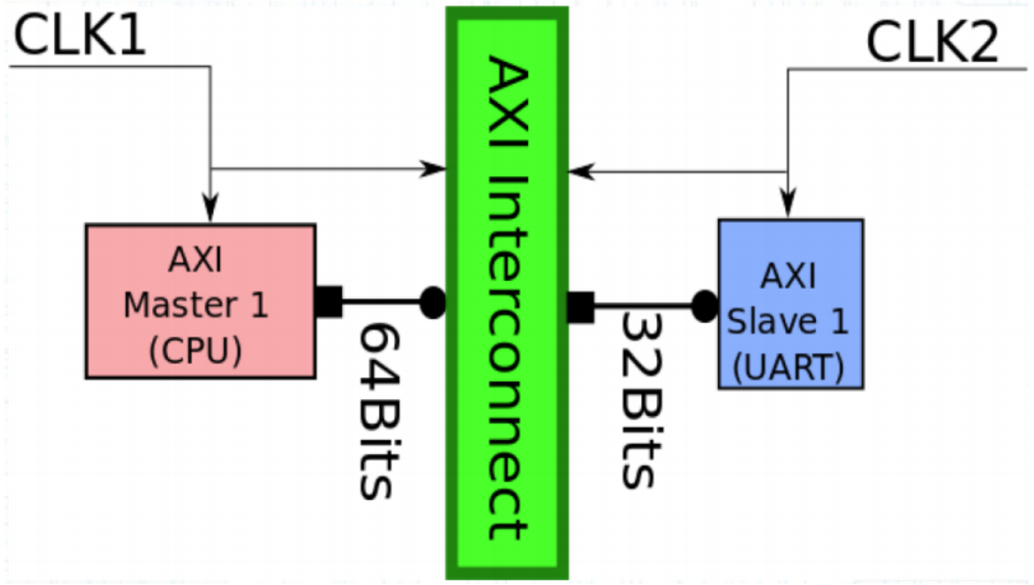
- Match speed of I/O connection to device demands
  - Special direct connection for graphics
  - Fast I/O bus for disk drives, ethernet
  - Slow I/O bus for keyboard, mouse, touchscreen
    - Reduces load on fast I/O bus + less bus logic needed on device



Interface Examples

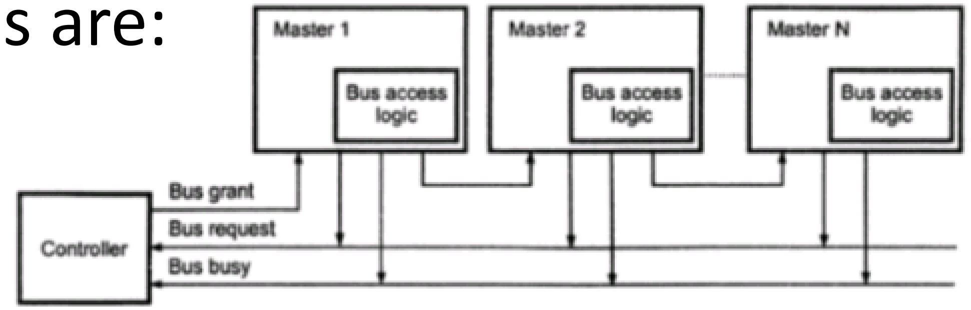
- AXI interconnect between masters and slaves

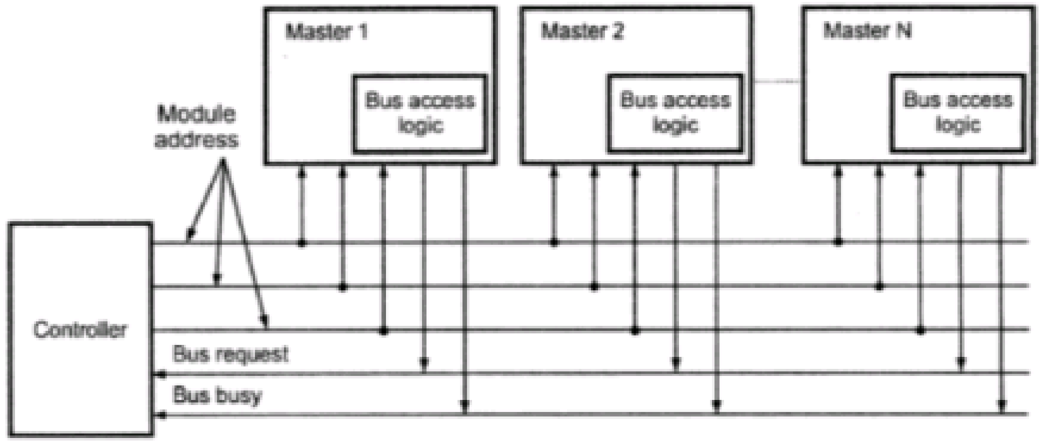

Address Decoding Table
UART : 0x40000000 - 0x40000FFF
GPIO : 0x40001000 - 0x40001FFF
RAM : 0x40010000 - 0x4001FFFF
AXI Interconnect
AXI Master 1 (CPU)
AXI Master 2 (DMA)
AXI Slave 1 (UART)
AXI Slave 2 (GPIO)
AXI Slave 3 (RAM)
Address Range : 4K
Address Offset : 0x40000000
Address : 0x40000000 - 0x40000FFF
Address Range : 4K
Address Offset : 0x40001000
Address : 0x40001000 - 0x40001FFF
Address Range : 64K
Address Offset : 0x40010000
Address : 0x40010000 - 0x4001FFFF

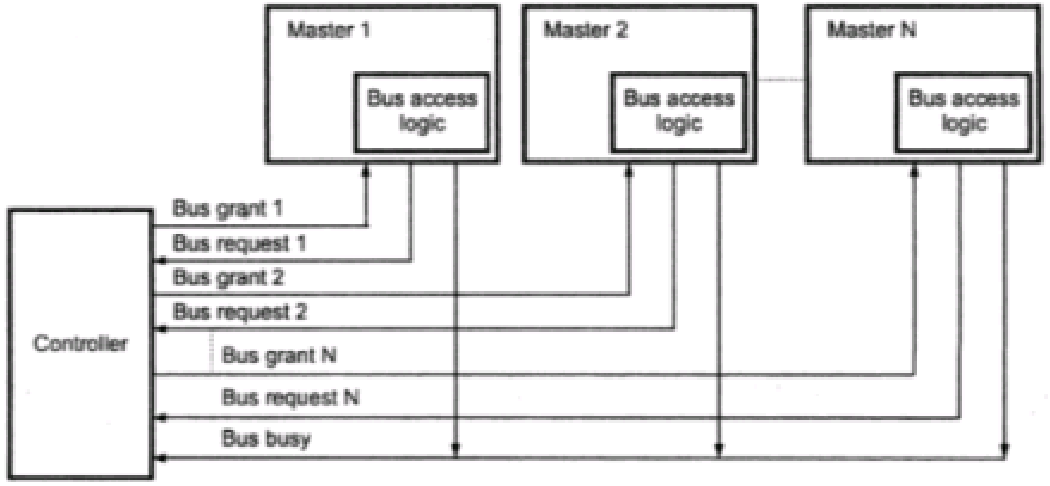


Multi-master: Arbitration needed !

- There are three different arbitration schemes that use the centralized bus arbitration approach. There schemes are:
- a. Daisy chaining
- b. Polling method
- c. Independent request







Accelerators

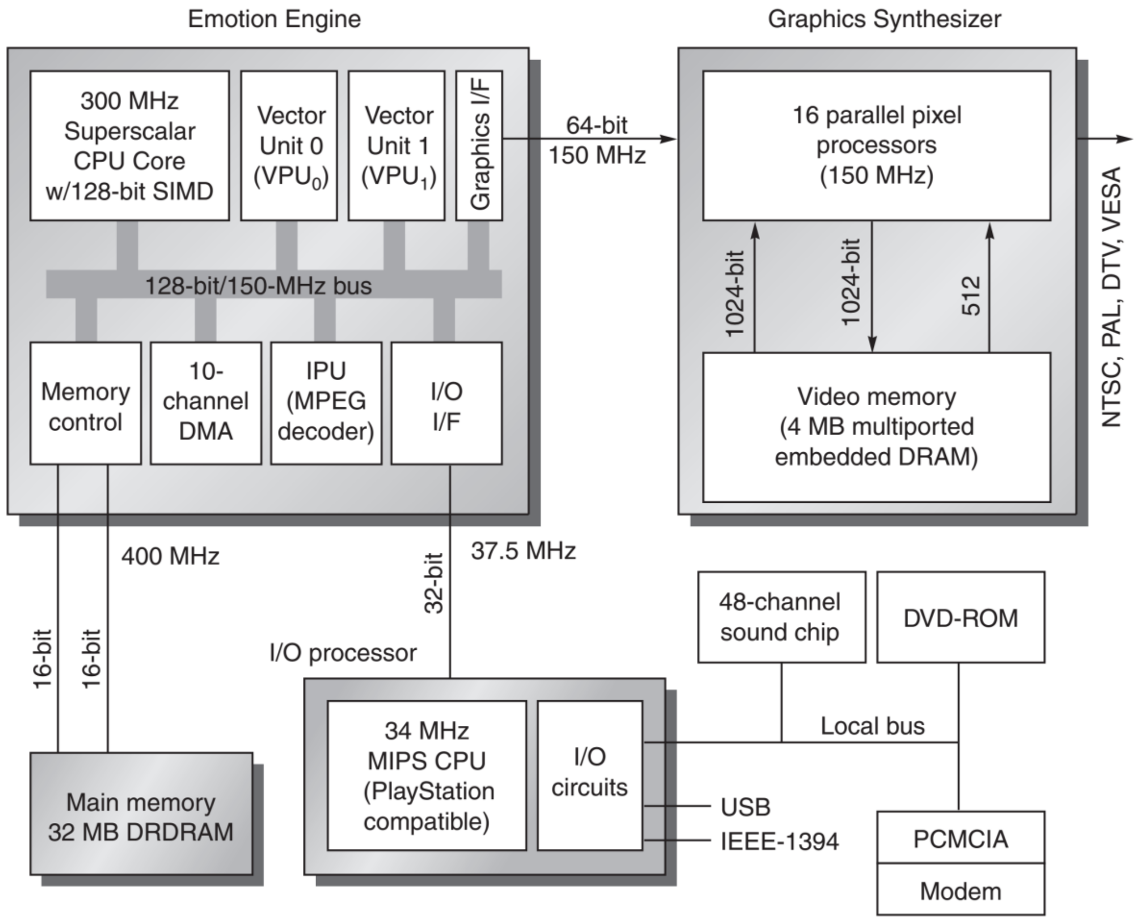
- Use additional computational unit dedicated to some functions?
  - Hardwired logic.
  - Extra CPU.

- Hardware/software co-design: joint design of hardware and software architectures

Accelerator vs. co-processor

- A co-processor executes instructions. Its instructions are dispatched by the CPU.

- An accelerator appears as a device on the bus. It is controlled by registers.

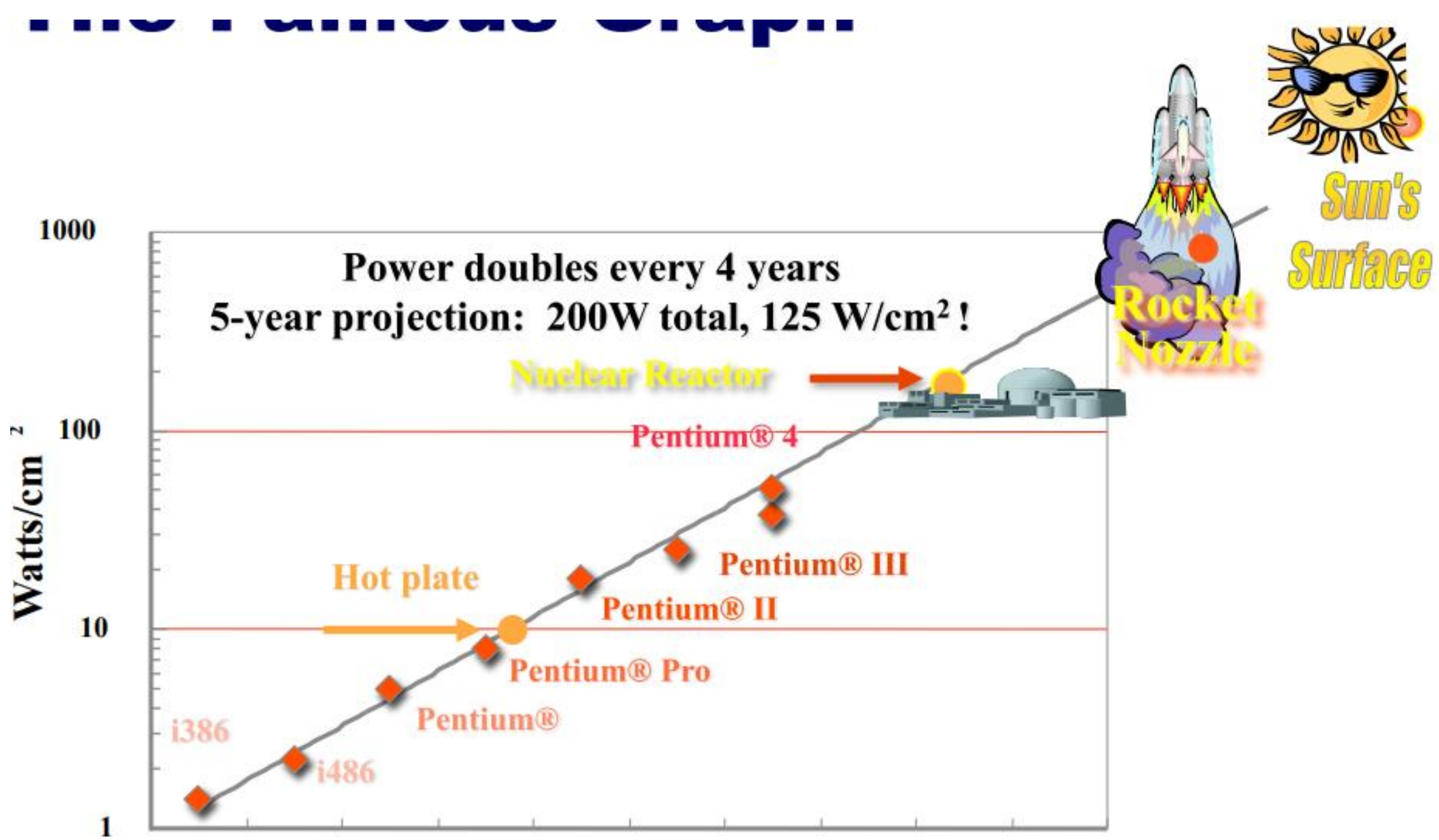


Why Accelerator ? In General

- Better cost/performance
  - Energy efficiency
  - Real-time application
  - Support Data streaming (audio, video, network traffic, real-time monitoring, etc.)
  - Specific “complex” operations: FFT, DCT, EXP, LOG, …
  - Specific “complex” algorithms: ⍰ Neuronal networks, …

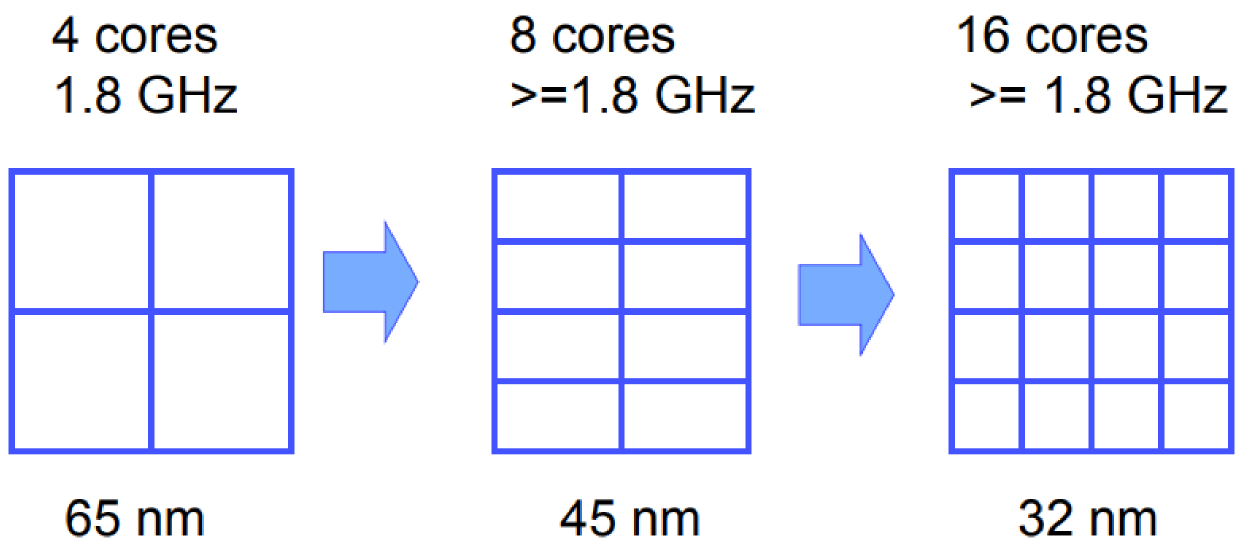
Why Accelerator ? Now it's mandatory

- Moore's Law
- More pipelined stages?
- Multi Core?

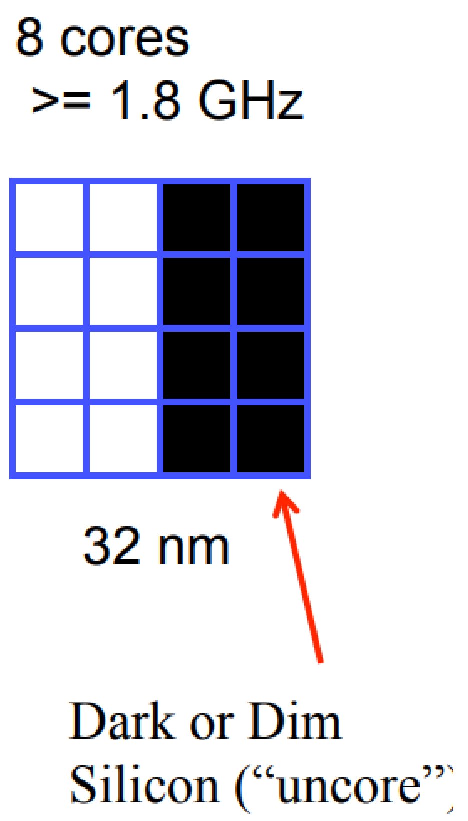


From "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference key note - 1999.

Multicore Roadmap



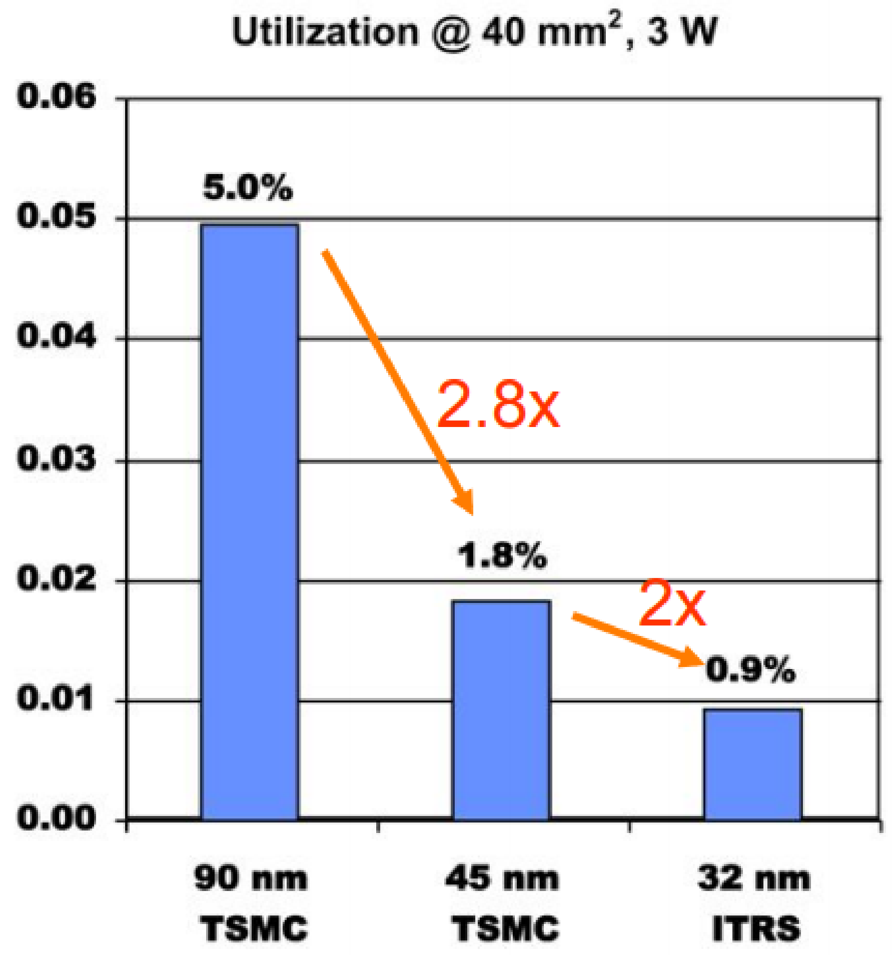
2x cores per generation,
flat or slightly growing frequency



General Purpose Processor hit the utilization wall

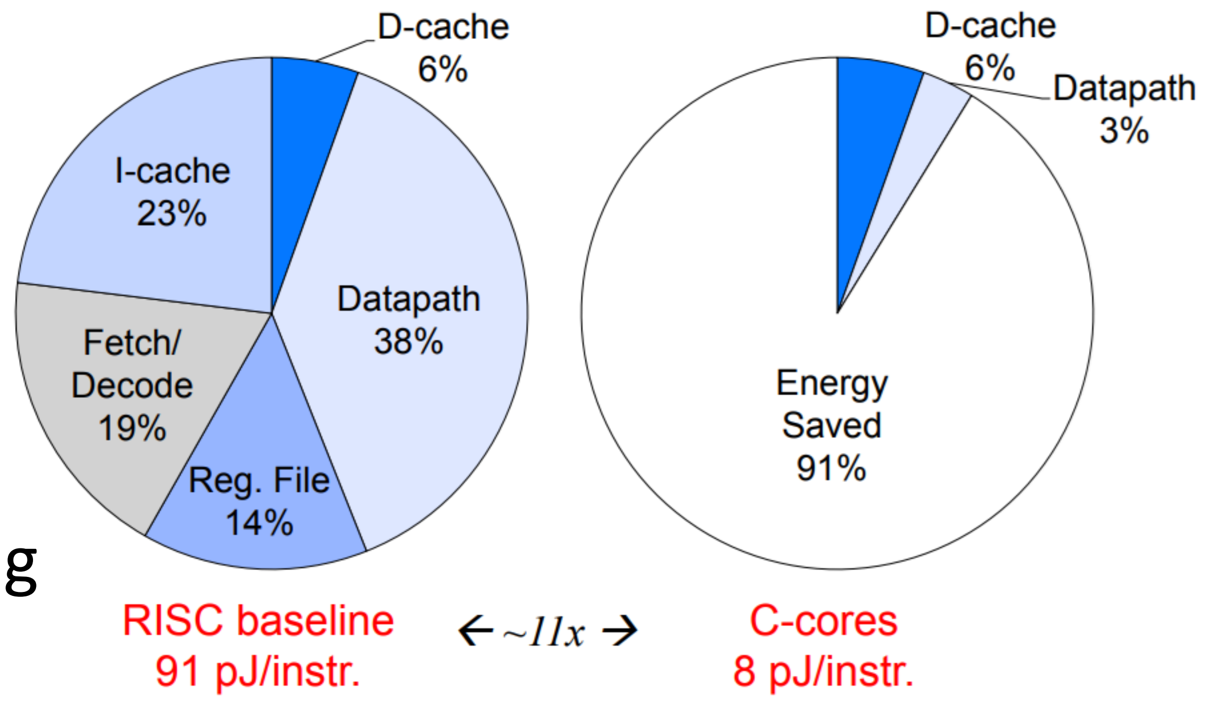
- Utilization Wall: With each successive process generation, the percentage of a chip that can actively switch drops exponentially due to power constraints.

- Experimental results
  - Replicated a small datapath
  - More "dark silicon" than active



The specialized method

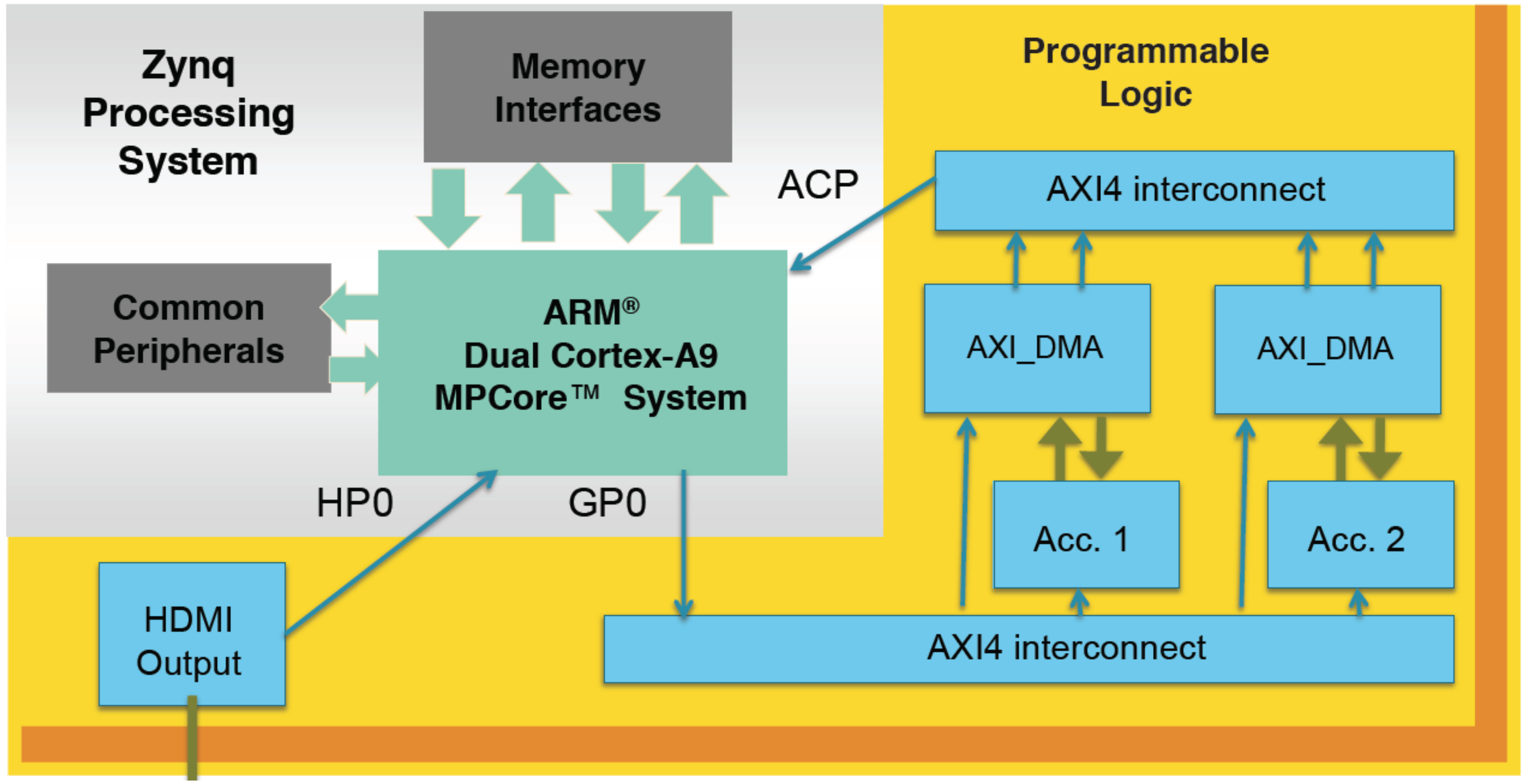
- Idea: Leverage dark silicon to "fight" the utilization wall
- Insights:
  - Power is now more expensive than area
  - Specialized logic can improve energy efficiency by 10-1000x
- Accelerator is the basic specializing method



Processor-Accelerator Interface

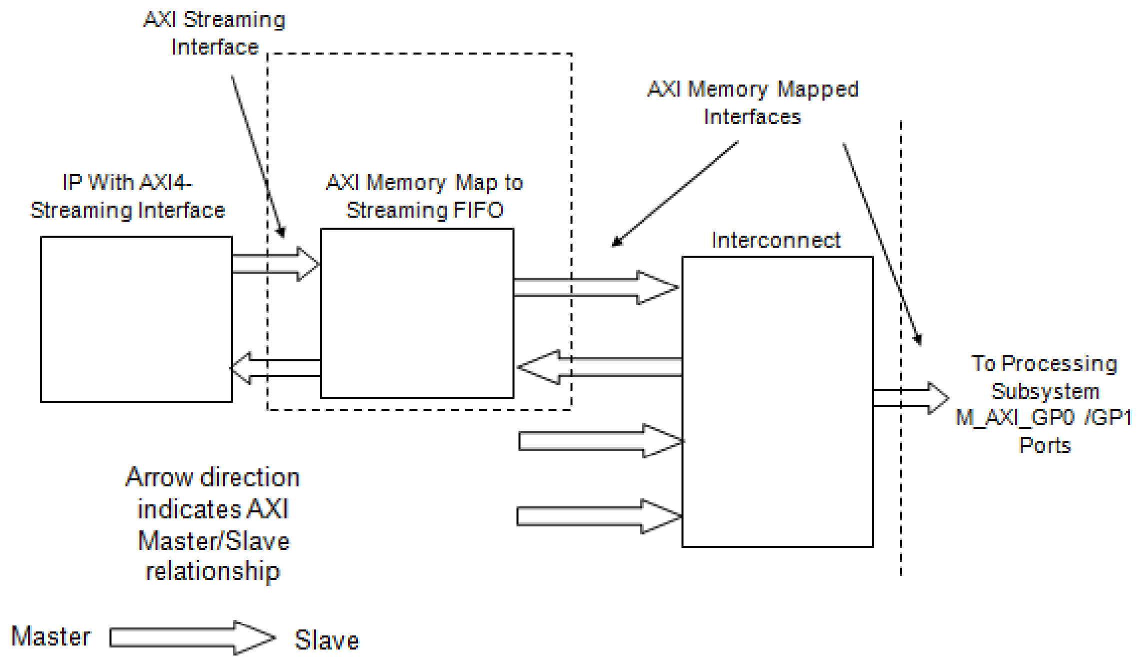
- Xilinx Zynq / Pynq platform
- Accelerator Coherency Port
  - Data coherency between accelerator/ cache/memory?

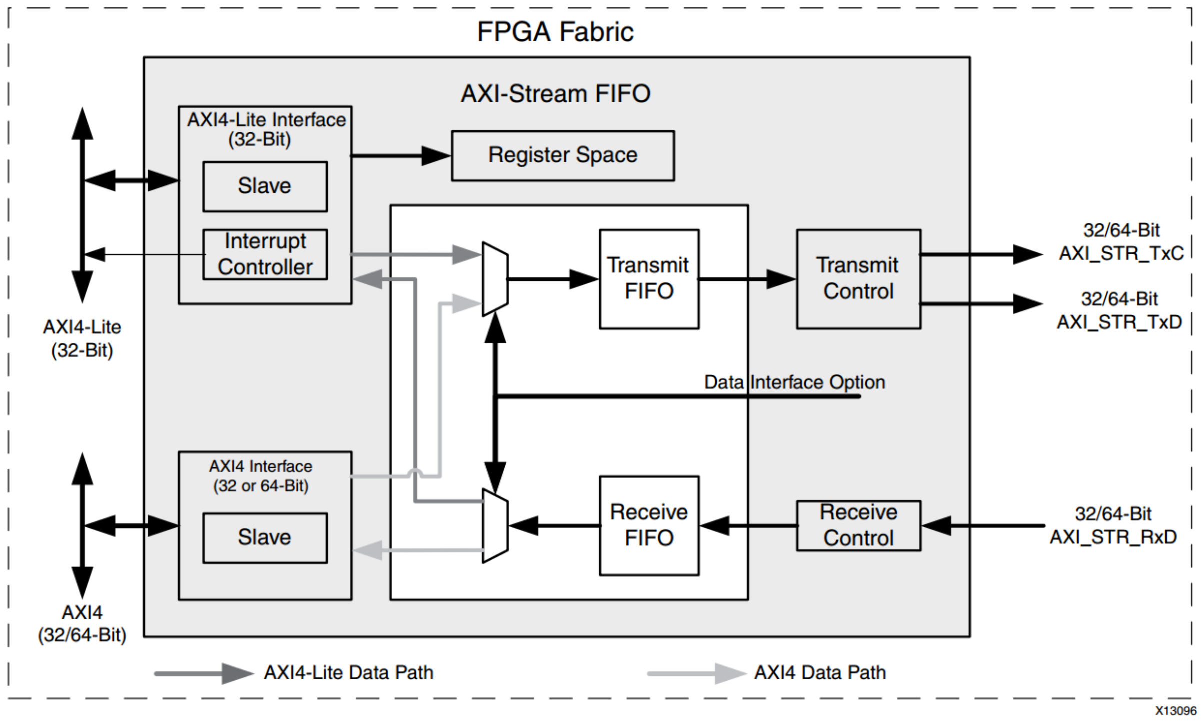
Pro: Low latency, high-bandwidth communication
Con: Complicated system architecture, Limited to data that fits in caches



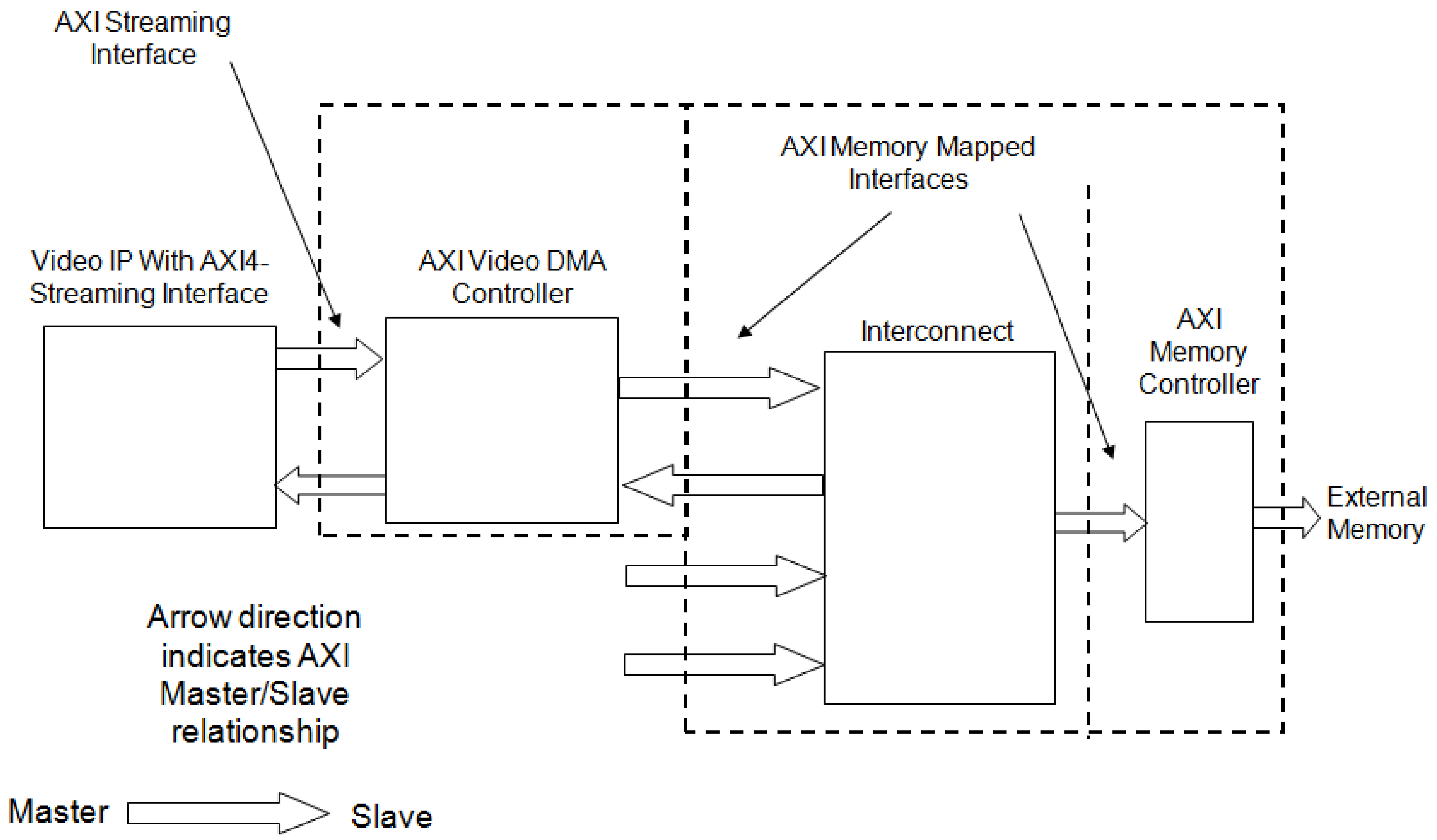
Interface : Streaming FIFO

- Three user-side AXI Stream interfaces: TX data, RX data, TX control


AXI Streaming Interface
AXI Memory Mapped Interfaces
IP With AXI4-Streaming Interface
AXI Memory Map to Streaming FIFO
Interconnect
To Processing Subsystem M_AXI_GP0 /GP1 Ports
Arrow direction indicates AXI Master/Slave relationship
Master
Slave

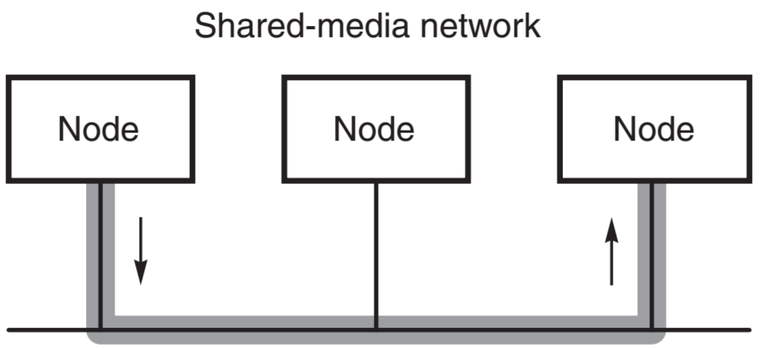


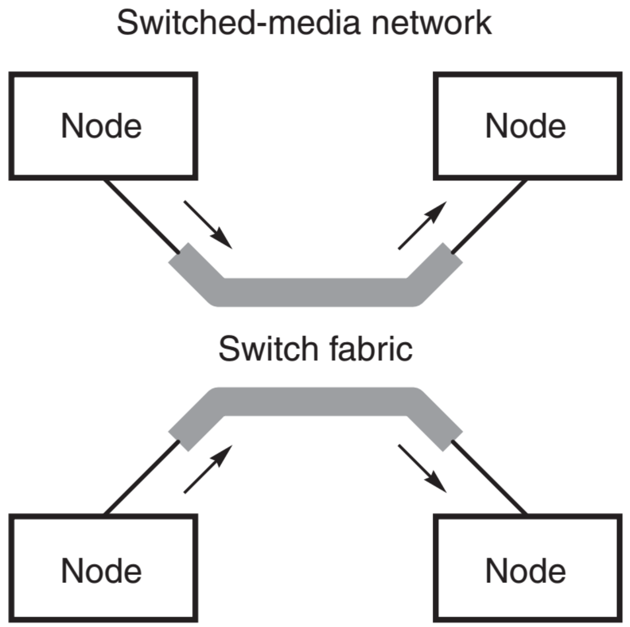
Interface : AXI DMA Controllwer


AXI Streaming Interface
AXI Memory Mapped Interfaces
Video IP With AXI4-Streaming Interface
AXI Video DMA Controller
Interconnect
AXI Memory Controller
External Memory
Arrow direction indicates AXI Master/Slave relationship
Master
Slave

More Advanced Networks

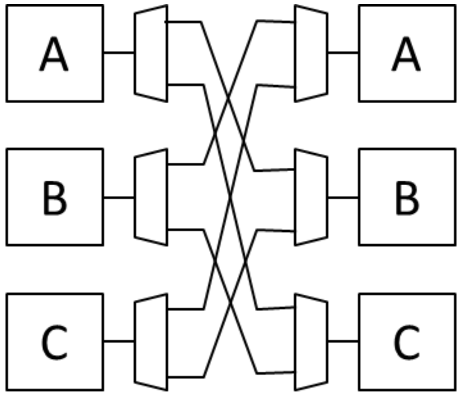
- Network-on-chip Connecting many cores
  - Shared media networks : Bus
  - Switch media networks



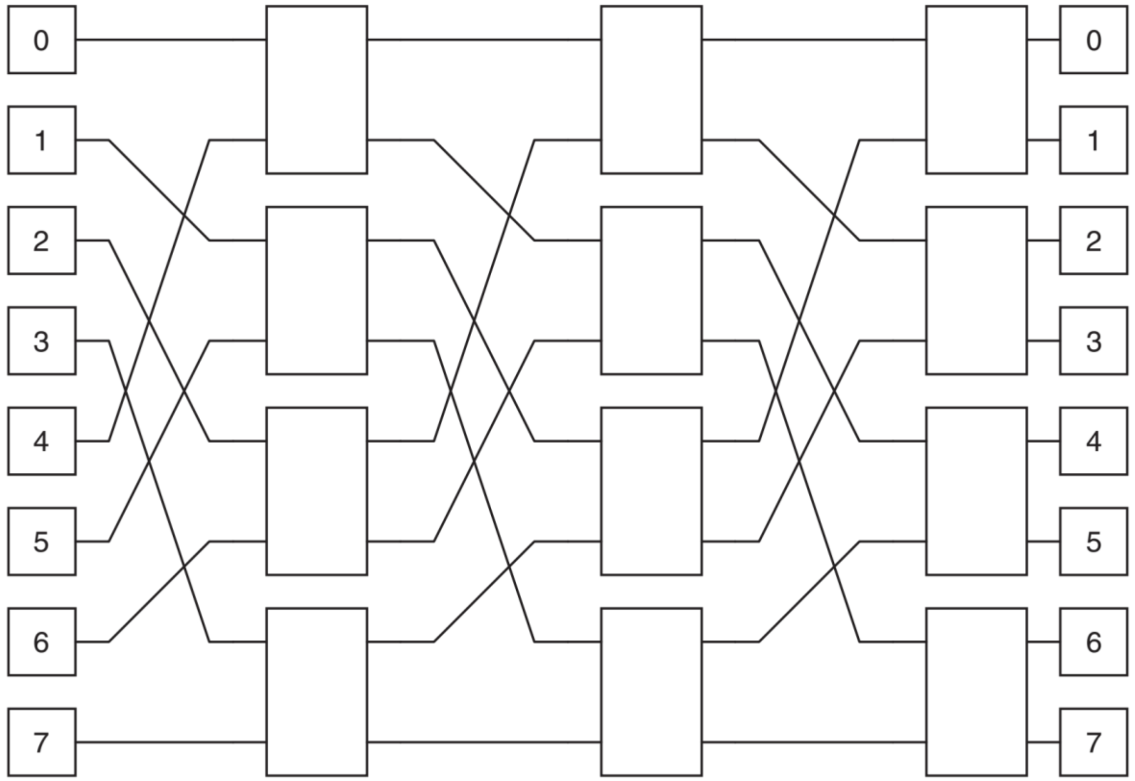


Crossbar On-Chip

- Busses evolved in era where wires were expensive and had to be shared
- Crossbar exploits density of on-chip wiring, allows multiple simultaneous transactions



Crossbar

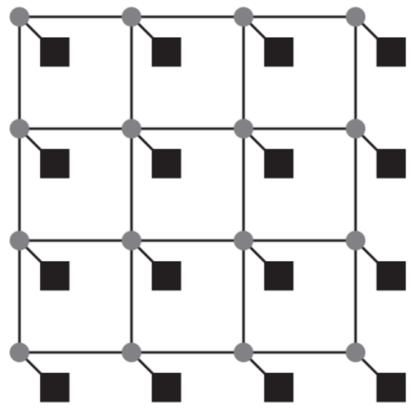


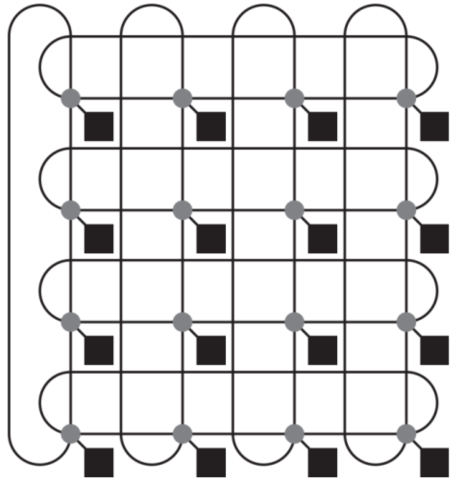
How many switches for N nodes cross bar?

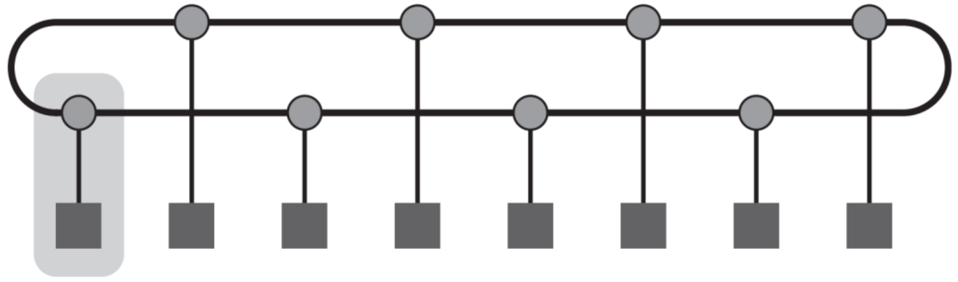
Distributed Ring/Mesh networks

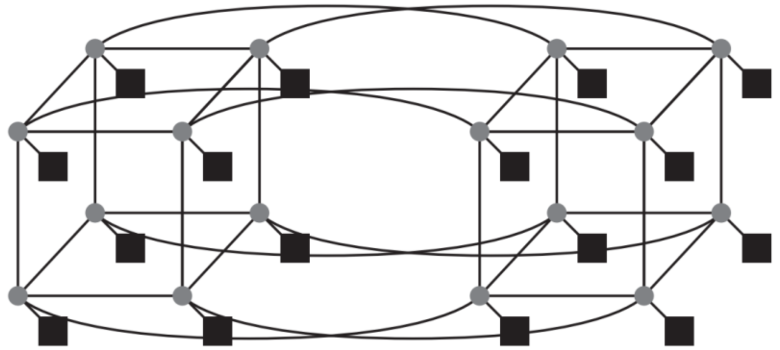
There are cases where it is convenient to more tightly integrate the end node devices with the network resources used to enable them to communicate.

Distribute the network switches among the end nodes


(A) 2D grid or mesh of 16 nodes

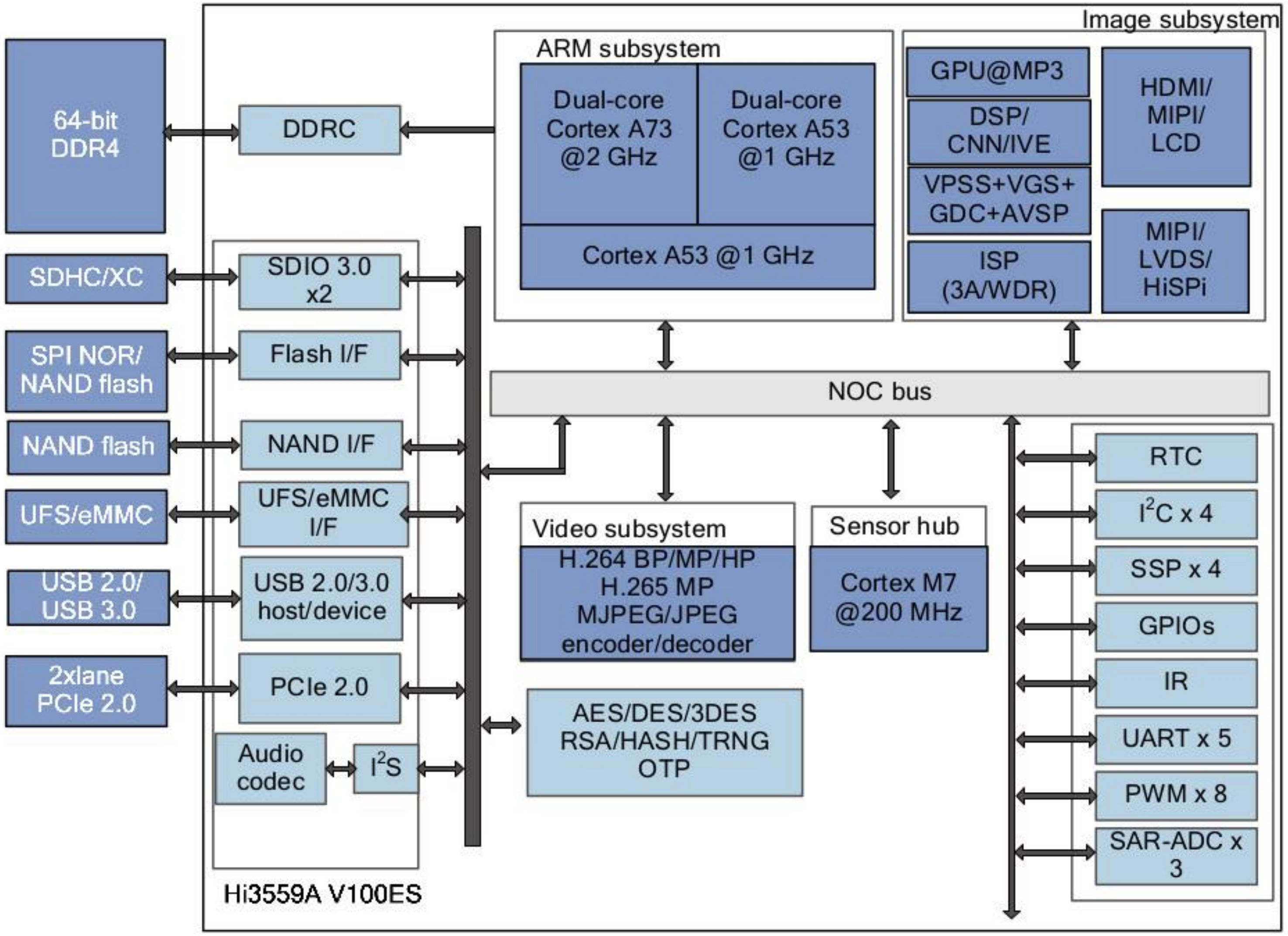

(B) 2D torus of 16 nodes




(C) Hypercube of 16 nodes ($16 = 2^4$ so $n = 4$)

Summary of the computer architecture

- Hisilicon H3559 for embedded video application


64-bit DDR4
DDRC
SDHC/XC
SDIO 3.0 x2
SPI NOR/ NAND flash
Flash I/F
NAND flash
NAND I/F
UFS/eMMC
UFS/eMMC I/F
USB 2.0/ USB 3.0
USB 2.0/3.0 host/device
2xlane PCIe 2.0
PCIe 2.0
Audio codec
I2S
Hi3559A V100ES
ARM subsystem
Dual-core Cortex A73 @2 GHz
Dual-core Cortex A53 @1 GHz
Cortex A53 @1 GHz
Image subsystem
GPU@MP3
DSP/ CNN/IVE
VPSS+VGS+ GDC+AVSP
ISP (3A/WDR)
HDMI/ MIPI/ LCD
MIPI/ LVDS/ HiSPi
NOC bus
Video subsystem
H.264 BP/MP/HP H.265 MP MJPEG/JPEG encoder/decoder
Sensor hub
Cortex M7 @200 MHz
AES/DES/3DES RSA/HASH/TRNG OTP
RTC
I2C x 4
SSP x 4
GPIOs
IR
UART x 5
PWM x 8
SAR-ADC x 3