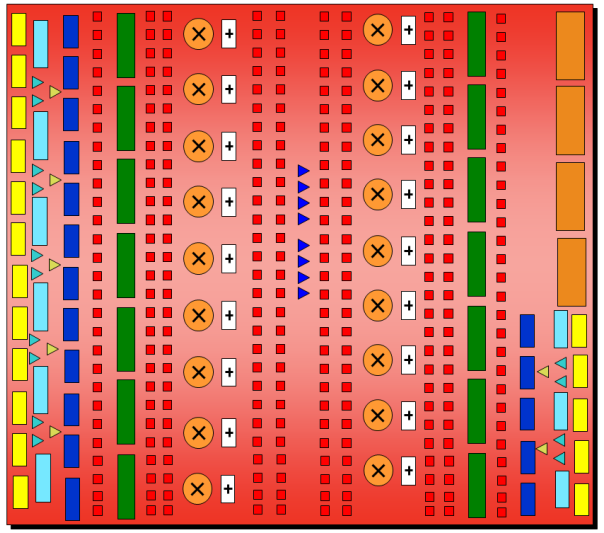
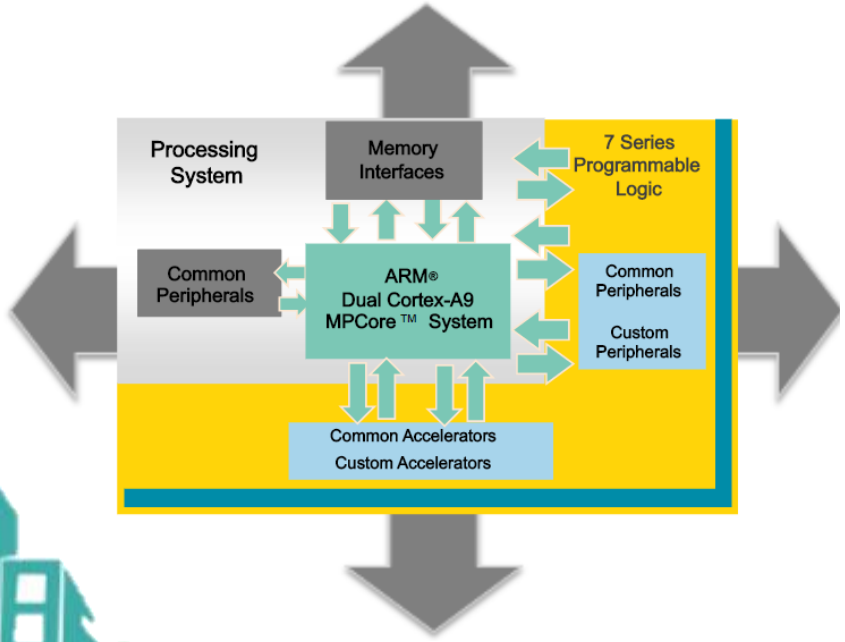
DEPARTMENT OF ELECTRONICS & COMMUNICATIONS ENGINEERING

A2A
Algorithms to Architecture

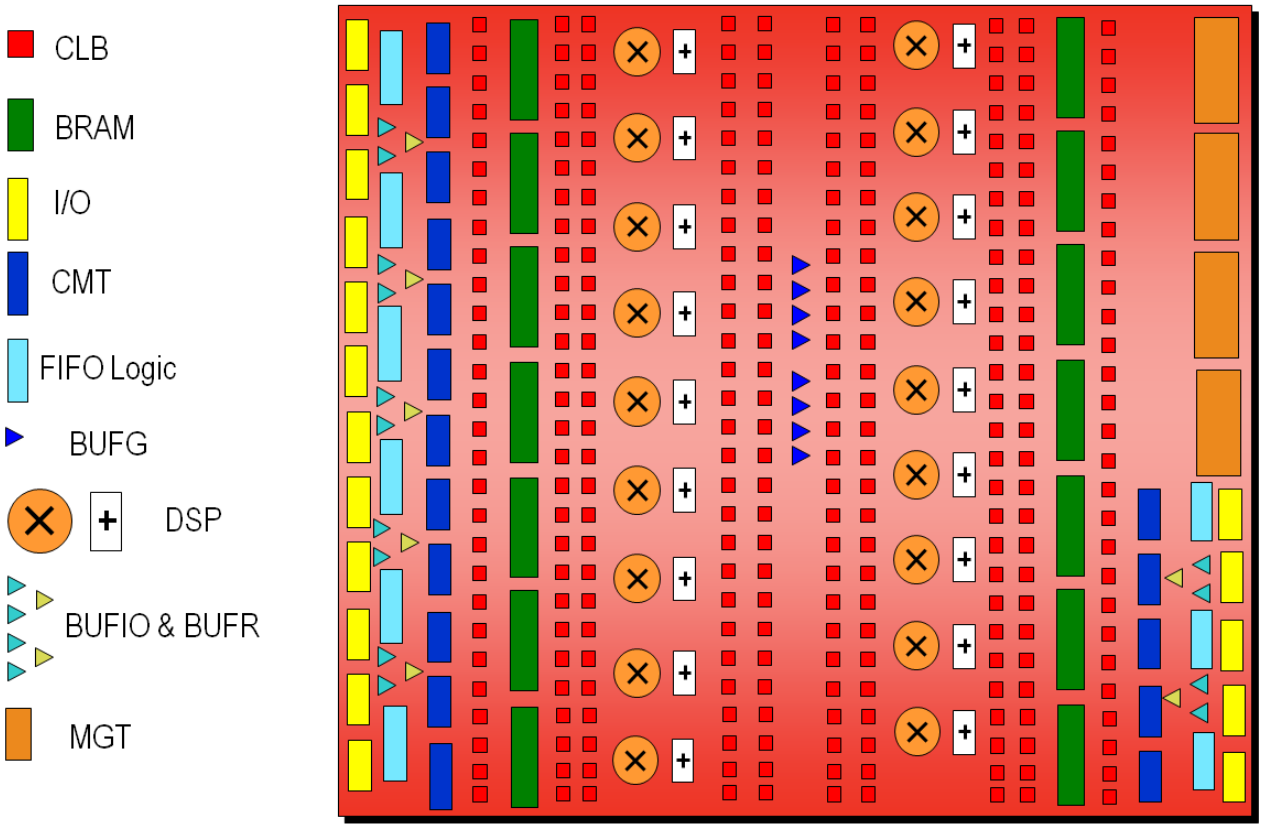
ECE 270: Embedded Logic Design

SDK
Software Development Kit
VIVADO™



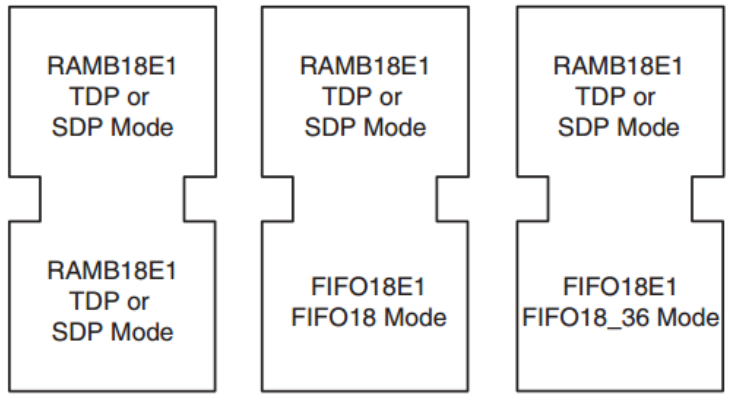


Block RAM

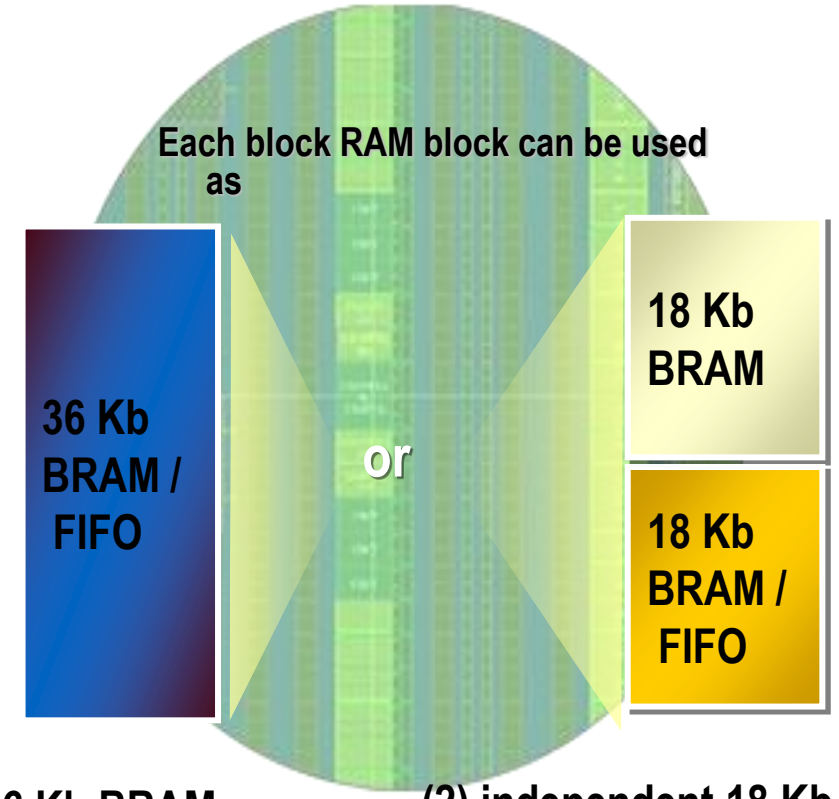


Block RAM

- Each BRAM can be segmented as:
  - 36 Kb BRAM
  - 36 Kb FIFO
  - Independent 18 Kb BRAMs
  - 18 Kb FIFO and 18 Kb BRAM



Legal Block RAM and FIFO Combinations

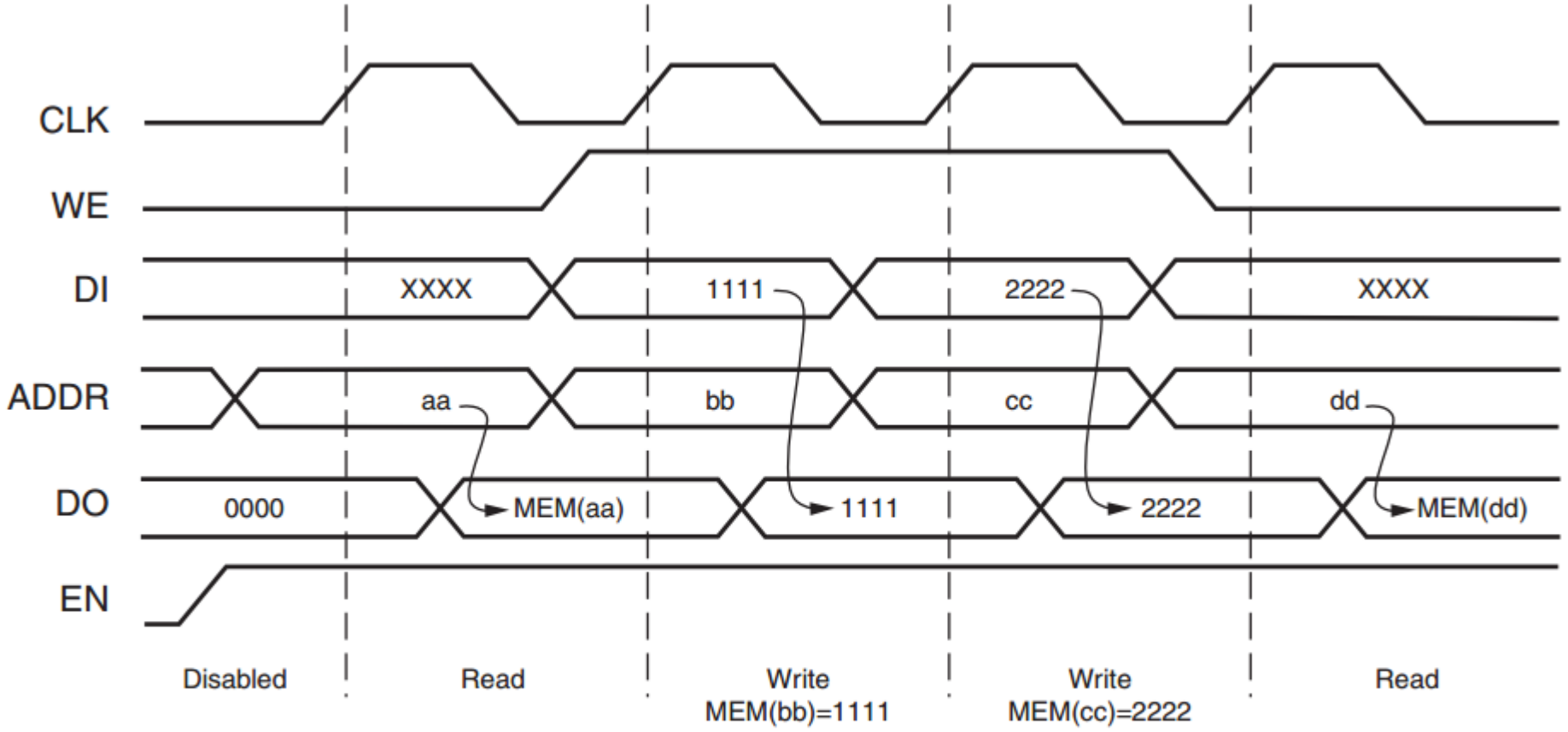


(1) 36 Kb BRAM OR (1) 36 Kb or FIFO

(2) independent 18 Kb block RAMs OR (1) 18 Kb FIFO + (1) 18 Kb block RAM

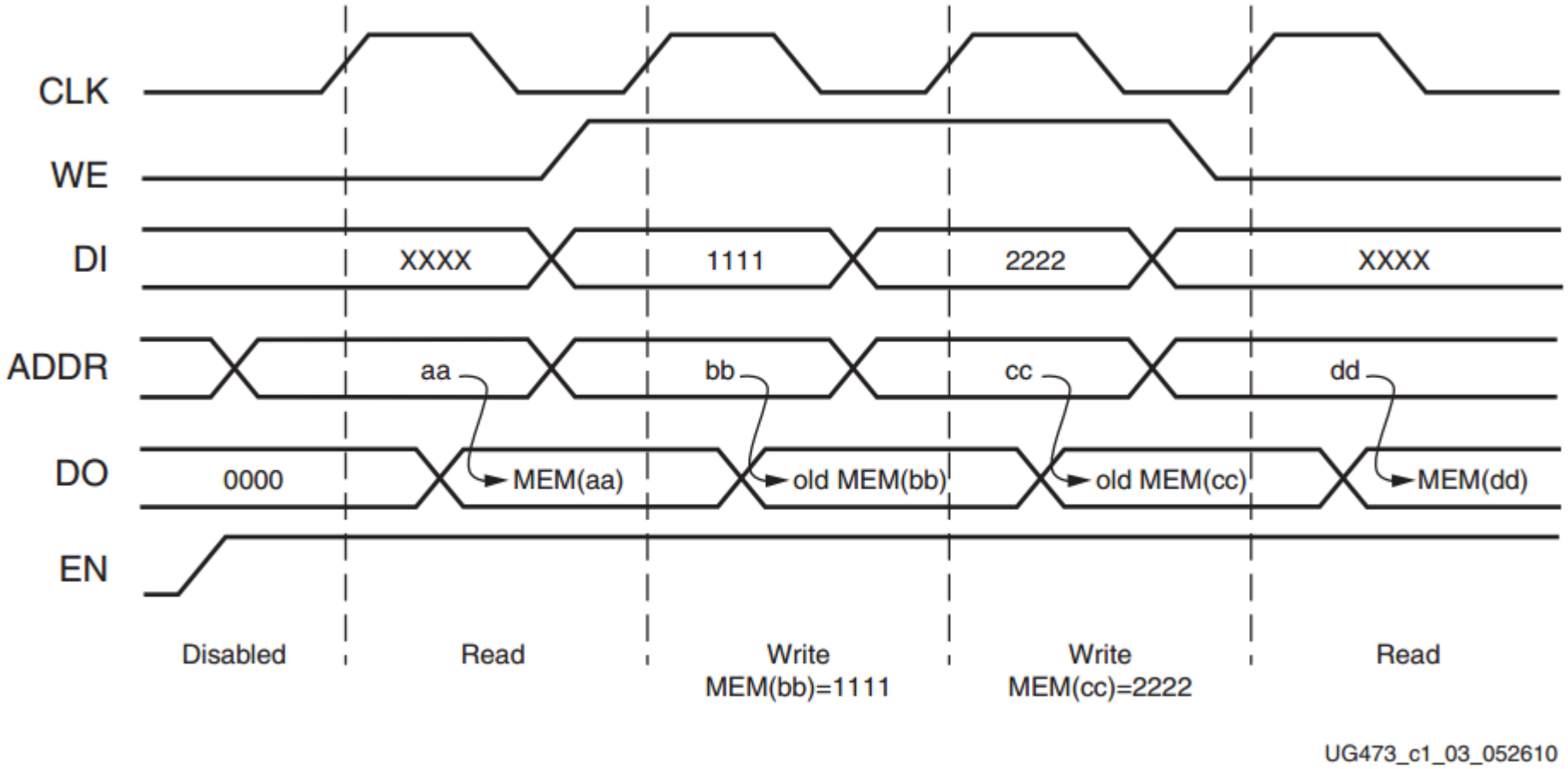
These waveforms correspond to latch mode when the optional output pipeline register is not used.

Block RAM: WRITE_FIRST



WRITE_FIRST Mode Waveforms

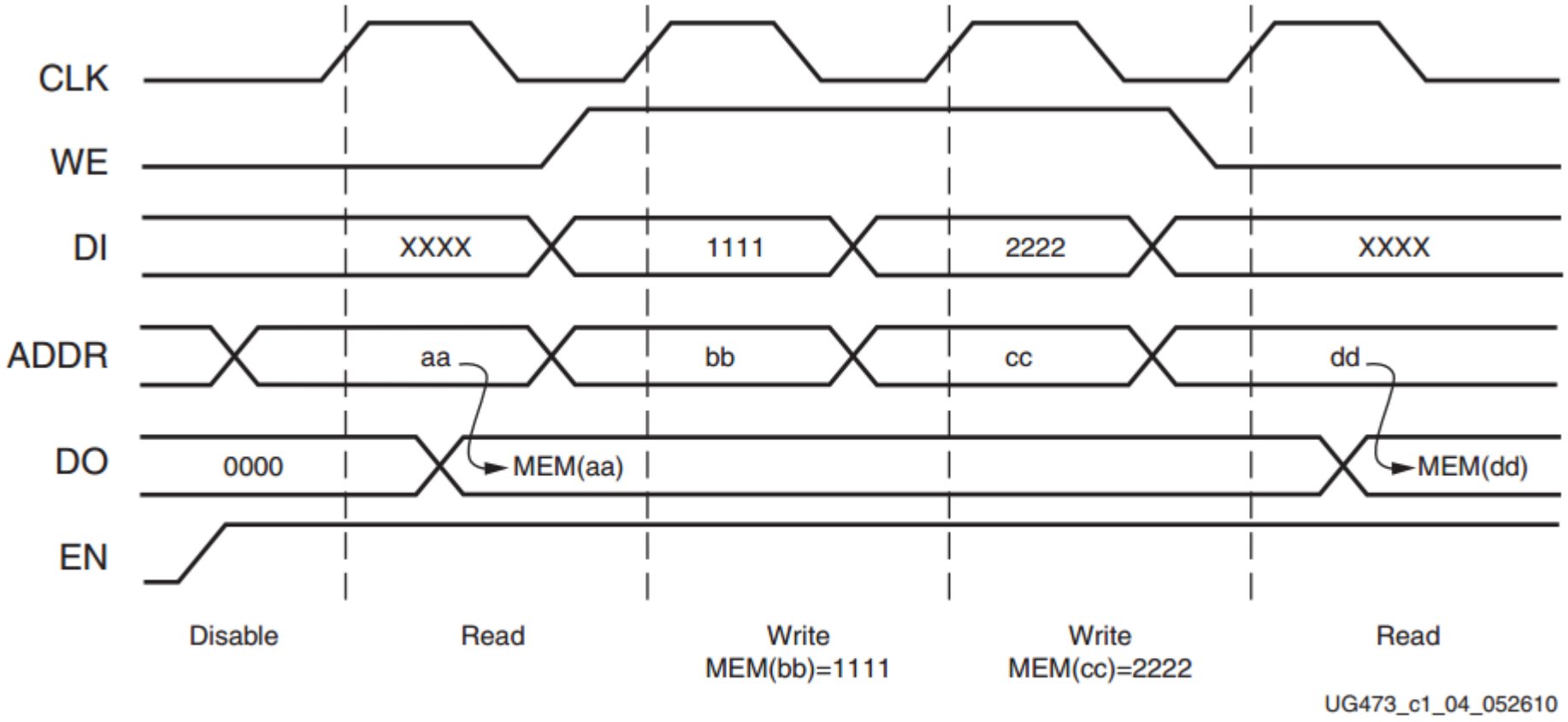
Block RAM: READ_FIRST



READ_FIRST Mode Waveforms

Block RAM: NO_CHANGE

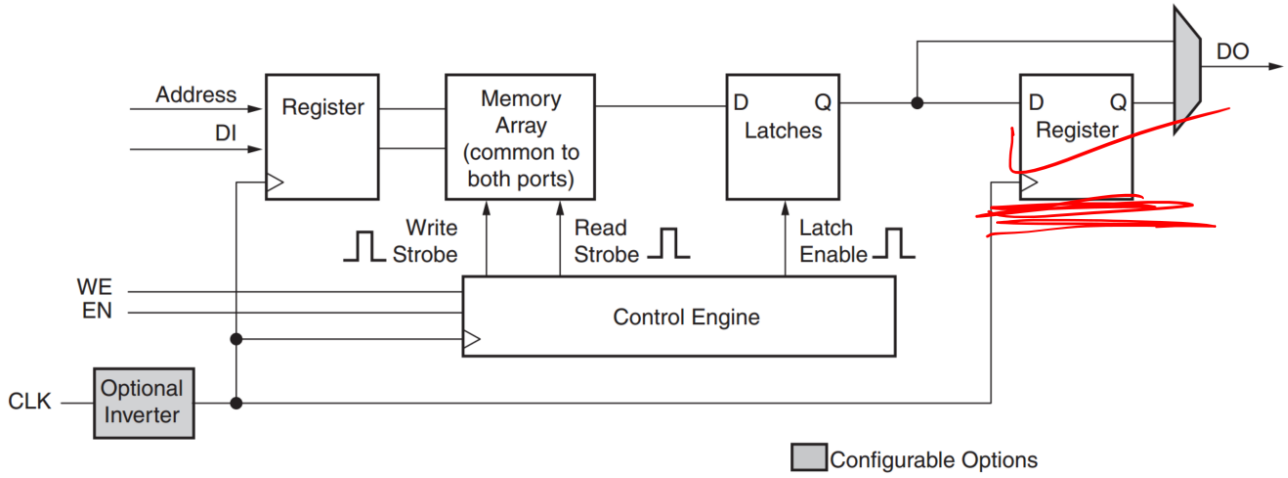
Most power efficient



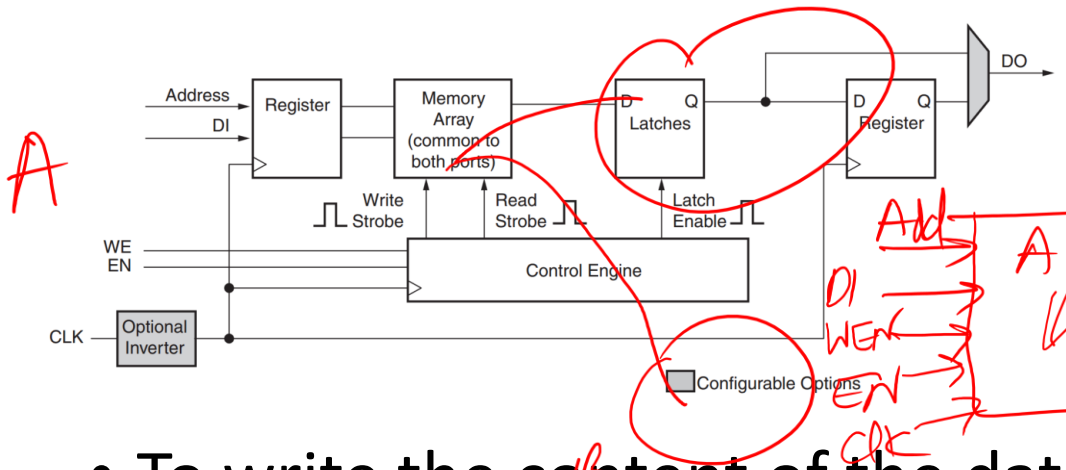
NO_CHANGE Mode Waveforms

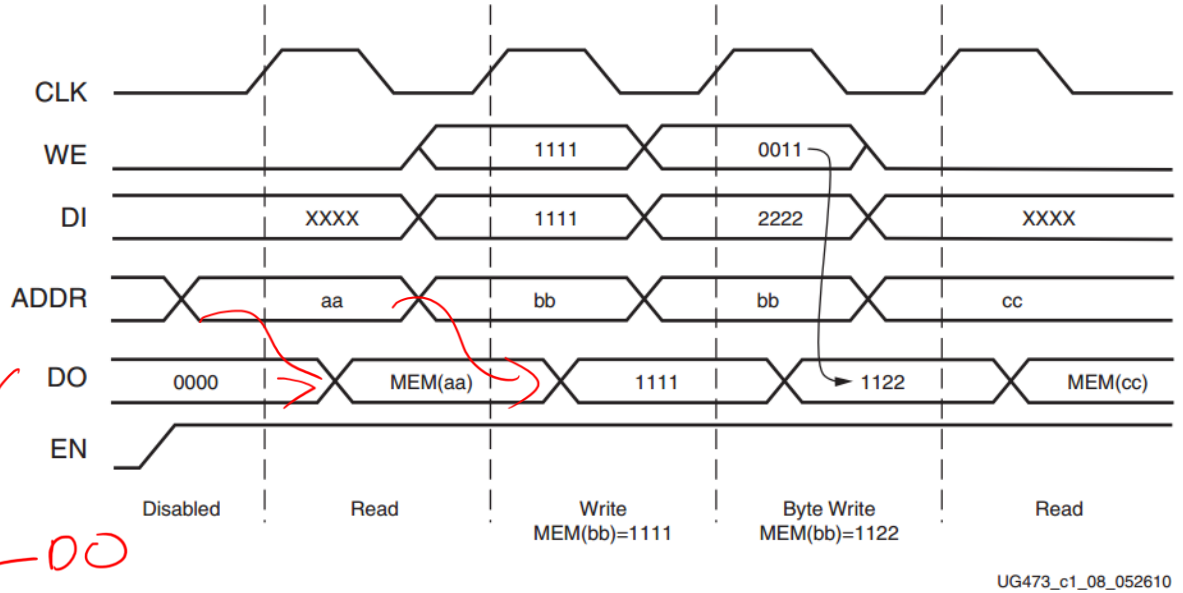
Block RAM

- Fully synchronous operation (Nothing happens without a clock)
- Each memory access, read or write, is controlled by the clock.
- All inputs, data, address, clock enables, and write enables are registered.
- The input address is always clocked, retaining data until the next operation.
- An optional output data pipeline register allows higher clock rates at the cost of an extra cycle of latency.



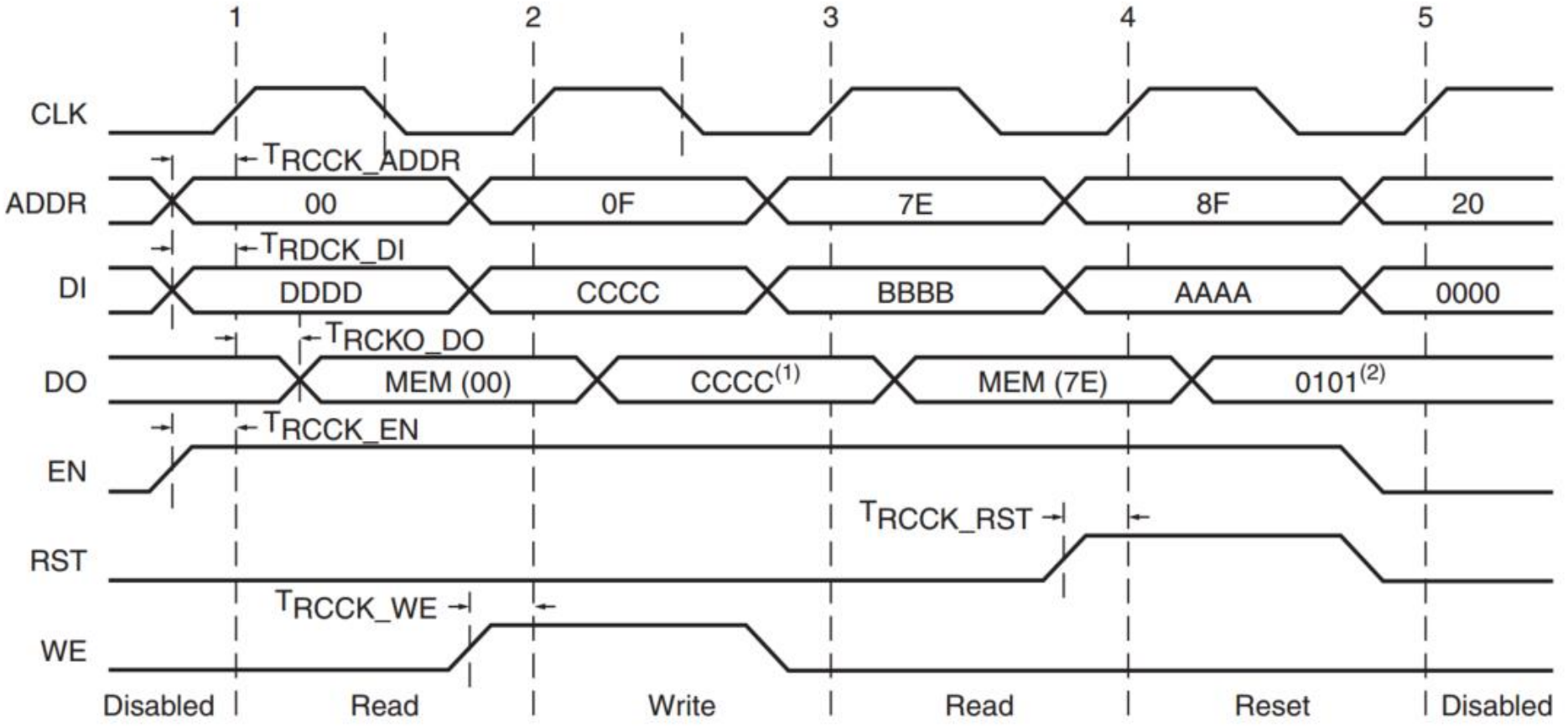
Block RAM





- To write the content of the data input bus into the addressed memory location, both EN and WE must be active.
- The output latches are loaded or not loaded according to the write configuration (WRITE_FIRST, READ_FIRST, NO_CHANGE).
- When WE is inactive and EN is active, a read operation occurs, and the contents of the memory cells referenced by the address bus appear on the data-out bus, regardless of the write mode attribute.

Ignore ☺

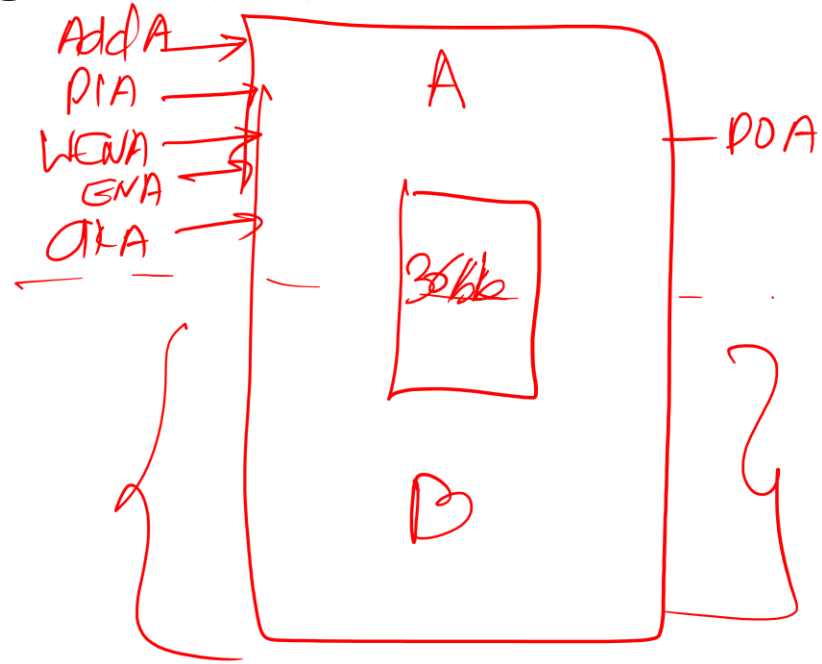


Note 1: Write Mode = WRITE_FIRST

Note 2: SRVAL = 0101

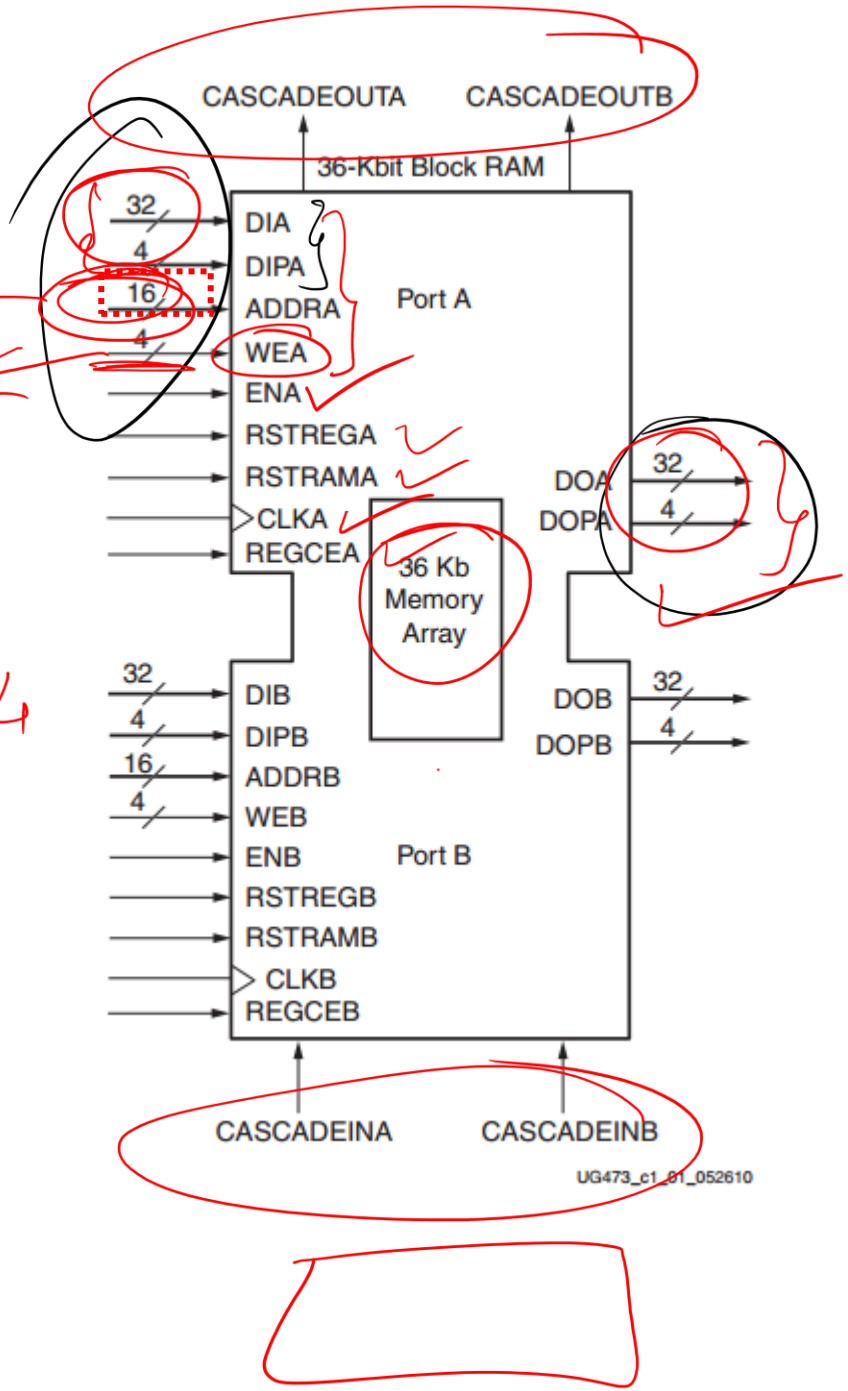
UG473_c1_15_052610

Dual Port Block RAM



Dual-Port Block RAM

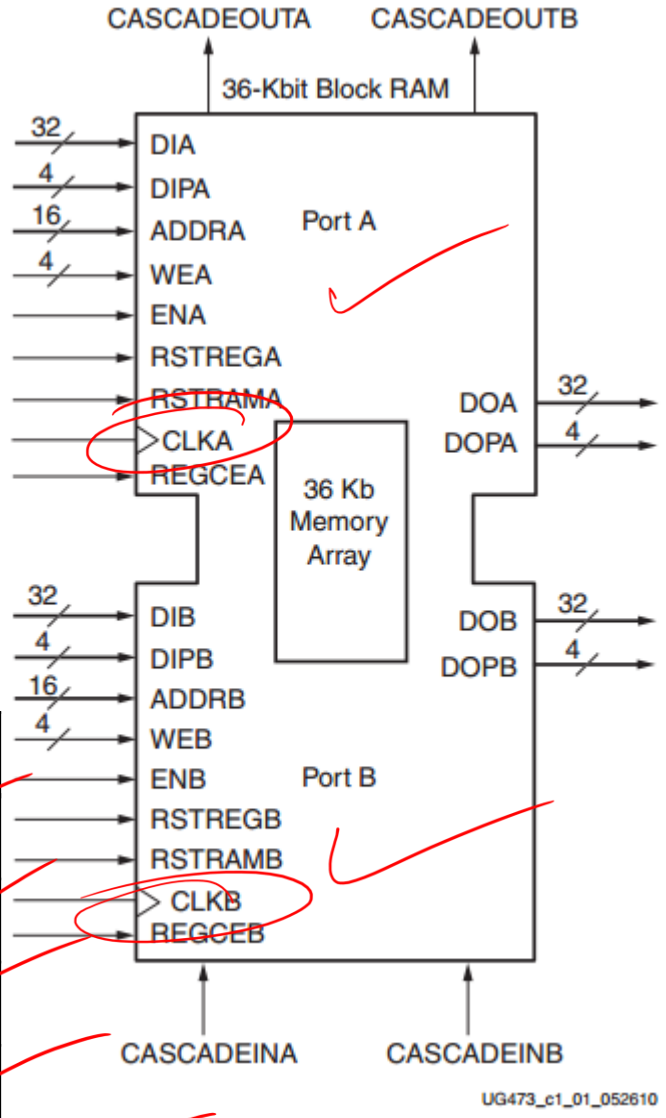
Port Function	Description
DI[A | B]	Data input bus.
DIP[A | B](1)	Data input parity bus. Can be used for additional data inputs.
ADDR[A | B]	Address bus.
WE[A | B]	Byte-wide write enable.
EN[A | B]	When inactive no data is written to the block RAM and the output bus remains in its previous state.
RSTREG[A | B]	Synchronous Set/Reset the output registers (DO_REG = 1). The RSTREG_PRIORITY attribute determines the priority over REGCE.
RSTRAM[A | B]	Synchronous Set/Reset the output data latches.
CLK[A | B]	Clock input.
DO[A | B]	Data output bus.
DOP[A | B](1)	Data output parity bus. Can be used for additional data outputs.
REGCE[A | B]	Output Register clock enable.
CASCADEIN[A | B]	Cascade input for 64K x 1 mode.
CASCADEOUT[A | B]	Cascade output for 64K x 1 mode.



Dual-Port Block RAM

- Two separate read/write ports
  - Each port has separate clock, address, data in, data out, write enable…
    - Clocks can be asynchronous to each other
  - The two ports can have different widths
  - The two ports can have different write modes
- Each block RAM can be divided into two completely independent 18 Kb block RAMs

Configurations
32K x 1
16K x 2
8K x 4
4K x 8(or 9)
2K x 16(or 18)
1K x 32(or 36)

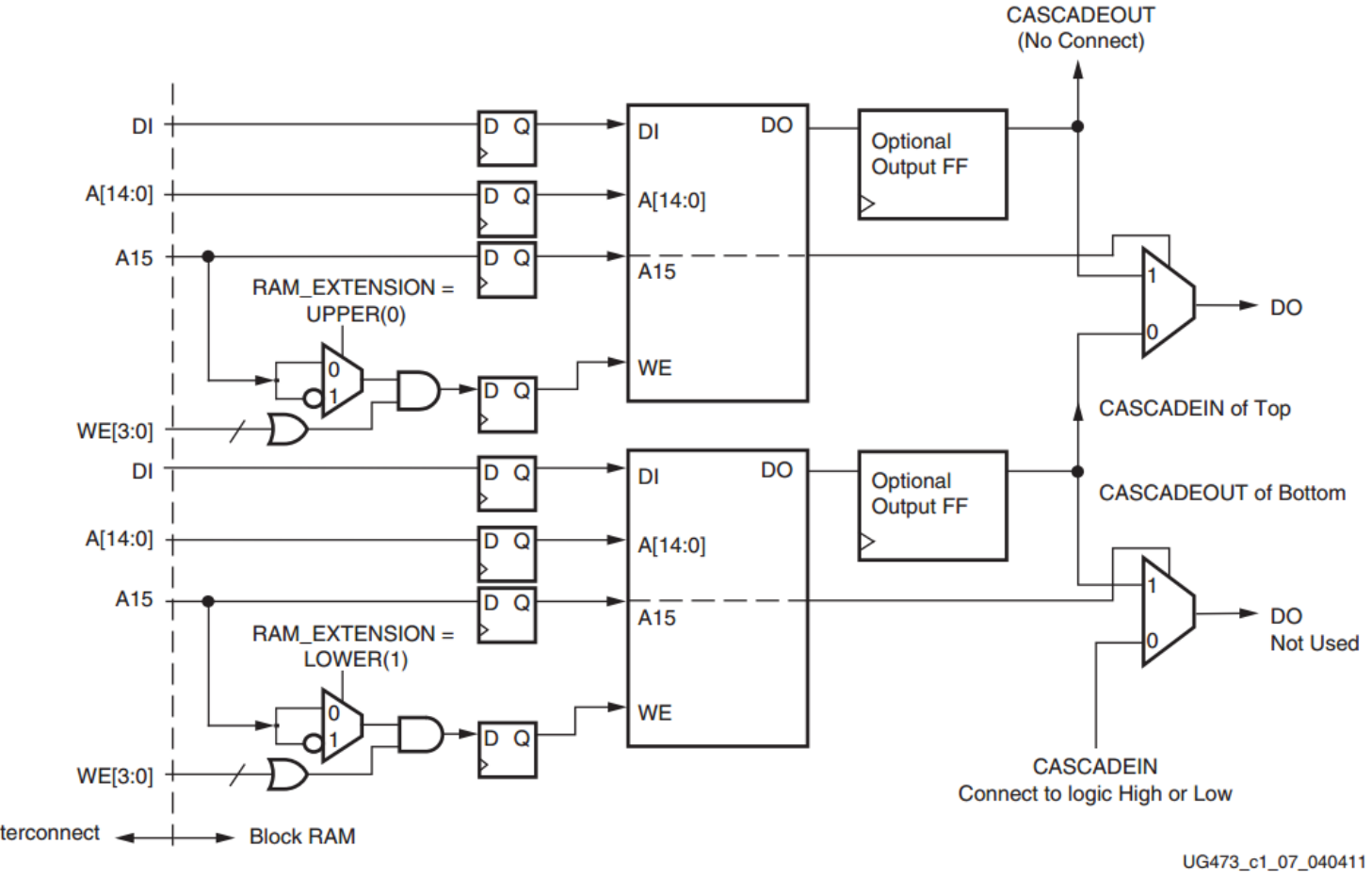


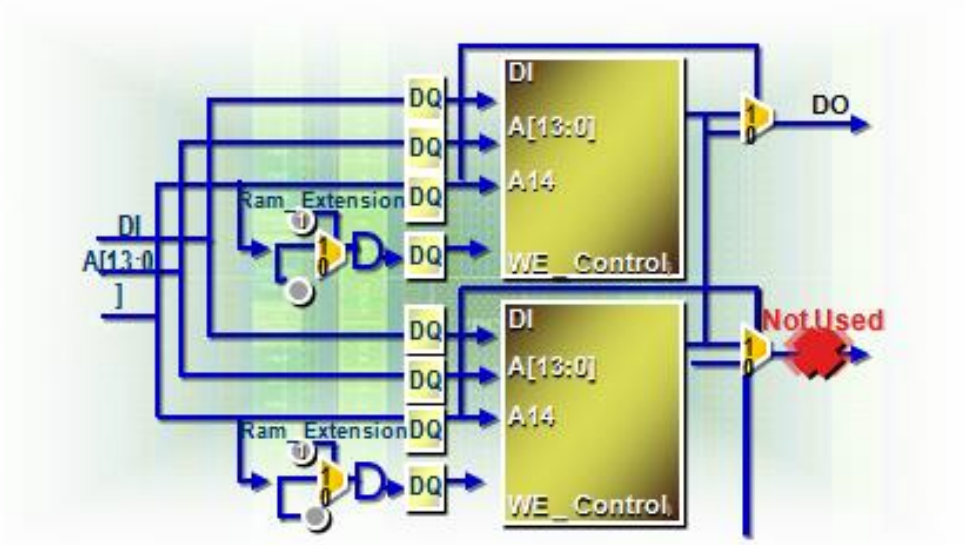
Dual Port BRAM

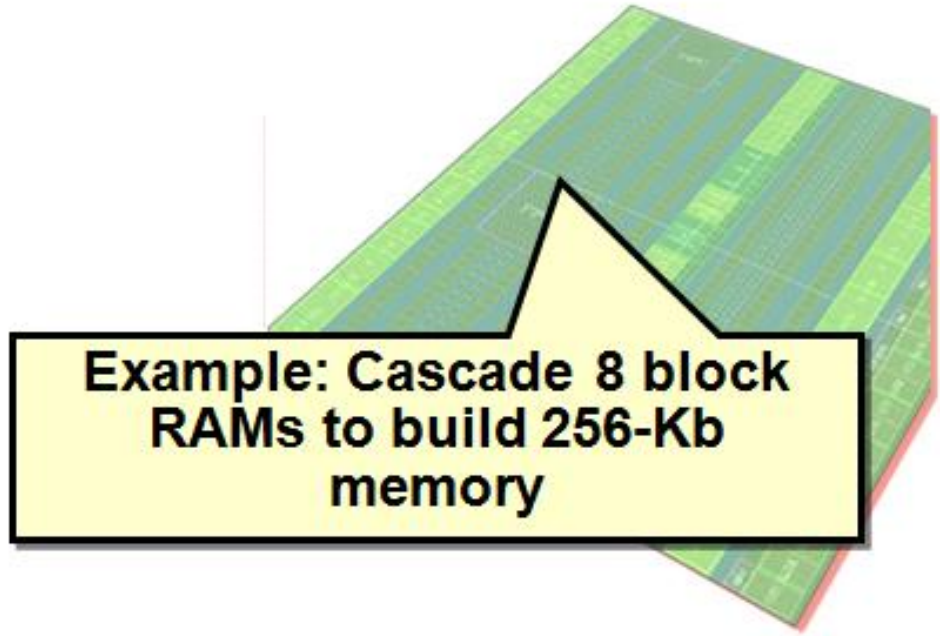
Block RAM Cascading

- Built-in cascade logic for 64Kx1
  - Cascade *two* vertically adjacent 32Kx1 block RAMs without using external CLB logic or compromising performance

Block RAM Cascading

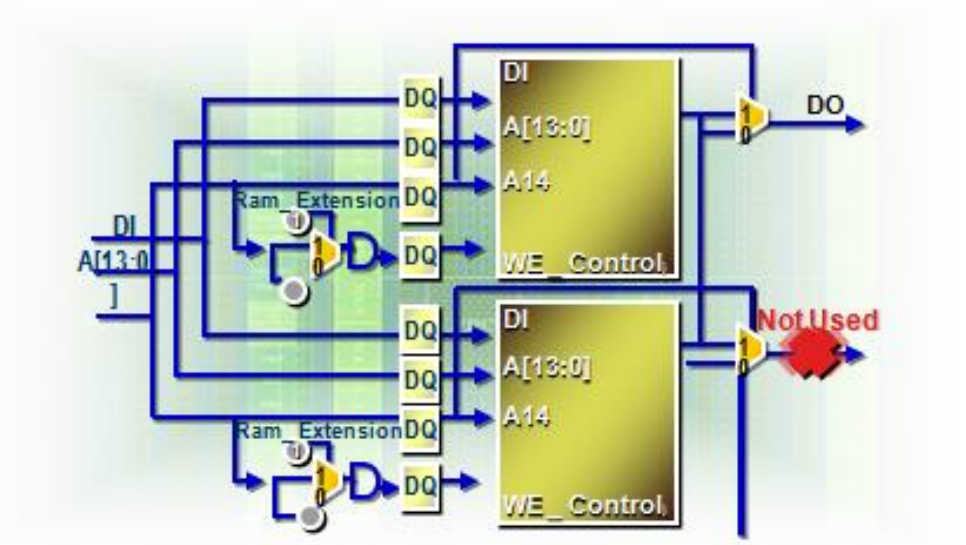


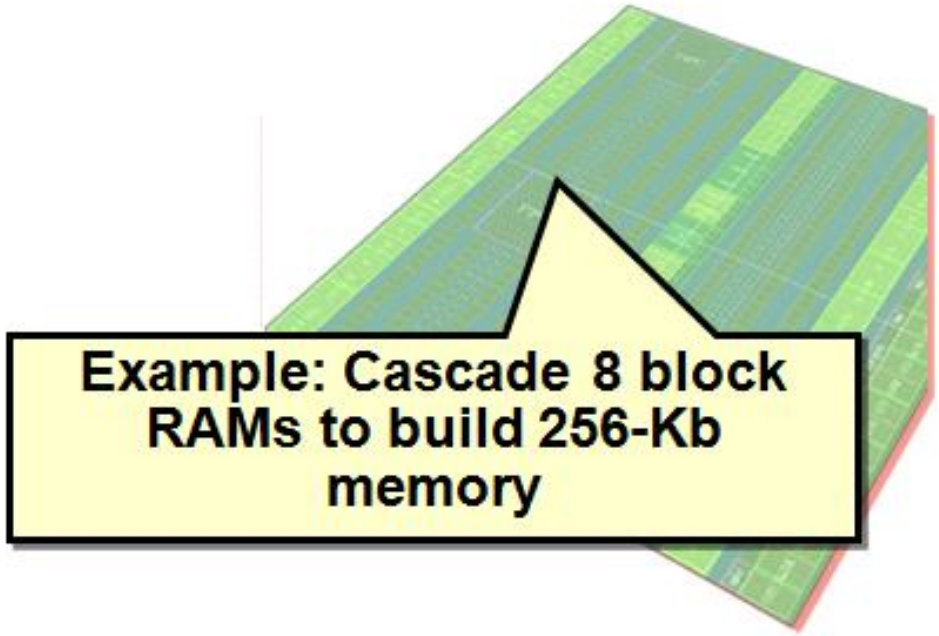




Block RAM Cascading

- Built-in cascade logic for 64Kx1
  - Cascade *two* vertically adjacent 32Kx1 block RAMs without using external CLB logic or compromising performance
  - Saves resources and improves speed of larger memories
- Cascade option for larger arrays
  - 128Kb, 256Kb, 512Kb, 1 Mb, …
  - Using external CLB logic for depth expansion
    - Not quite as fast as cascaded block RAMs
  - Width expansion uses parallel block RAMs

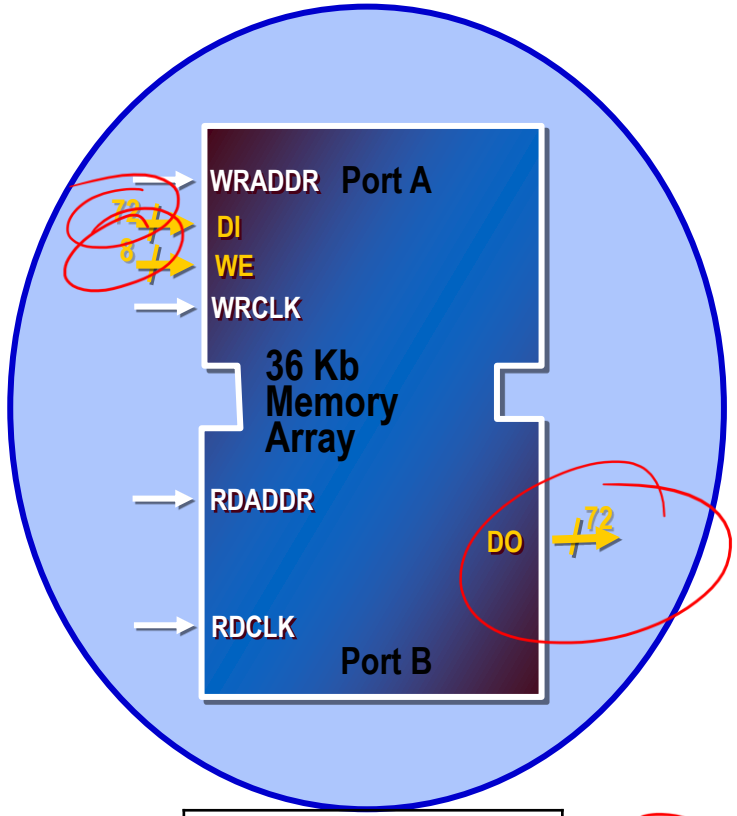




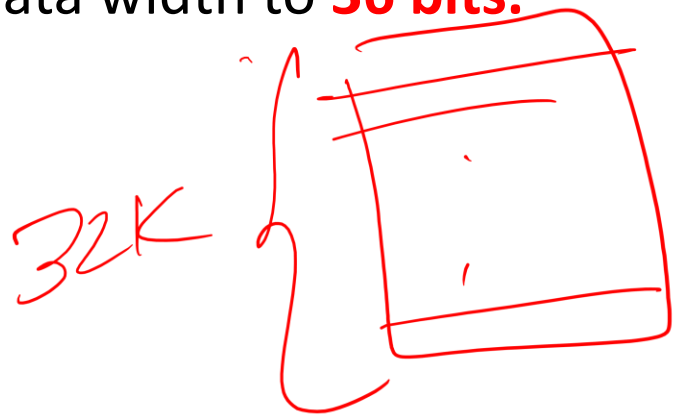
Simple Dual Port Block RAM

Simple Dual Port Block RAM

- Simple dual-port mode is defined as having one read-only port and one write-only port with independent clocks.
- Each 36Kb block RAM can be set to simple dual-port (SDP) mode, doubling data width of the block RAM to **72 bits.**
- The 18Kb block RAM can also be set to simple dual-port mode, doubling data width to **36 bits.**

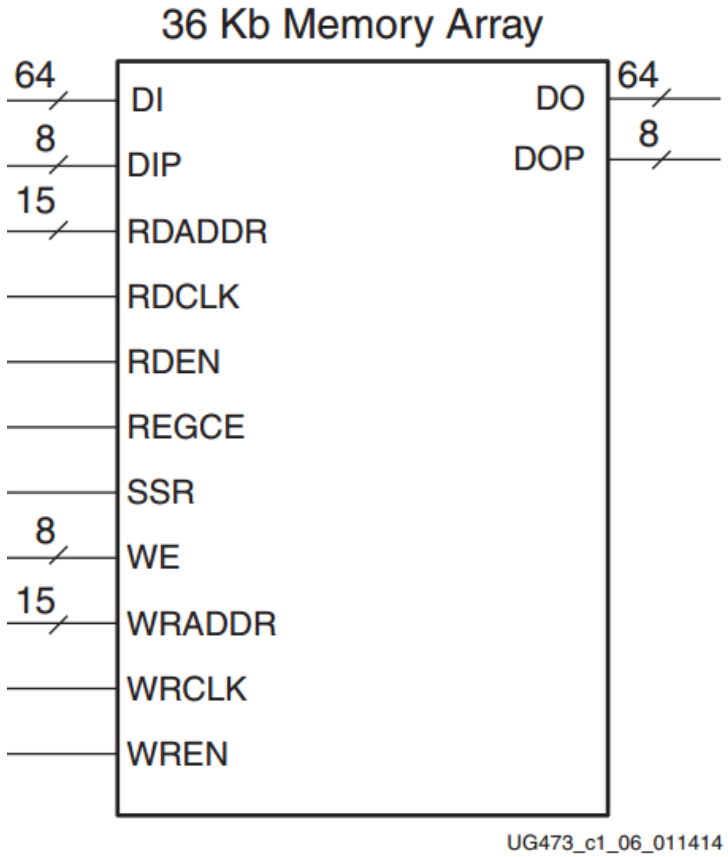


Configurations
32K x 1
16K x 2
8K x 4
4K x 8(or 9)
2K x 16(or 18)
1K x 32(or 36)
512 x 64(or 72)



Simple Dual-Port Block RAM

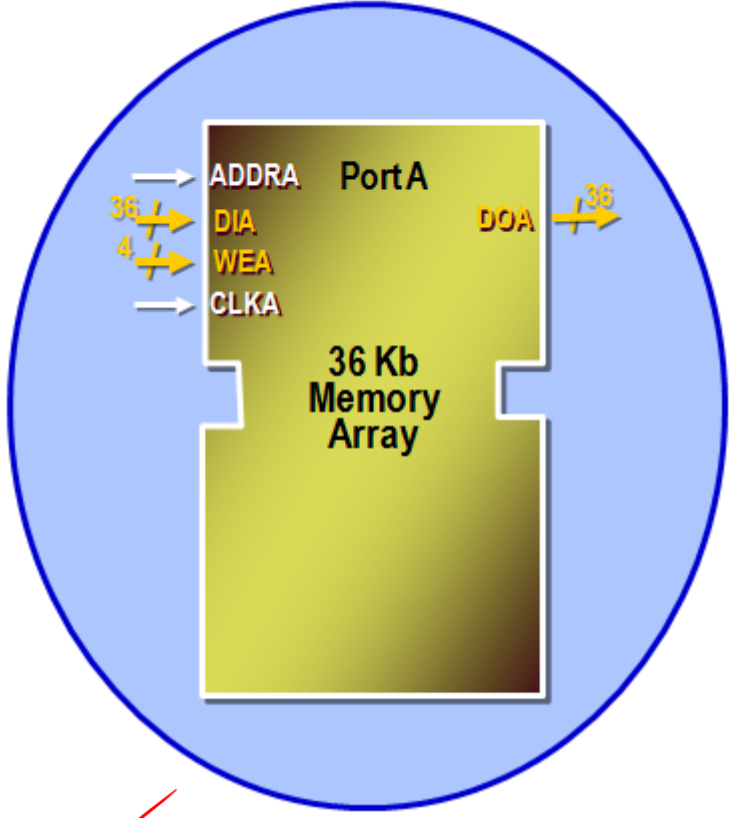
Port Function	Description
DO	Data Output Bus
DOP	Data Output Parity Bus
DI	Data Input Bus
DIP	Data Input Parity Bus
RDADDR	Read Data Address Bus
RDCLK	Read Data Clock
RDEN	Read Port Enable
REGCE	Output Register Clock Enable
SBITERR	Single Bit Error Status
DBITERR	Double Bit Error Status
ECCPARITY	ECC Encoder Output Bus
SSR	Synchronous Set or Reset of Output Registers or Latches
WE	Byte-wide Write Enable
WRADDR	Write Data Address Bus



RAMB36 in the Simple Dual-Port Data Flow

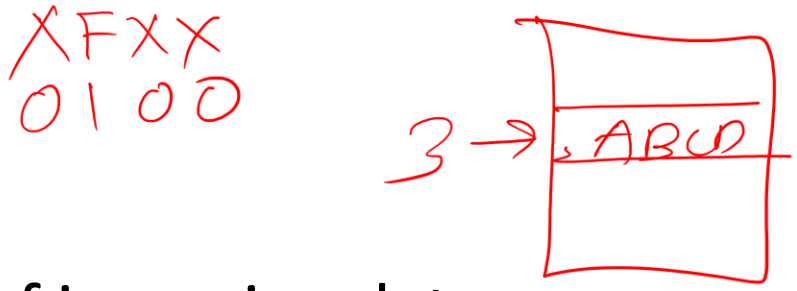
Single Port Block RAM

Single-Port Block RAM



- Single read/write port
  - Clock: CLKA, Address: ADDRA, Write enable: WEA
  - Write data: DIA, Read data: DOA
- 36-kbit configurations
  - 32k x 1, 16k x 2, 8k x 4, 4k x 9, 2k x 18, 1k x 36
- 18-kbit configurations*: 16k x 1, 8k x 2, 4k x 4, 2k x 9, 1k x 18
- The parity bits are only available for the x9, x18, and x36 port widths. The parity bits should not be used when the read width is x1, x2, or x4

Byte-Wide Write Enable



- Allows writing eight-bit (one byte) portions of incoming data.
- If configured in READ_FIRST mode, the DO bus shows the previous content of the whole addressed word.
- In WRITE_FIRST mode, DO shows a combination of the newly written enabled byte(s), and the initial memory contents of the unwritten bytes.
- There are four independent byte-wide write enable inputs to the 36Kb true dual-port RAM.
- There are eight independent byte-wide write enable inputs to block RAM in simple dual-port mode
- This feature is useful when using block RAM to interface with a microprocessor.

RAM HDL Coding Techniques

- Data is written synchronously into the RAM for both types.
- The primary difference between distributed RAM and dedicated block RAM lies in the way data is read from the RAM

Distributed RAM versus Dedicated Block RAM

Action	Distributed RAM	Dedicated Block RAM
Write	Synchronous	Synchronous
Read	Asynchronous	Synchronous

- Single-Port block RAM with read first mode

```
module rams_          : (clk, en, we, rst, addr, di, dout);
input clk;
input en;
input we;
input rst;
input [9:0] addr;
input [15:0] di;
output [15:0] dout;

reg [15:0] ram [1023:0];
reg [15:0] dout;

always @(posedge clk)
begin
  if (en) //optional enable
    begin
      if (we) //write enable
        ram[addr] <= di;
      if (rst) //optional reset
        dout <= 0;
      else
        dout <= ram[addr];
    end
end

endmodule
```

- Single-Port block RAM with write first mode

```
module rams_          (clk, we, en, addr, di, dout);
input clk;
input we;
input en;
input [9:0] addr;
input [15:0] di;
output [15:0] dout;
reg [15:0] RAM [1023:0];
reg [15:0] dout;

always @(posedge clk)
begin
  if (en)
  begin
    if (we)
      begin
        RAM[addr] <= di;
        dout <= di;
      end
   else
    dout <= RAM[addr];
  end
end
endmodule
```

- Single port block RAM with no change mode

```
module rams_        (clk, we, en, addr, di, dout);

input clk;
input we;
input en;
input [9:0] addr;
input [15:0] di;
output [15:0] dout;

reg [15:0] RAM [1023:0];
reg [15:0] dout;

always @(posedge clk)
begin
  if (en)
  begin
    if (we)
      RAM[addr] <= di;
    else
      dout <= RAM[addr];
  end
end
endmodule
```

```
module simple_ram_dual_clock
(clk,ena,enb,wea,addra,addrb,dia,dob);
input clk,ena,enb,wea;
input [9:0] addra,addrb;
input [15:0] dia;
output [15:0] dob;
reg [15:0] ram [1023:0];
reg [15:0] doa,dob;

always @(posedge clk) begin
 if (ena) begin
    if (wea)
        ram[addra] <= dia;
 end
end

always @(posedge clk) begin
  if (enb)
    dob <= ram[addrb];
end

endmodule
```

- Simple Dual Port Block RAM
- Single Clock
- Two clock (HW)

True Dual Port BRAM

- Dual-Port Block RAM with Two Write Ports in Read First Mode

```
module rams_          (clka,clkb,ena,enb,wea,web,
addra,addrb,dia,dib,doa,dob);

input clka,clkb,ena,enb,wea,web;
input [9:0] addra,addrb;
input [15:0] dia,dib;
output [15:0] doa,dob;
reg [15:0] ram [1023:0];
reg [15:0] doa,dob;

always @(posedge clka)
begin
  if (ena)
    begin
      if (wea)
        ram[addra] <= dia;
      doa <= ram[addra];
    end
end

always @(posedge clkb)
begin
  if (enb)
    begin
      if (web)
        ram[addrb] <= dib;
      dob <= ram[addrb];
    end
end

endmodule
```

- Dual-Port RAM with Asynchronous Read (Distributed RAM)

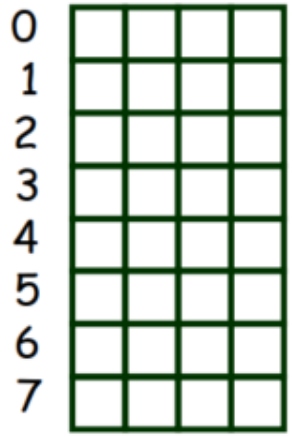
```
module rams_dist (clk, we, a, dpra, di, spo, dpo);

input clk;
input we;
input [5:0] a;
input [5:0] dpra;
input [15:0] di;
output [15:0] spo;
output [15:0] dpo;
reg  [15:0] ram [63:0];

always @(posedge clk)
begin
 if (we)
   ram[a] <= di;
end

assign spo = ram[a];
assign dpo = ram[dpra];

endmodule
```

ROM Modelling

```
module rom_case(
output reg [3:0] z,
input wire [2:0] a; // Address - 8 deep memory.
always @* // @(a)
case (a)
3'b000: z = 4'b1011;
3'b001: z = 4'b0001;
3'b100: z = 4'b0011;
3'b110: z = 4'b0010;
3'b111: z = 4'b1110;
default: z = 4'b0000;
endcase
endmodule // rom_case
// z is the ROM, and its address size is
// determined by a.
```

ROM Modelling

```
module rom_2dimarray_initial (
output wire [3:0] z,
input wire [2:0] a; // address- 8 deep memory
// Declare a memory rom of 8 4-bit registers.
// The indices are 0 to 7:
reg [3:0] rom[0:7];
initial begin
rom[0] = 4'b1011;
rom[1] = 4'b0001;
rom[2] = 4'b0011;
rom[3] = 4'b0010;
rom[4] = 4'b1110;
rom[5] = 4'b0111;
rom[6] = 4'b0101;
rom[7] = 4'b0100;
end
assign z = rom[a];
endmodule
```




```
module rom_2dimarray_initial_readmem (
output wire [3:0] z,
input wire [2:0] a);
// Declare a memory rom of 8 4-bit registers.
// The indices are 0 to 7:
reg [3:0] rom[0:7];
initial $readmemb("rom.data", rom);
assign z = rom[a];
endmodule
// with data in text file
```

```
// Example of content "rom.data" file:
1011 // addr=0
1000 // addr=1
0000 // addr=2
1000 // addr=3
0010 // addr=4
0101 // addr=5
1111 // addr=6
1001 // addr=7
```

Summary: Block RAM

- Dual-port 36 Kb block RAM
- Multiple configuration options: True dual port, simple dual port, single port
- Built-in cascade logic for 64Kx1 (only for true dual port)
- Fully synchronous operation (Nothing happens without a clock)
- Two ports are symmetrical and totally independent, sharing only the stored data
- In addition, the read port width can be different from the write port width for each port.
- The memory content can be initialized or cleared by the configuration bitstream.

Summary: Block RAM

- The Block Memory Generator core can generate memory structures from 1 to 4608 bits wide, and at least two locations deep.
- The maximum depth of the memory is limited only by the number of block RAM primitives in the target device.

Port Aspect Ratio for RAMB18E1 (in TDP Mode)

Port Data Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
1	14	16,384	[13:0]	[0]	NA
2	13	8,192	[13:1]	[1:0]	NA
4	12	4,096	[13:2]	[3:0]	NA
9	11	2,048	[13:3]	[7:0]	[0]
18	10	1,024	[13:4]	[15:0]	[1:0]

Port Aspect Ratio for RAMB18E1 (in SDP Mode)

Port Data Width(1)	Alternate Port Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
32	1	14	16,384	[13:0]	[0]	NA
32	2	13	8,192	[13:1]	[1:0]	NA
32	4	12	4,096	[13:2]	[3:0]	NA
36	9	11	2,048	[13:3]	[7:0]	[0]
36	18	10	1,024	[13:4]	[15:0]	[1:0]
36	36	9	512	[13:5]	[31:0]	[3:0]

Port Aspect Ratio for RAMB36E1 (in TDP Mode)

Port Data Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
1	15	32,768	[14:0]	[0]	NA
2	14	16,384	[14:1]	[1:0]	NA
4	13	8,192	[14:2]	[3:0]	NA
9	12	4,096	[14:3]	[7:0]	[0]
18	11	2,048	[14:4]	[15:0]	[1:0]
36	10	1,024	[14:5]	[31:0]	[3:0]
1 (Cascade)	16	65536	[15:0]	[0]	NA

Port Aspect Ratio for RAMB36E1 (in SDP Mode)

Port Data Width(1)	Alternate Port Width	Port Address Width	Depth	ADDR Bus	DI Bus DO Bus	DIP Bus DOP Bus
64	1	15	32,768	[14:0]	[0]	NA
64	2	14	16,384	[14:1]	[1:0]	NA
64	4	13	8,192	[14:2]	[3:0]	NA
72	9	12	4,096	[14:3]	[7:0]	[0]
72	18	11	2,048	[14:4]	[15:0]	[1:0]
72	36	10	1,024	[14:5]	[31:0]	[3:0]
72	72	9	512	[14:6]	[63:0]	[7:0]

Conflict Avoidance

- The 7 series FPGAs block RAM is a true dual-port RAM where both ports can access any memory location at any time.
- Address collisions can occur when accessing the same memory location from both ports. (An address collision is when both block RAM ports access the same address location in the same clock cycle.)
- There are two fundamental clock type setups, common clock and independent clock.
- Common (synchronous) clocks are driven by a common clock buffer driver.
- All other CLKA and CLKB connections are considered independent (asynchronous) clocks.

Conflict Avoidance

- When both ports are reading, the operations complete successfully
- When both ports are writing different data, the memory location is written with non-deterministic data.
- When one port is writing and the other port is reading, the write is always successful but the resulting read memory value can vary.

Conflict Avoidance

Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Common	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change

Conflict Avoidance

Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Common	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change
Common	RF	RF/WF/NC	1 (DIA)	0	Old memory data	Old memory data	DIA

Conflict Avoidance

Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Common	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change
Common	RF	RF/WF/NC	1 (DIA)	0	Old memory data	Old memory data	DIA
Common	WF	RF/WF/NC	1 (DIA)	0	DIA	X	DIA

Conflict Avoidance

Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Common	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change
Common	RF	RF/WF/NC	1 (DIA)	0	Old memory data	Old memory data	DIA
Common	WF	RF/WF/NC	1 (DIA)	0	DIA	X	DIA
Common	NC	RF/WF/NC	1 (DIA)	0	No change	X	DIA

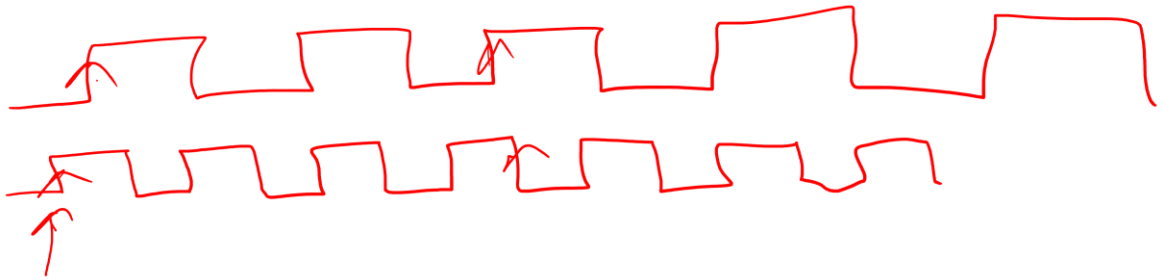
Conflict Avoidance

Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Common	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change
Common	RF	RF/WF/NC	1 (DIA)	0	Old memory data	Old memory data	DIA
Common	WF	RF/WF/NC	1 (DIA)	0	DIA	X	DIA
Common	NC	RF/WF/NC	1 (DIA)	0	No change	X	DIA
Common	RF/WF/NC	RF	0	1 (DIB)	Old memory data	Old memory data	DIB
Common	RF/WF/NC	WF	0	1 (DIB)	X	DIB	DIB
Common	RF/WF/NC	NC	0	1 (DIB)	X	No change	DIB

Conflict Avoidance

Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Common	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change
Common	RF	RF/WF/NC	1 (DIA)	0	Old memory data	Old memory data	DIA
Common	WF	RF/WF/NC	1 (DIA)	0	DIA	X	DIA
Common	NC	RF/WF/NC	1 (DIA)	0	No change	X	DIA
Common	RF/WF/NC	RF	0	1 (DIB)	Old memory data	Old memory data	DIB
Common	RF/WF/NC	WF	0	1 (DIB)	X	DIB	DIB
Common	RF/WF/NC	NC	0	1 (DIB)	X	No change	DIB
Common	RF/WF/NC	RF/WF/NC	1	1	X	X	X

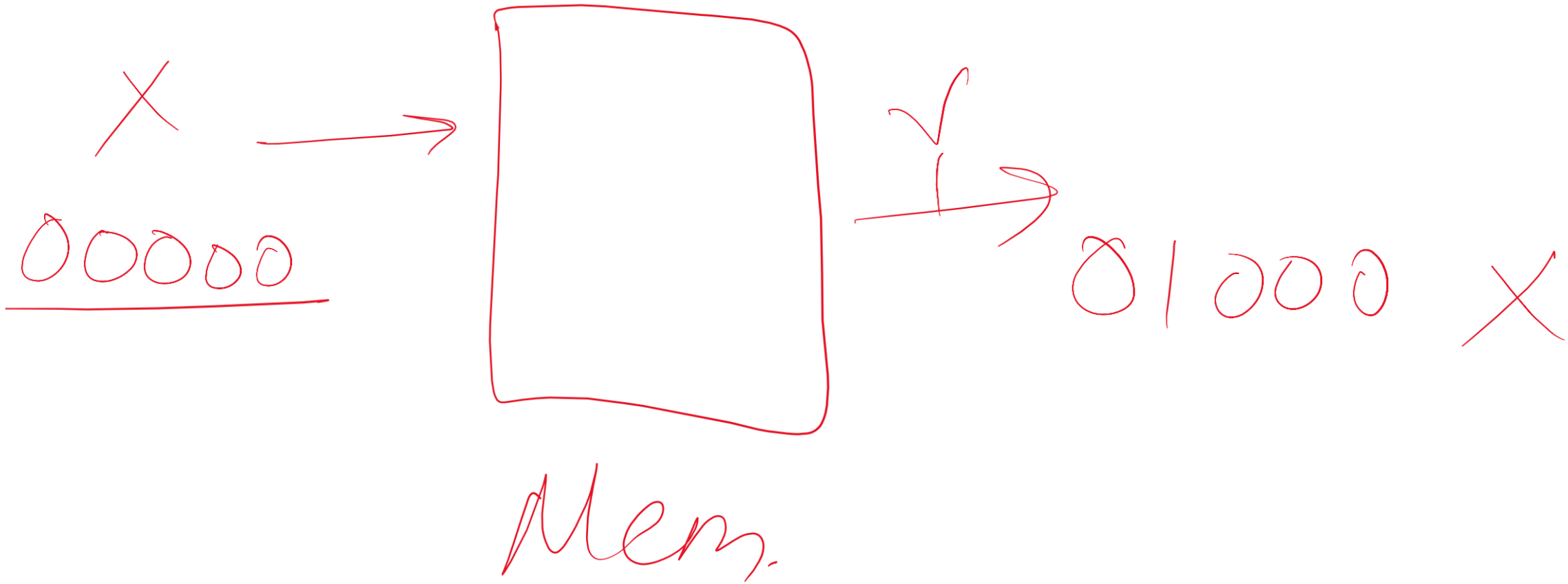
Conflict Avoidance

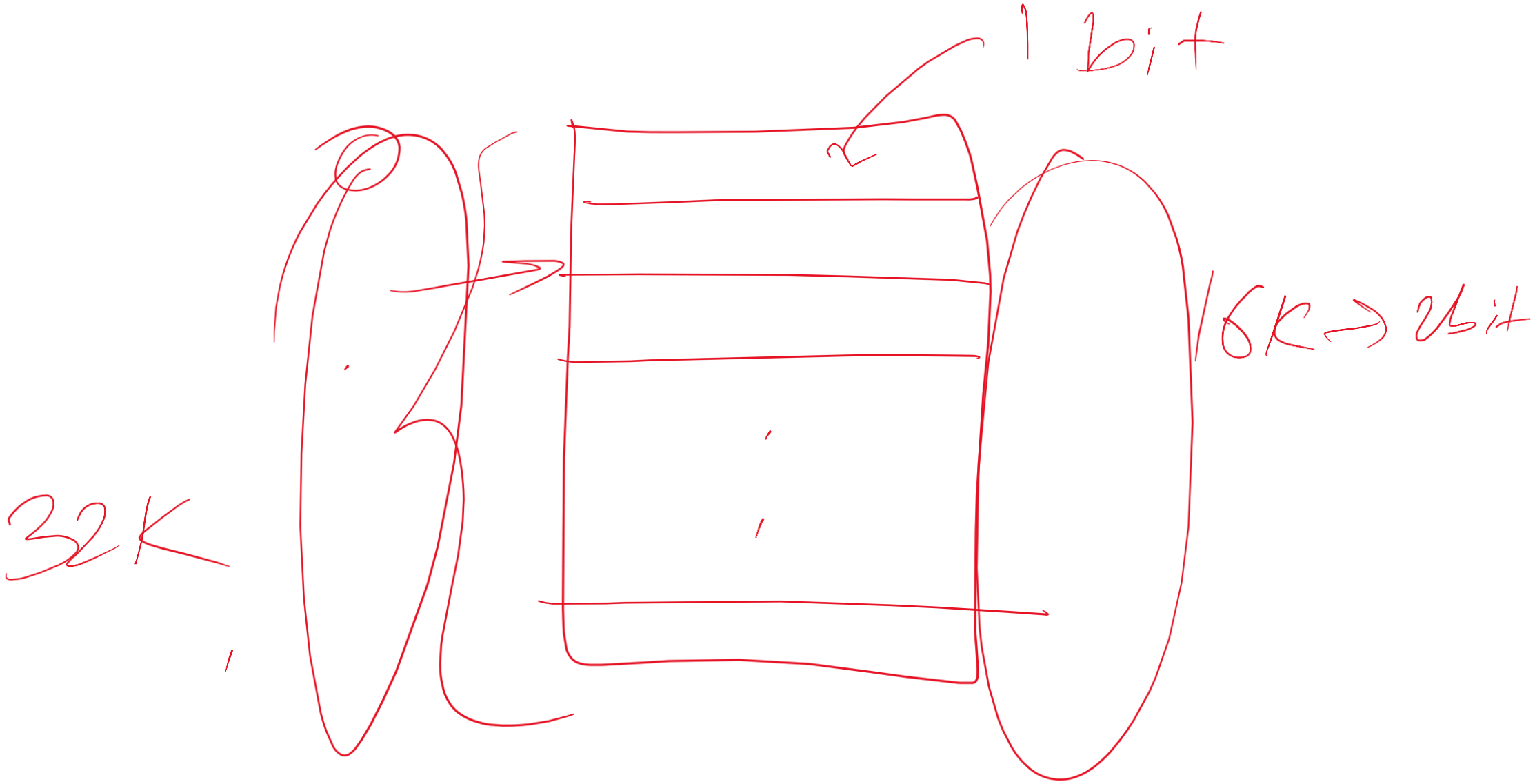


Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Independent	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change
Independent	RF	RF/WF/NC	1 (DIA)	0	Old memory data	X	DIA
Independent	WF	RF/WF/NC	1 (DIA)	0	DIA	X	DIA
Independent	NC	RF/WF/NC	1 (DIA)	0	No change	X	DIA

Conflict Avoidance

Clock Type	Write Mode Port A	Write Mode Port B	Write Enable Port A (Data)	Write Enable Port B (Data)	Resulting Data Out Port A	Resulting Data Out Port B	Resulting Memory Value
Independent	RF/WF/NC	RF/WF/NC	0	0	Old memory data	Old memory data	No change
Independent	RF	RF/WF/NC	1 (DIA)	0	Old memory data	X	DIA
Independent	WF	RF/WF/NC	1 (DIA)	0	DIA	X	DIA
Independent	NC	RF/WF/NC	1 (DIA)	0	No change	X	DIA
Independent	RF/WF/NC	RF	0	1 (DIB)	X	Old memory data	DIB
Independent	RF/WF/NC	WF	0	1 (DIB)	X	DIB	DIB
Independent	RF/WF/NC	NC	0	1 (DIB)	X	No change	DIB
Independent	RF/WF/NC	RF/WF/NC	1	1	X	X	X


X
00000
Y
01000 X
Mem.

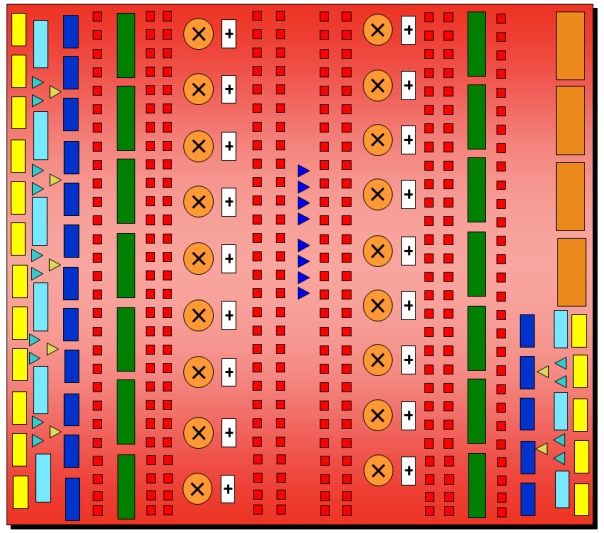

1 bit
32K
16K → 4bit

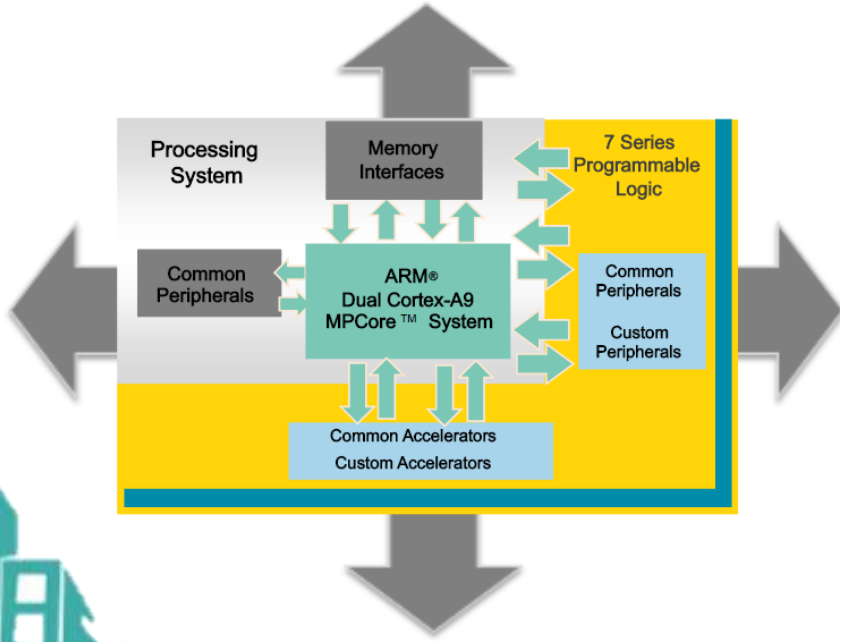
DEPARTMENT OF ELECTRONICS & COMMUNICATIONS ENGINEERING

A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design

SDK
Software Development Kit
VIVADO™

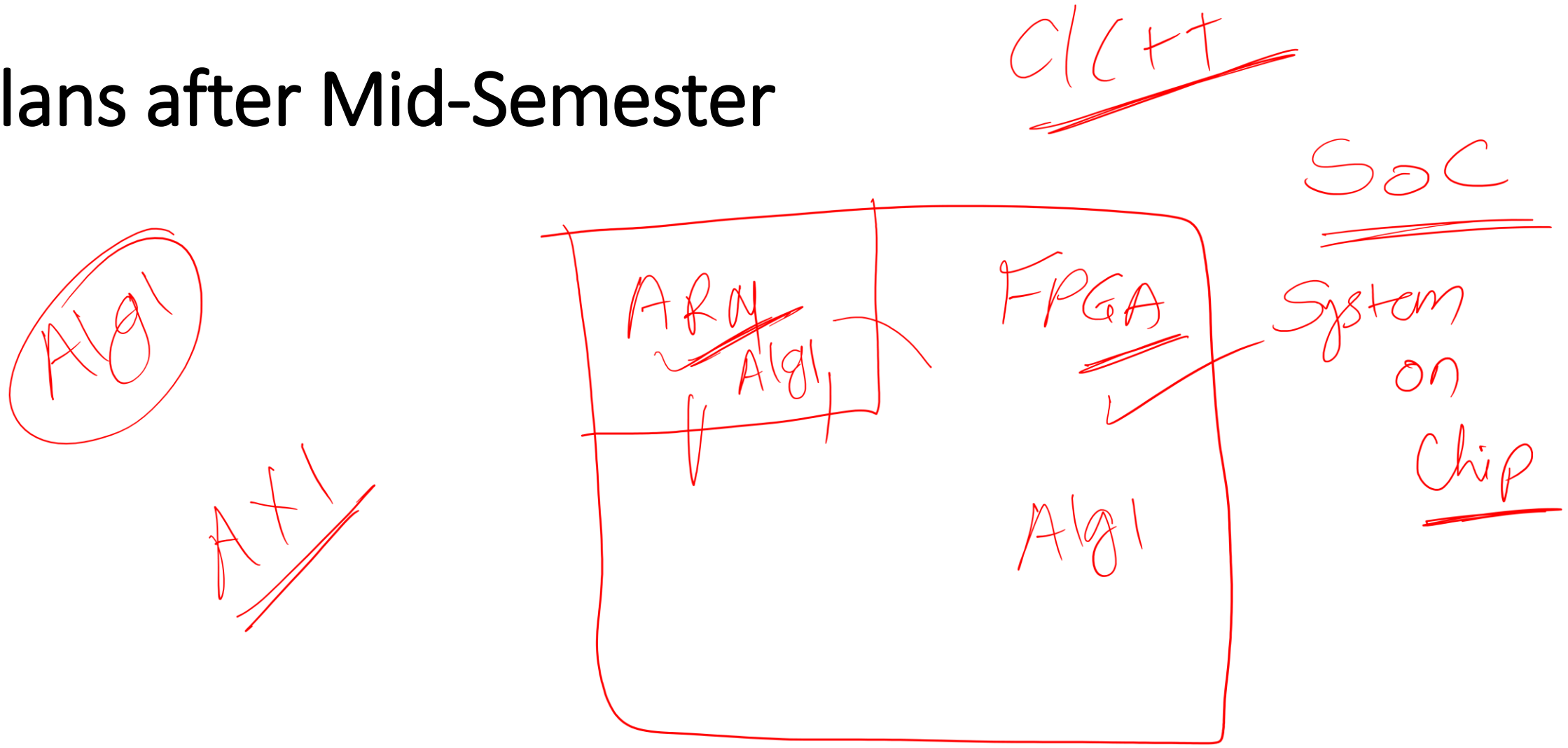




A+ Projects

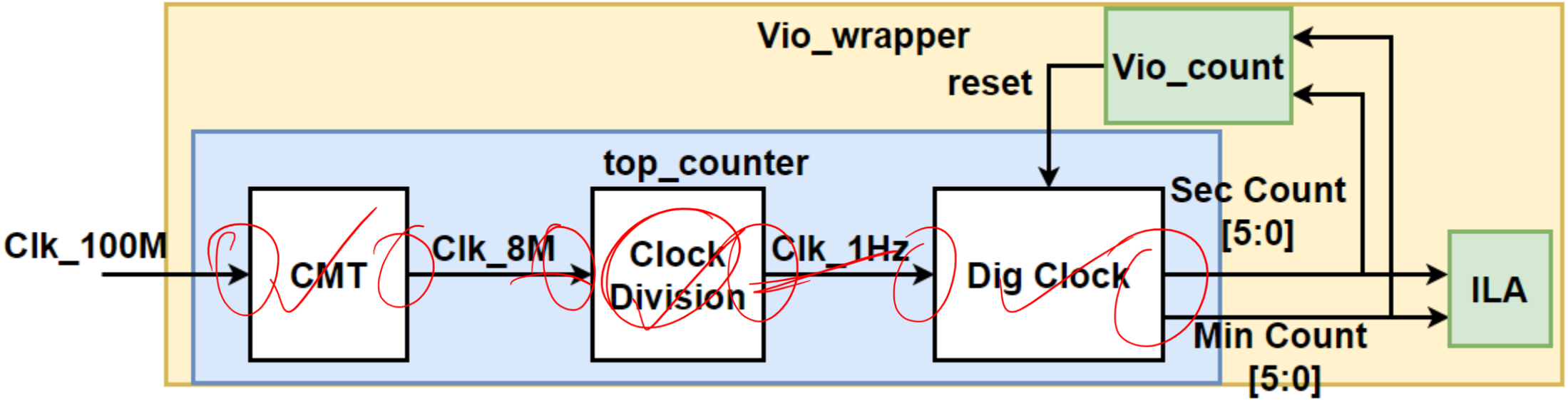
- Understand Vivado High Level Synthesis (HLS) Tool which converts C/C++ code into Verilog. Create hardware IP for FFT using HLS tool.
- Understand DDS (Direct Digital Synthesizer) and FIR Filter IPs in Vivado and their features. Design and validate various features of these IP using suitable examples.
- Deliverables: Handouts and Code
- Mentors: PhD Students in A2A lab
- Confirmation Deadline: Nov. 22, 2024
- Submission Deadline: Dec. 17, 2024

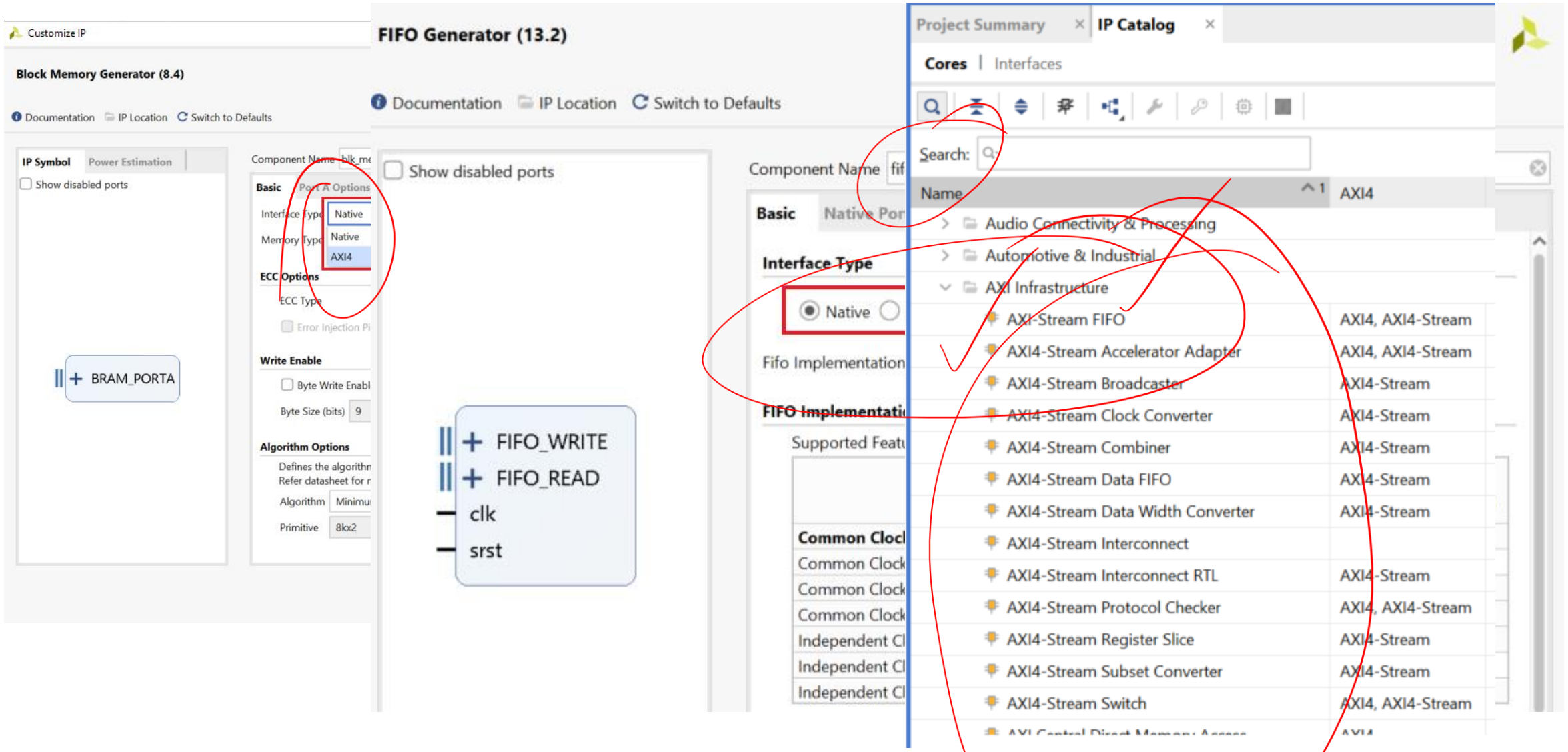
Plans after Mid-Semester

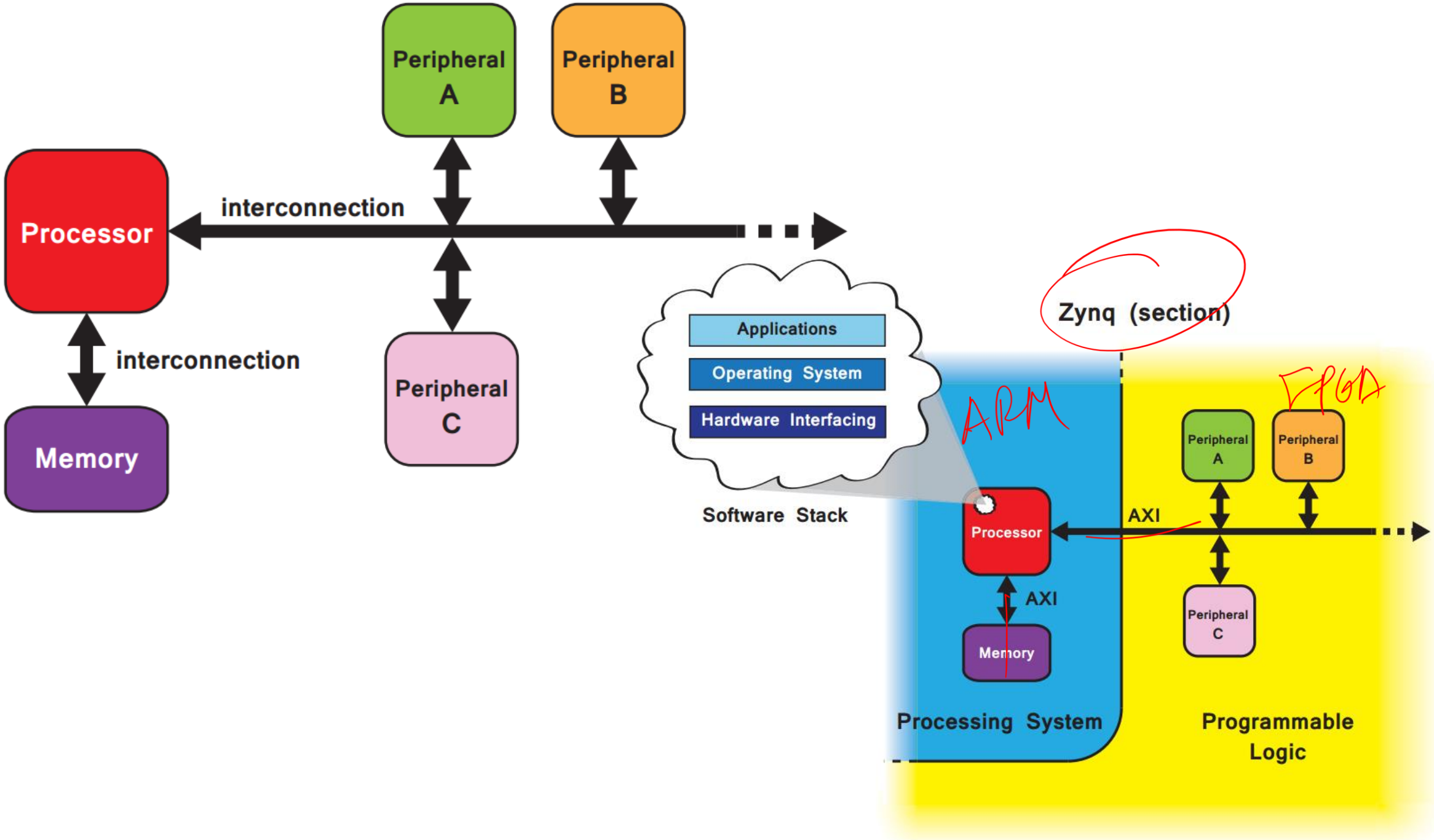


Advanced eXtensible Interface (AXI)

Native Interface

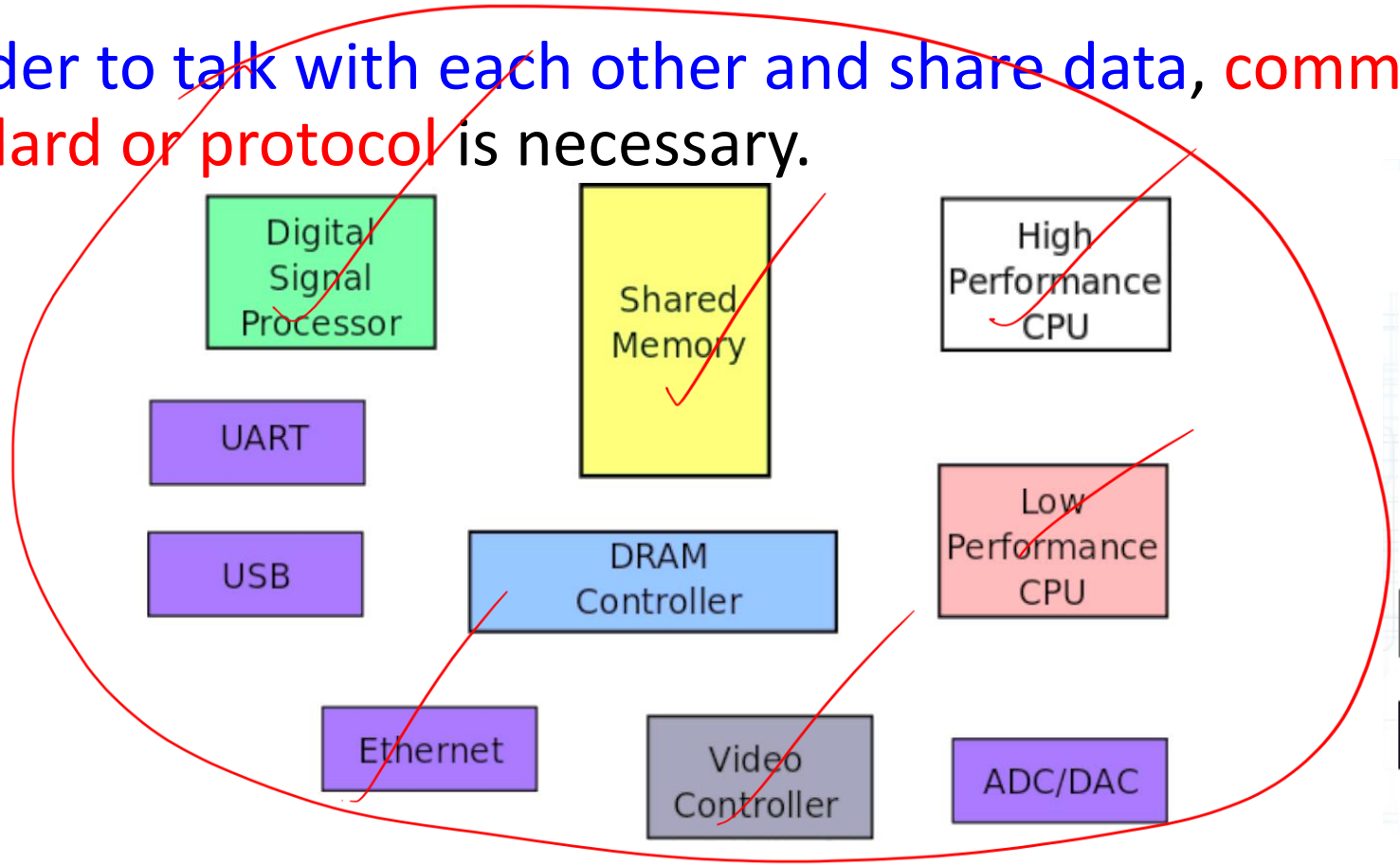


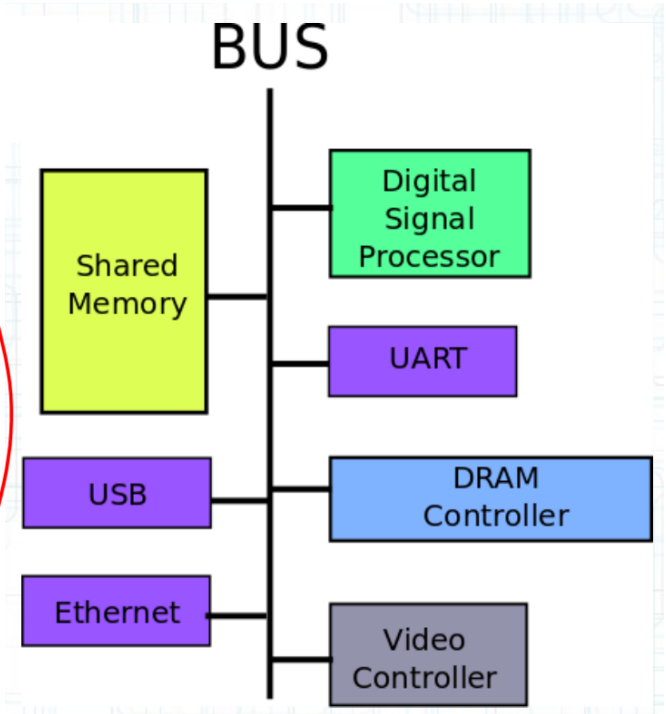

Customize IP
Block Memory Generator (8.4)
Documentation
IP Location
Switch to Defaults
IP Symbol
Power Estimation
Show disabled ports
BRAM_PORTA
Component Name
Basic
Port A Options
Interface Type
Native
Memory Type
Native
AXI4
ECC Options
ECC Type
Write Enable
Byte Write Enabl
Byte Size (bits)
9
Algorithm Options
Algorithm
Primitive
8kx2
FIFO Generator (13.2)
Documentation
IP Location
Switch to Defaults
Show disabled ports
FIFO_WRITE
FIFO_READ
clk
srst
Component Name
Basic
Interface Type
Native
Fifo Implementation
Common Clock
Independent Cl
Project Summary
IP Catalog
Cores
Interfaces
Search:
Name
AXI4
Audio Connectivity & Processing
Automotive & Industrial
AXI Infrastructure
AXI-Stream FIFO
AXI4, AXI4-Stream
AXI4-Stream Accelerator Adapter
AXI4, AXI4-Stream
AXI4-Stream Broadcaster
AXI4-Stream
AXI4-Stream Clock Converter
AXI4-Stream
AXI4-Stream Combiner
AXI4-Stream
AXI4-Stream Data FIFO
AXI4-Stream
AXI4-Stream Data Width Converter
AXI4-Stream
AXI4-Stream Interconnect
AXI4-Stream Interconnect RTL
AXI4-Stream
AXI4-Stream Protocol Checker
AXI4, AXI4-Stream
AXI4-Stream Register Slice
AXI4-Stream
AXI4-Stream Subset Converter
AXI4-Stream
AXI4-Stream Switch
AXI4, AXI4-Stream


Peripheral A
Peripheral B
Processor
interconnection
interconnection
Memory
Peripheral C
Applications
Operating System
Hardware Interfacing
Software Stack
Zynq (section)
ARM
FPGA
Peripheral A
Peripheral B
AXI
Processor
AXI
Memory
Peripheral C
Processing System
Programmable Logic

Interface

- Modern day design consists of large number of IP blocks, each one designed to efficiently perform assigned task.
- In order to talk with each other and share data, communication standard or protocol is necessary.





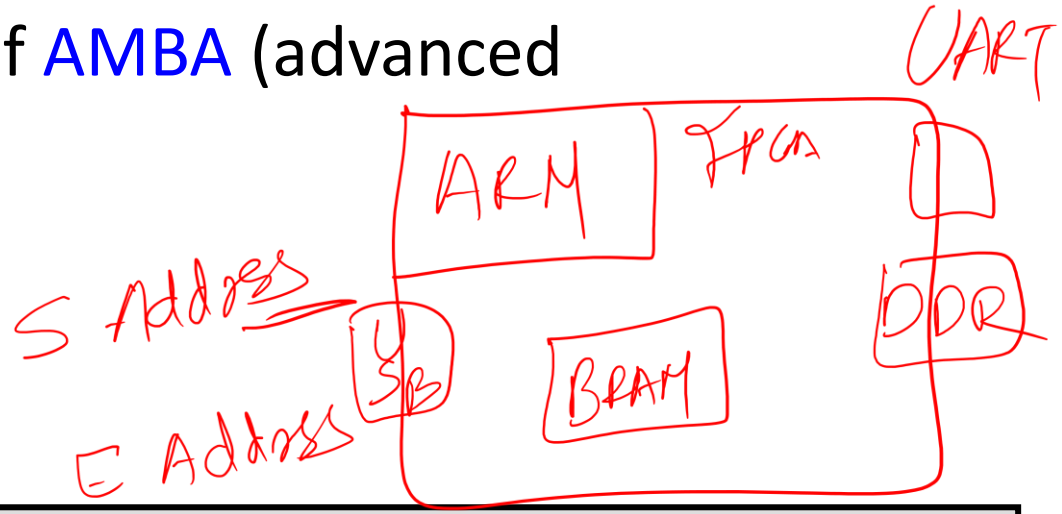
Interface

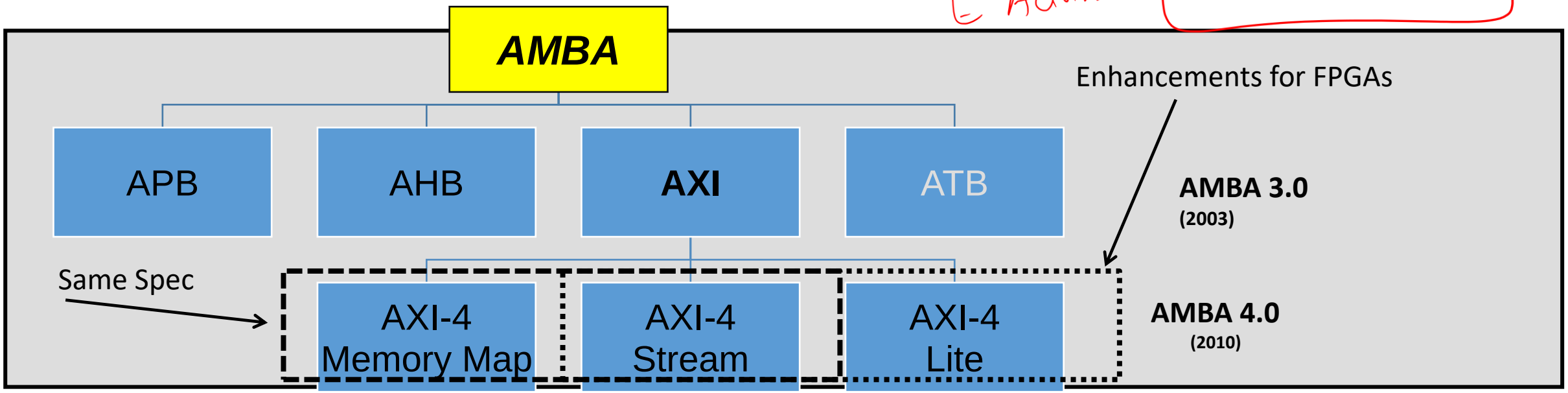
- **Advanced eXtensible Interface (AXI)** is one of the several standards developed by ARM and adopted by Xilinx for Zynq architecture (Others are IBM core connect and WishBone).
- **Availability**: By moving to an industry-standard, you have access not only to the Vivado IP Catalog, but also to a worldwide community of ARM partners.
- **Productivity**: By standardizing on the AXI interface, developers need to learn only a single protocol for IP
- **Flexibility**: Providing the right protocol for the application

AXI

Memory Mapped

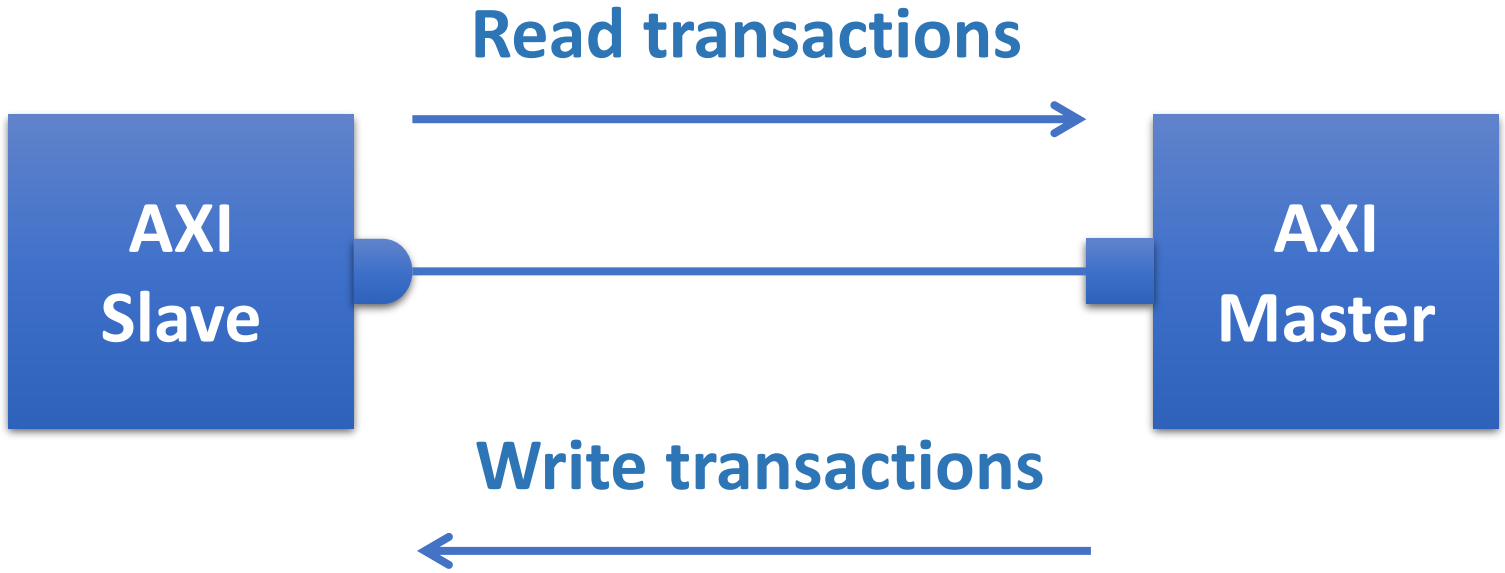
❖ Advanced eXtensible Interface (AXI): part of AMBA (advanced microcontroller bus architecture)





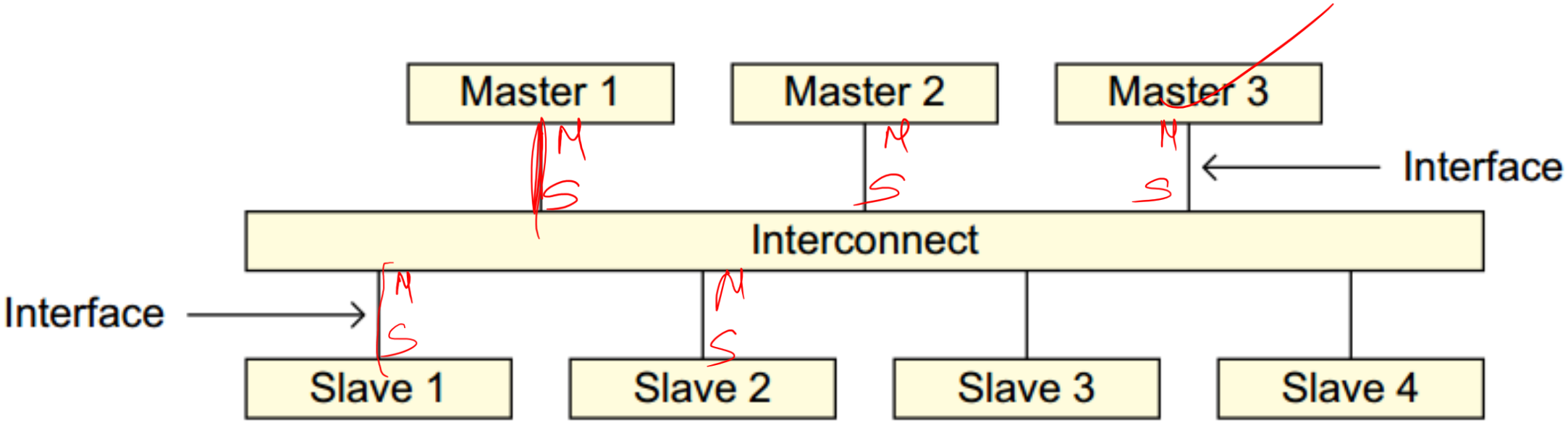
AXI

- Every AXI link contains two part: AXI master and AXI slave.
- AXI master initializes the transactions such as read and write. AXI slave is the one who responds to AXI master transactions.
- **Transaction:** Transfer of data from one point to another point



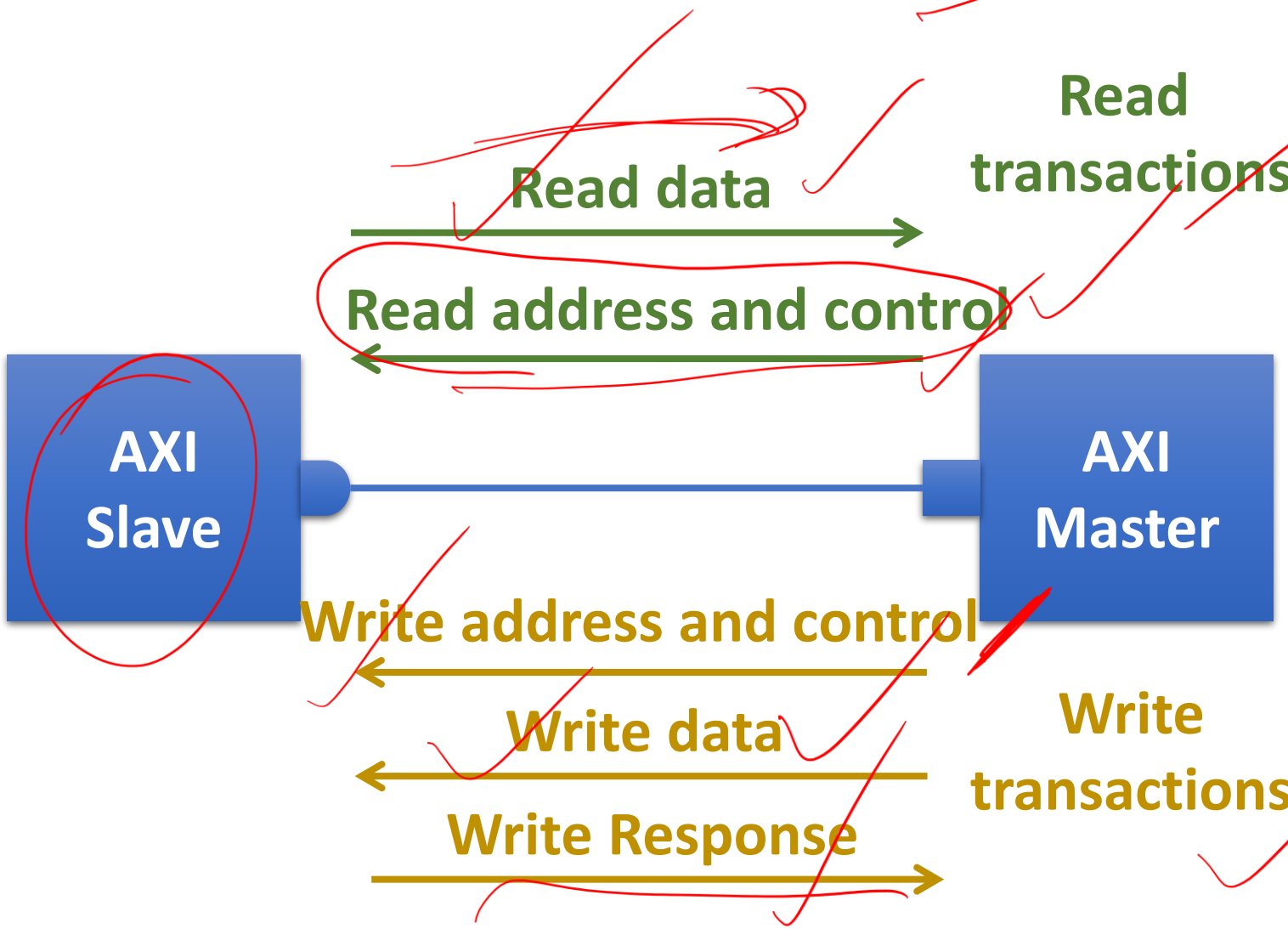
The AXI protocol provides a single interface definition, for the interfaces:

- between a master and the interconnect
- between a slave and the interconnect
- between a master and a slave.



AXI Memory Mapped

AXI Memory Mapped: Channels



Each channel may carry additional control information that describes the parameters of the transactions

AXI Memory Mapped

Burst Transfer

1MB

1 - 1B

16

Read transactions

Read data

Read address and control

AXI Slave

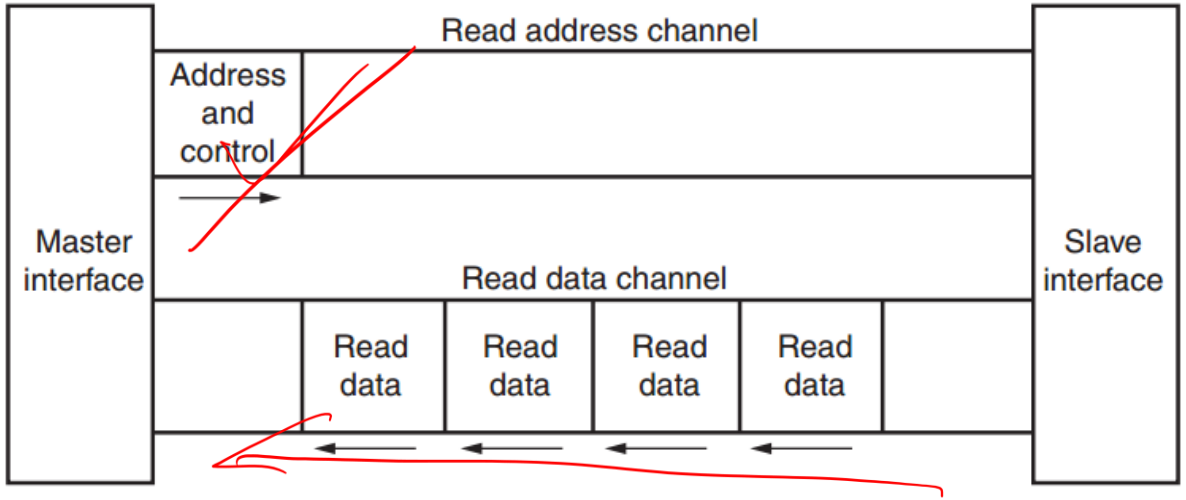
AXI Master

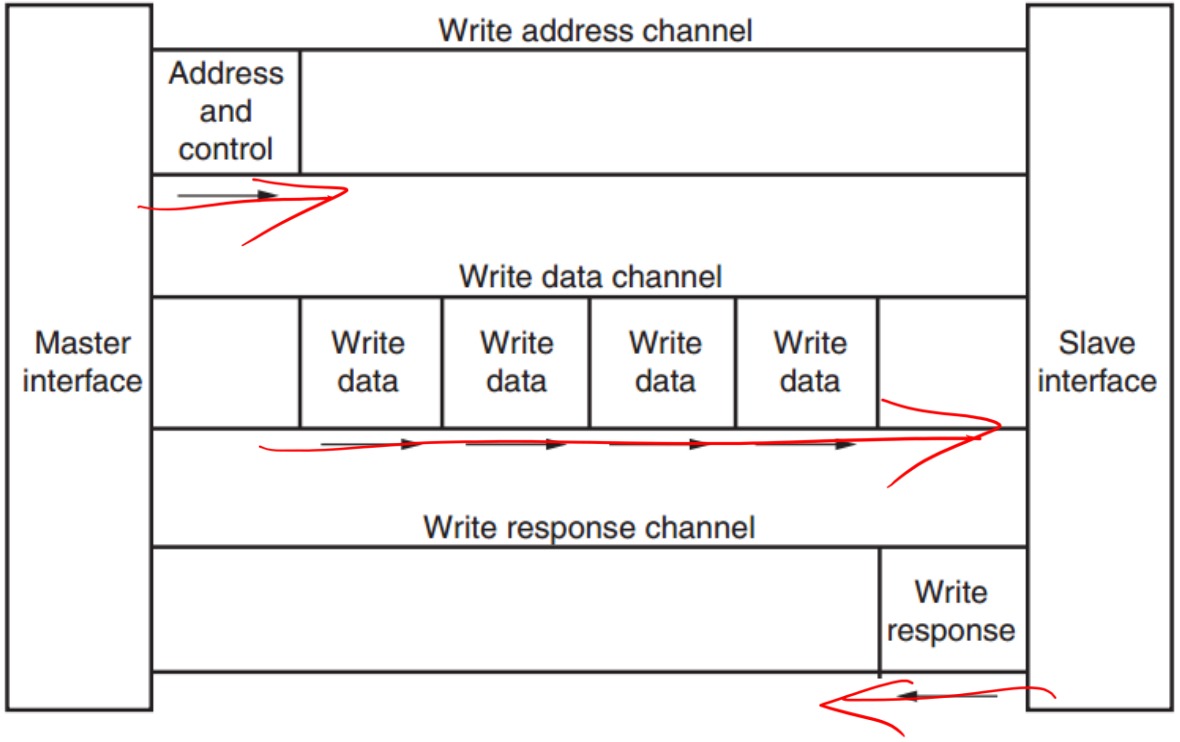
Write address and control

Write data

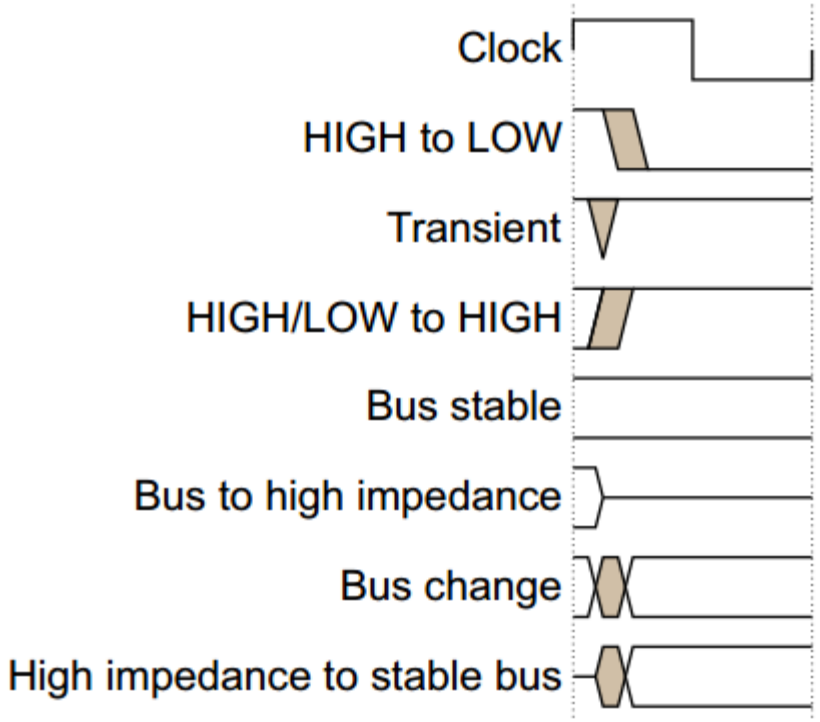
Write Response

Write transactions

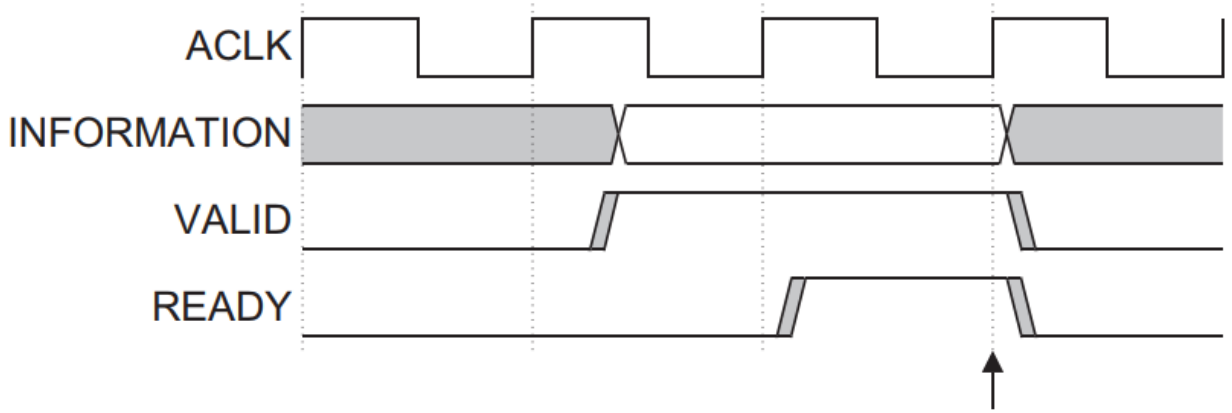




AXI Memory Mapped

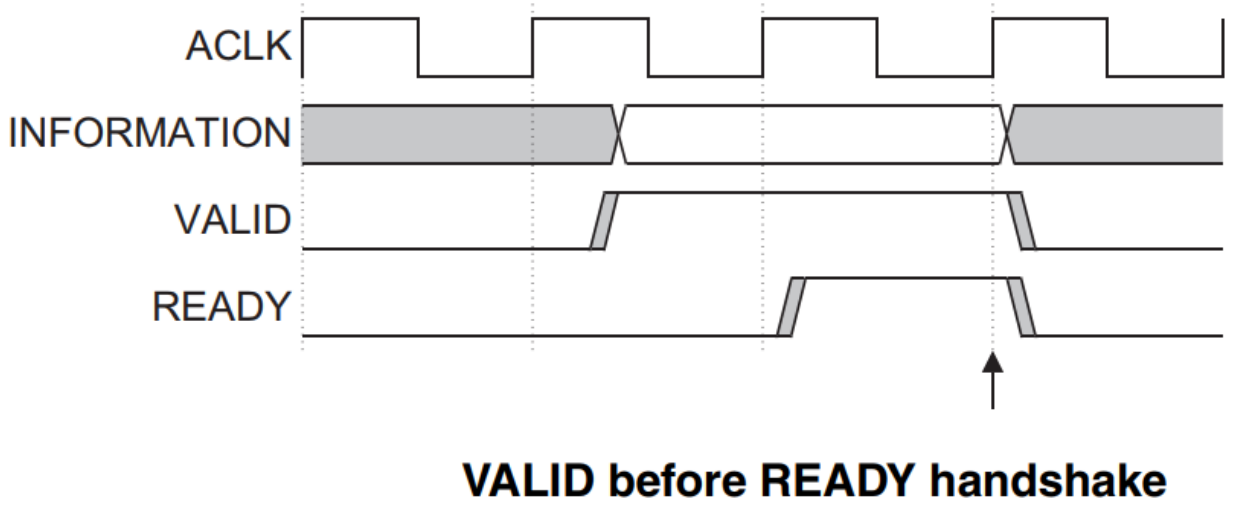


Key to timing diagram conventions



VALID before READY handshake

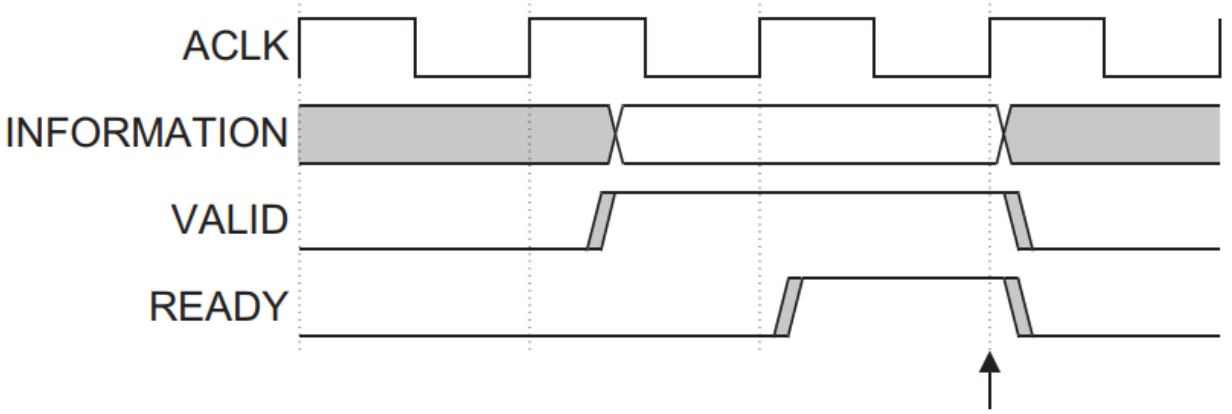
AXI Memory Mapped



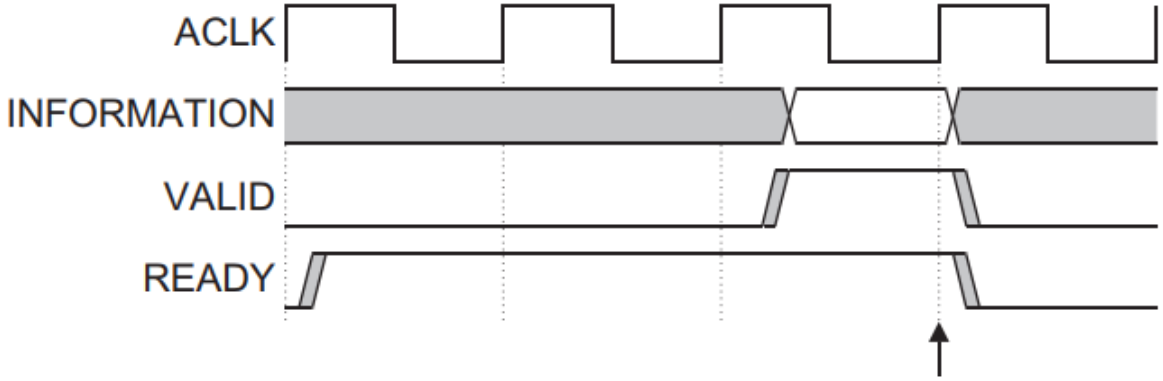
VALID before READY handshake

- The *source* generates the **VALID** signal to indicate when the address, data or control information is available.
- The *destination* generates the **READY** signal to indicate that it can accept the information.
- Transfer occurs only when *both* the **VALID** and **READY** signals are HIGH

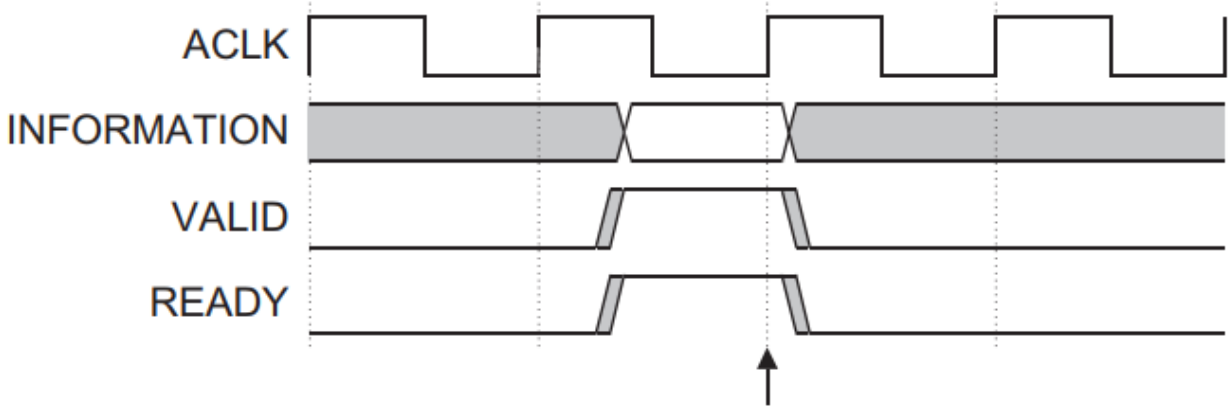
AXI Memory Mapped



VALID before READY handshake

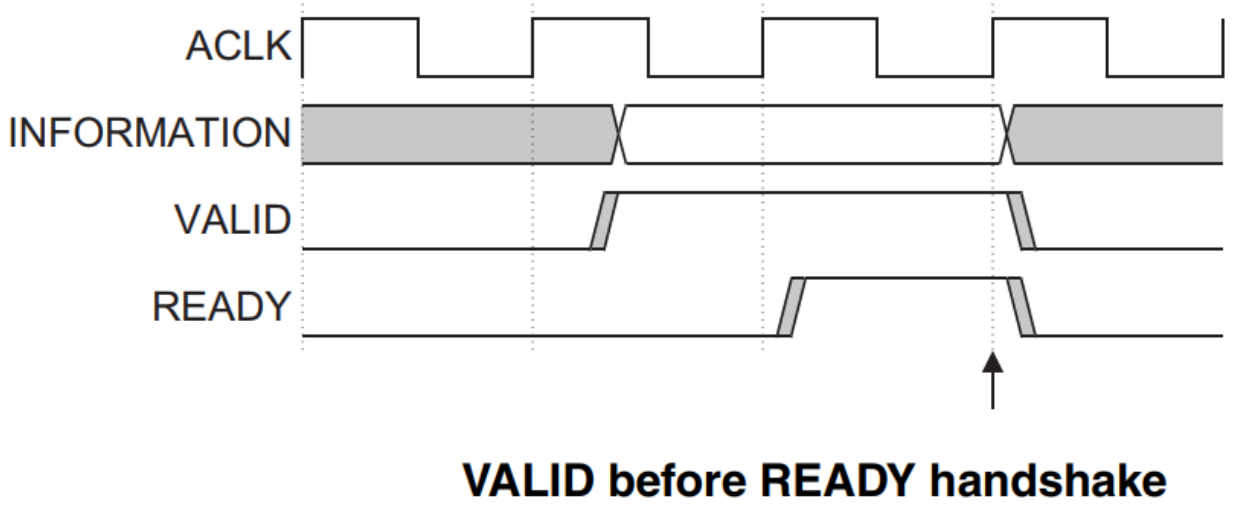


READY before VALID handshake



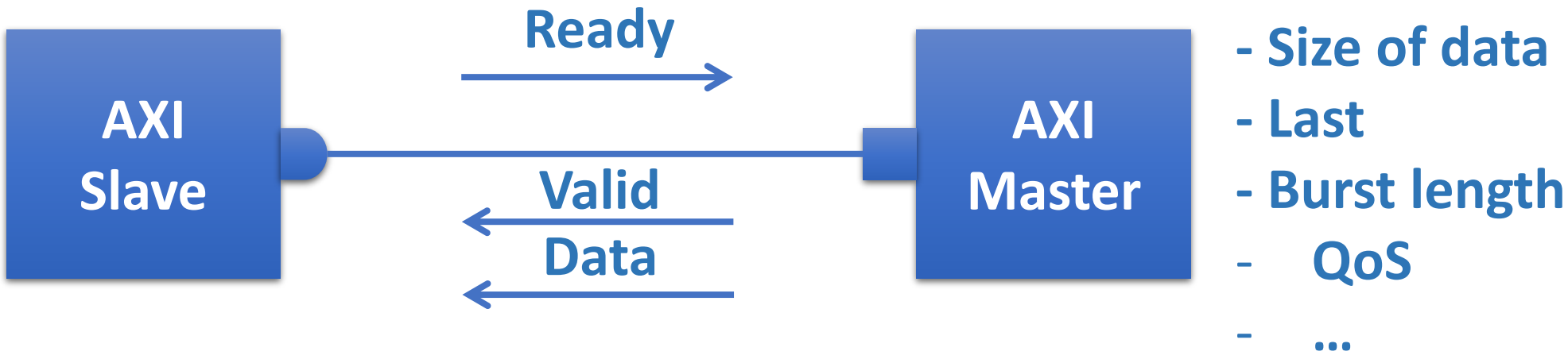
VALID with READY handshake

AXI Memory Mapped



- A source is NOT permitted to wait until READY is asserted before asserting VALID
- Once VALID is asserted it must remain asserted until the handshake occurs, at a rising clock edge at which VALID and READY are both asserted.
- A destination is permitted to wait for **VALID** to be asserted before asserting the corresponding **READY**.
- If **READY** is asserted, it is permitted to deassert **READY** before **VALID** is asserted.

AXI Memory Mapped



- Each of the independent channels consists of a set of information signals along with **VALID** and **READY** signals that provide a two-way handshake mechanism.
- The information source uses the **VALID** signal to show when valid address, data or control information is available on the channel.
- The destination uses the **READY** signal to show when it can accept the information.
- Both the read data channel and the write data channel also include a **LAST** signal to indicate the transfer of the final data item in a transaction.

AXI Memory Mapped

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**

Write address channel: The master can assert the AWVALID signal only when it drives valid address and control information. When asserted, AWVALID must remain asserted until the rising clock edge after the slave asserts AWREADY.

AXI Memory Mapped

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**

Write data channel: During a write burst, the master can assert the WVALID signal only when it drives valid write data. When asserted, WVALID must remain asserted until the rising clock edge after the slave asserts WREADY.

AXI Memory Mapped

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**
Write response channel	**BVALID, BREADY**

Write response channel: The slave can assert the BVALID signal only when it drives a valid write response. When asserted, BVALID must remain asserted until the rising clock edge after the master asserts BREADY.

AXI Memory Mapped

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**
Write response channel	**BVALID, BREADY**
Read address channel	**ARVALID, ARREADY**
Read data channel	**RVALID, RREADY**

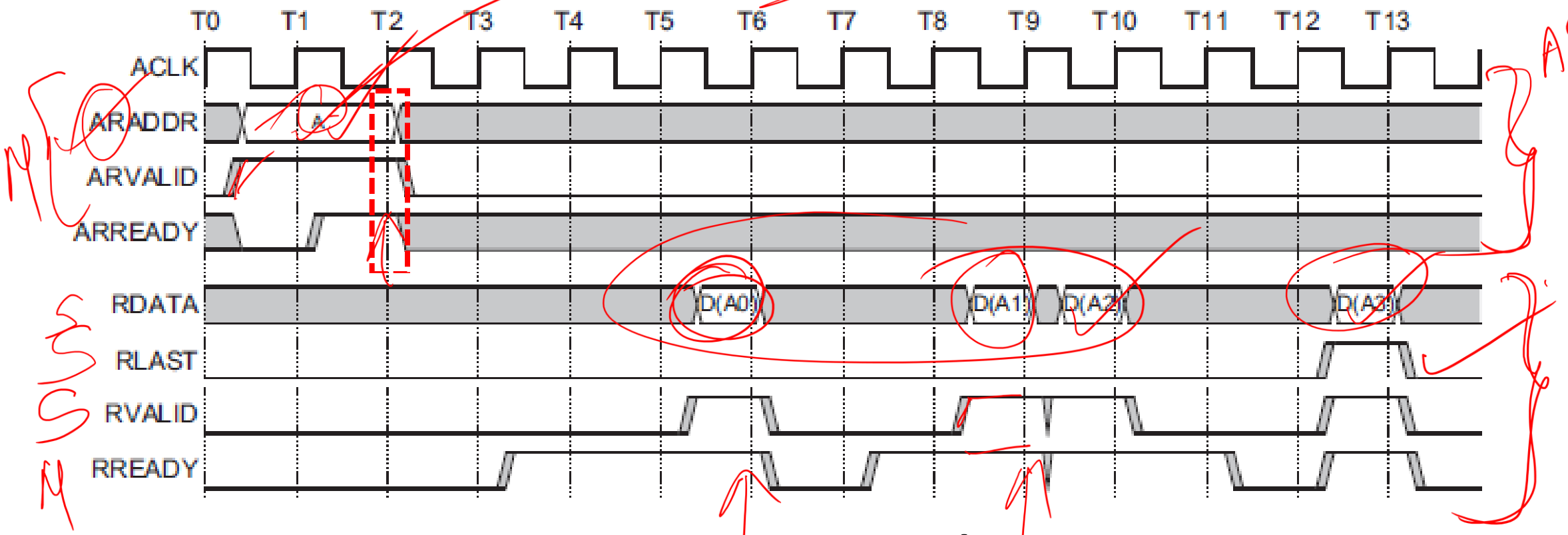
Read address channel: The master can assert the ARVALID signal only when it drives valid address and control information. When asserted, ARVALID must remain asserted until the rising clock edge after the slave asserts the ARREADY signal.

AXI Memory Mapped

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**
Write response channel	**BVALID, BREADY**
Read address channel	**ARVALID, ARREADY**
Read data channel	**RVALID, RREADY**

Read data channel: The slave can assert the RVALID signal only when it drives valid read data. When asserted, RVALID must remain asserted until the rising clock edge after the master asserts RREADY. The master interface uses the RREADY signal to indicate that it accepts the data.

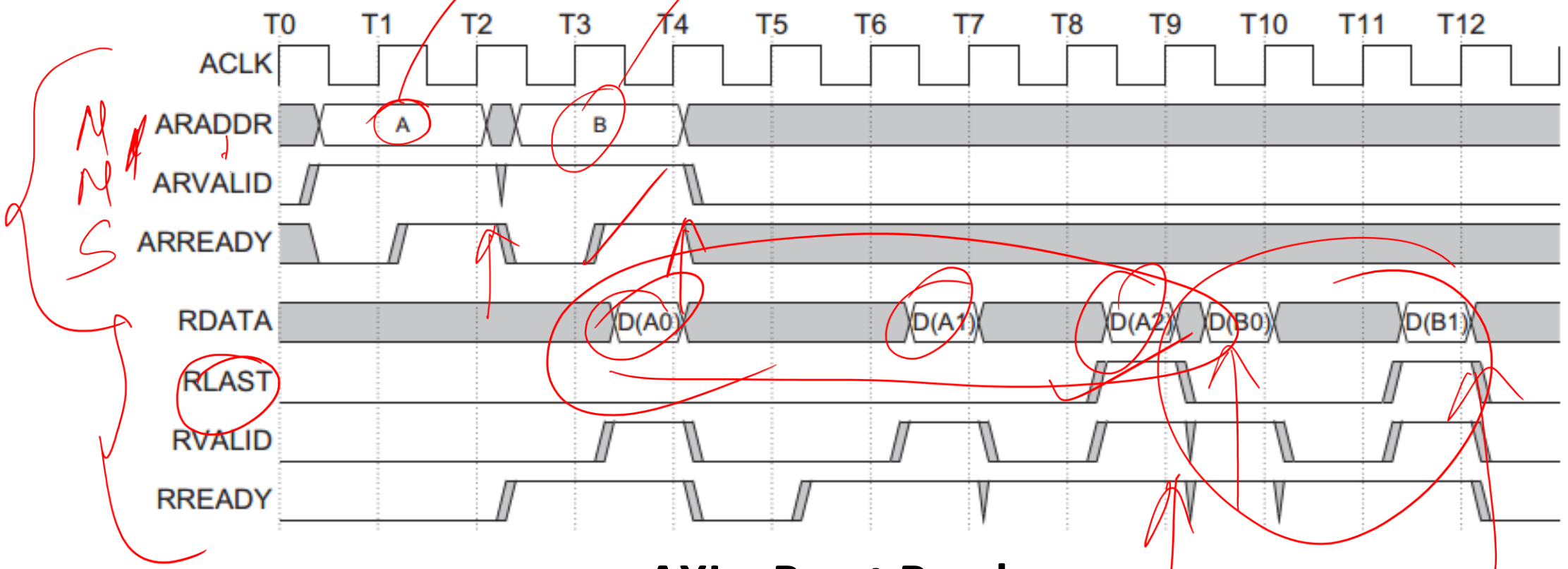
AXI Interface: Read



AXI – Burst Read

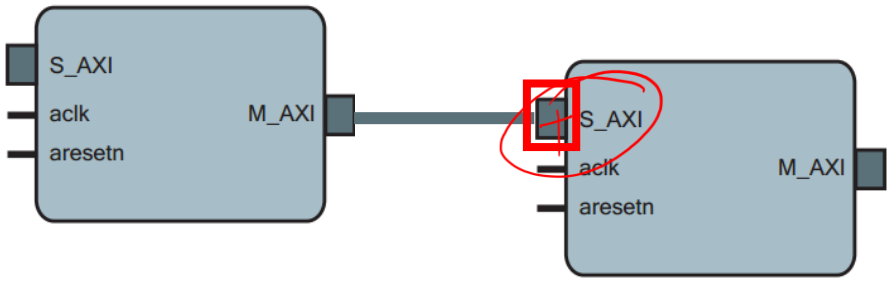
*Other control signals are not discussed here: ARID, ARLEN, ARSIZE, RID, RRESP,.....

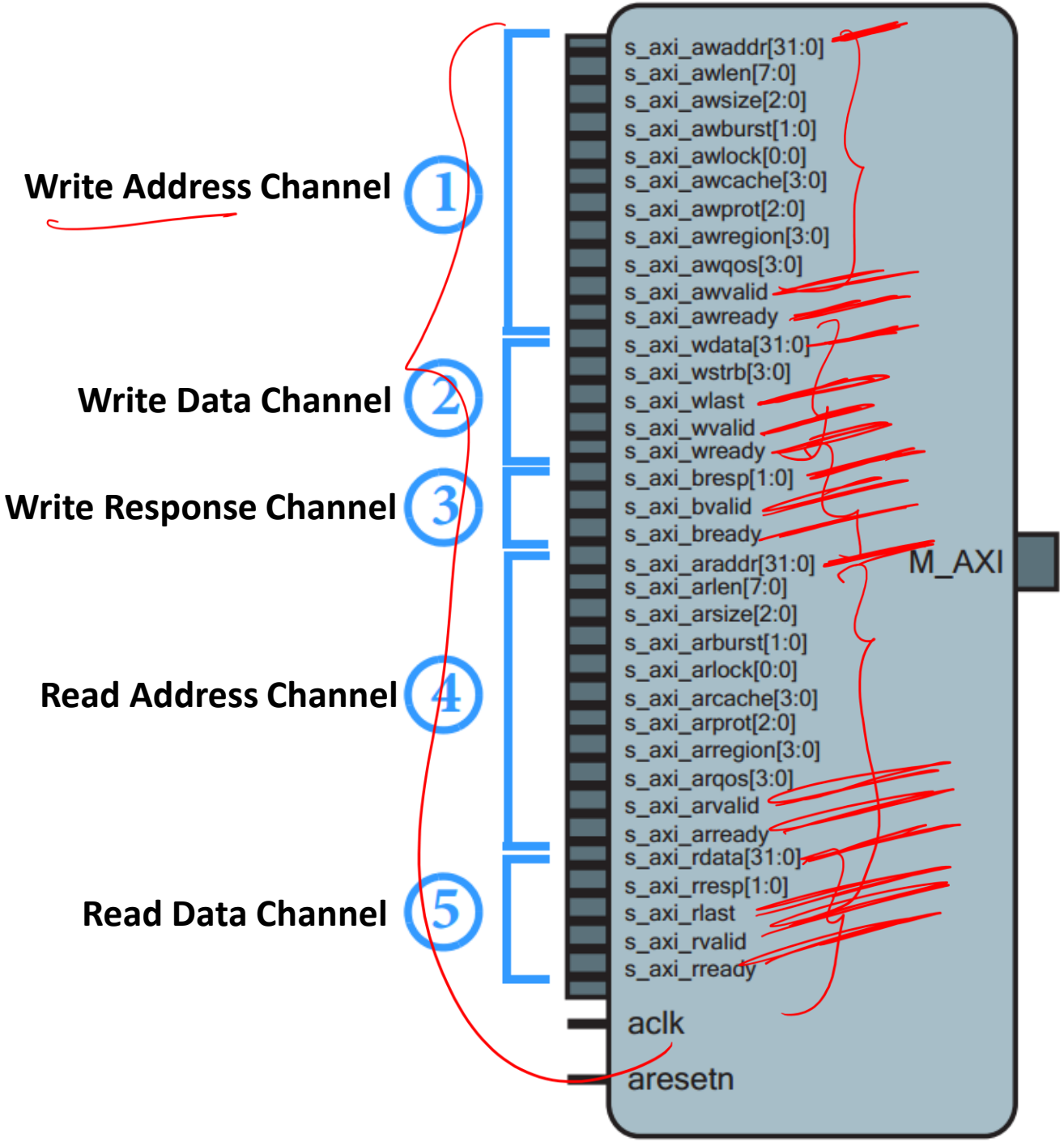
AXI Interface: Read



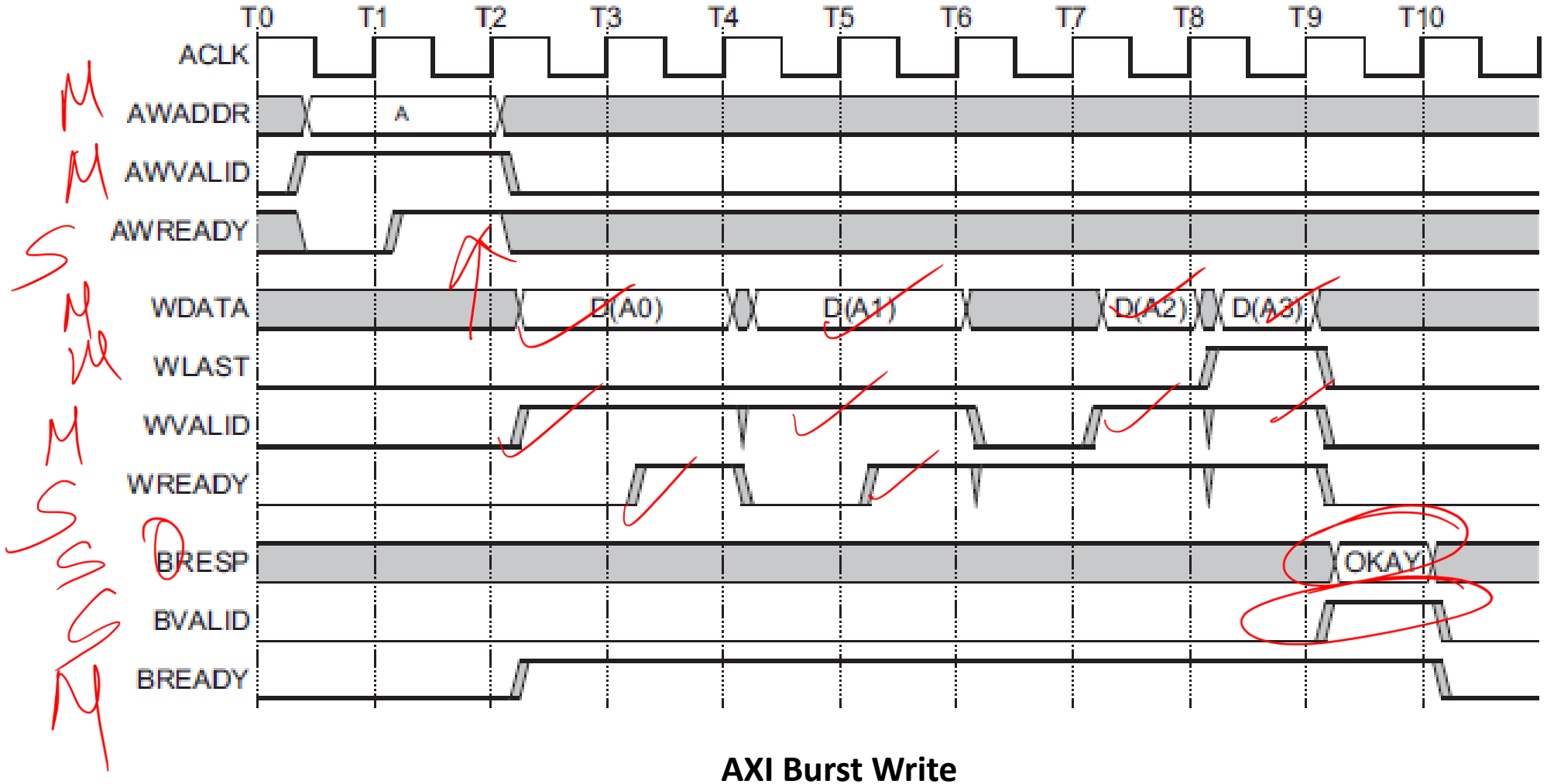
AXI – Burst Read

*Other control signals are not discussed here: ARID, ARLEN, ARSIZE, RID, RRESP,.....


S_AXI
aclk
aresetn
M_AXI
S_AXI
aclk
aresetn
M_AXI


Write Address Channel
1
Write Data Channel
2
Write Response Channel
3
Read Address Channel
4
Read Data Channel
5
s_axi_awaddr[31:0]
s_axi_awlen[7:0]
s_axi_awsize[2:0]
s_axi_awburst[1:0]
s_axi_awlock[0:0]
s_axi_awcache[3:0]
s_axi_awprot[2:0]
s_axi_awregion[3:0]
s_axi_awqos[3:0]
s_axi_awvalid
s_axi_awready
s_axi_wdata[31:0]
s_axi_wstrb[3:0]
s_axi_wlast
s_axi_wvalid
s_axi_wready
s_axi_bresp[1:0]
s_axi_bvalid
s_axi_bready
s_axi_araddr[31:0]
s_axi_arlen[7:0]
s_axi_arsize[2:0]
s_axi_arburst[1:0]
s_axi_arlock[0:0]
s_axi_arcache[3:0]
s_axi_arprot[2:0]
s_axi_arregion[3:0]
s_axi_arqos[3:0]
s_axi_arvalid
s_axi_arready
s_axi_rdata[31:0]
s_axi_rresp[1:0]
s_axi_rlast
s_axi_rvalid
s_axi_rready
M_AXI
aclk
aresetn

AXI Interface: Write



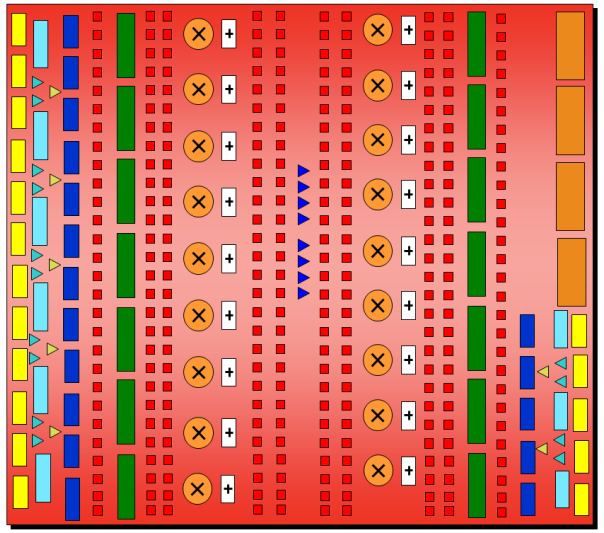
AXI Burst Write

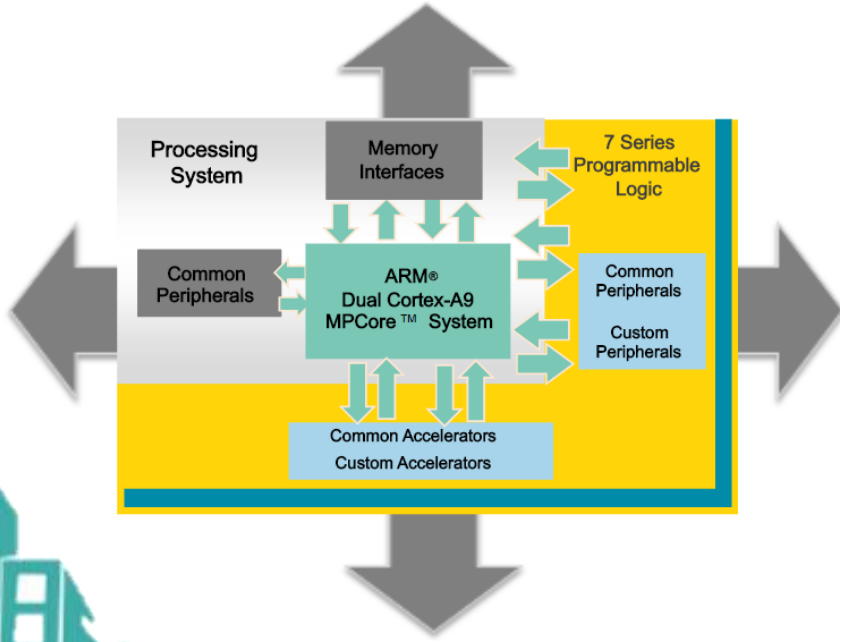
DEPARTMENT OF ELECTRONICS & COMMUNICATIONS ENGINEERING

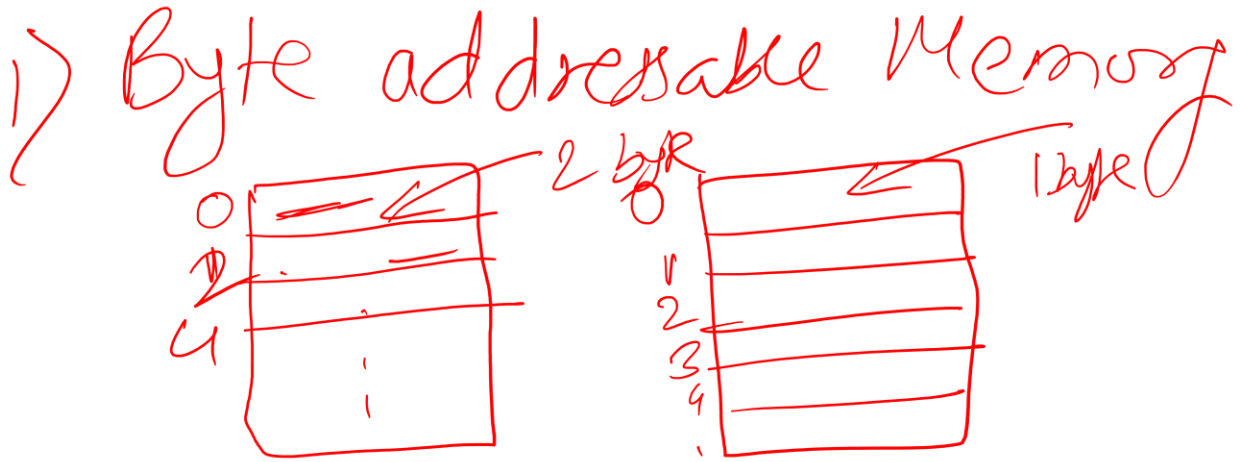
A2A
Algorithms to Architecture

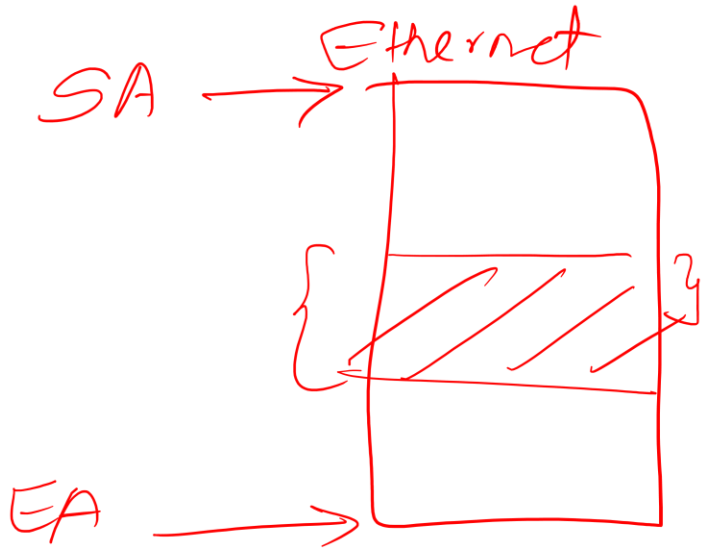
ECE 270: Embedded Logic Design

SDK
Software Development Kit
VIVADO™

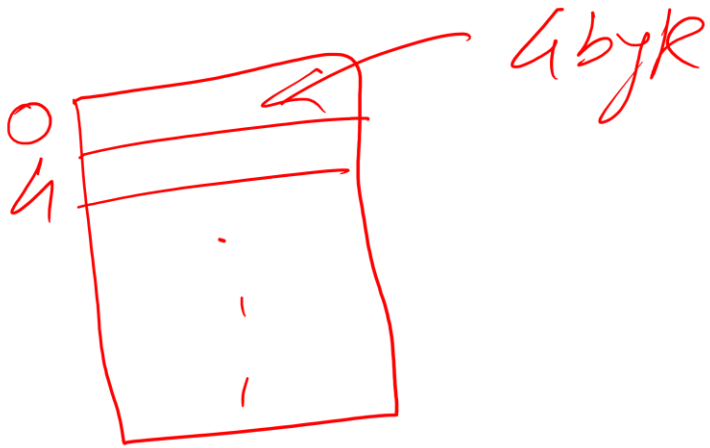


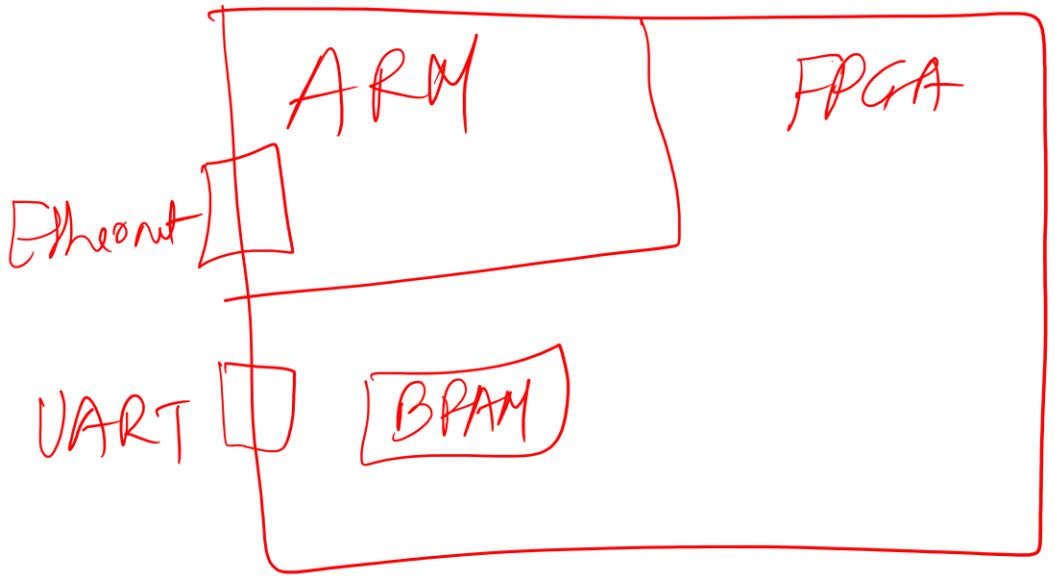






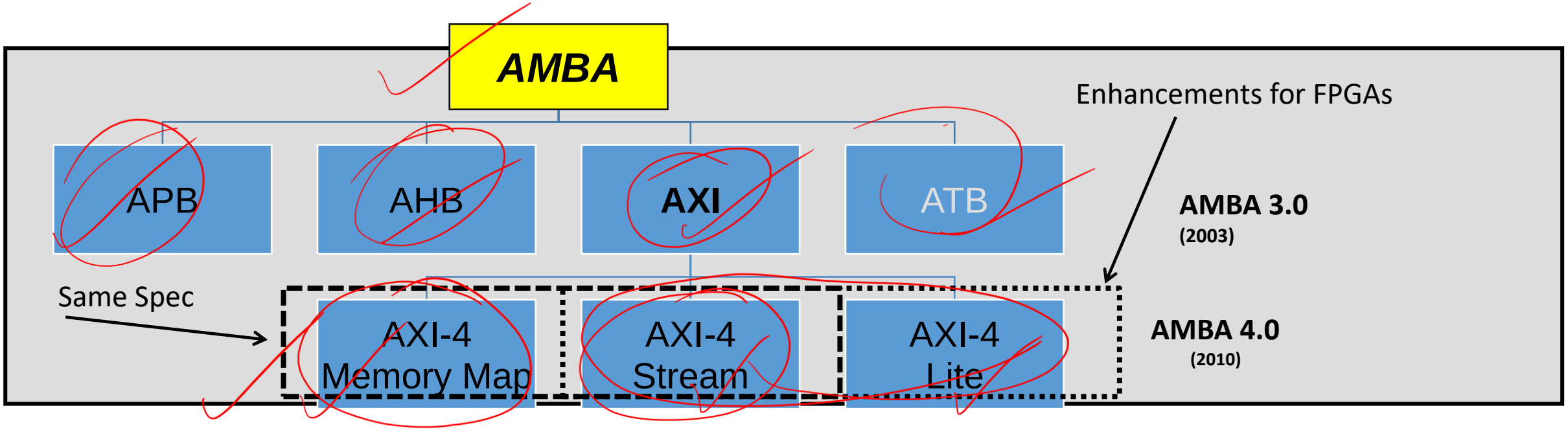
Advanced eXtensible Interface (AXI)





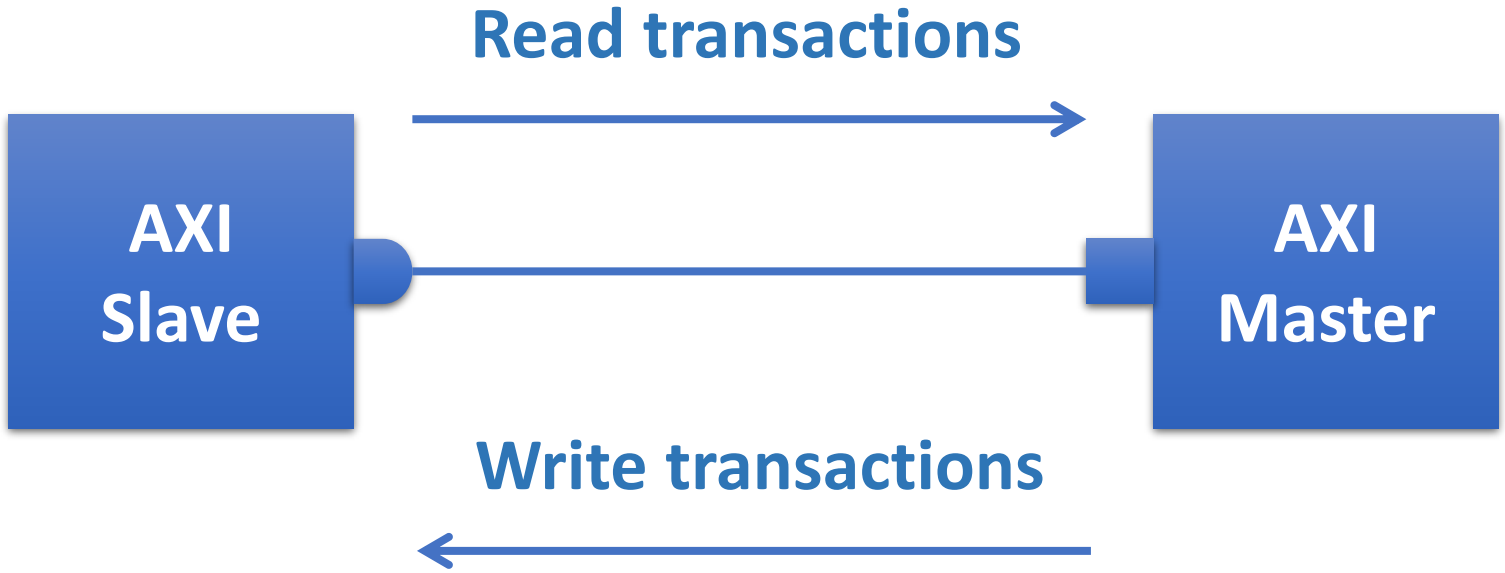
AXI

- Advanced eXtensible Interface (AXI): part of AMBA (advanced microcontroller bus architecture)



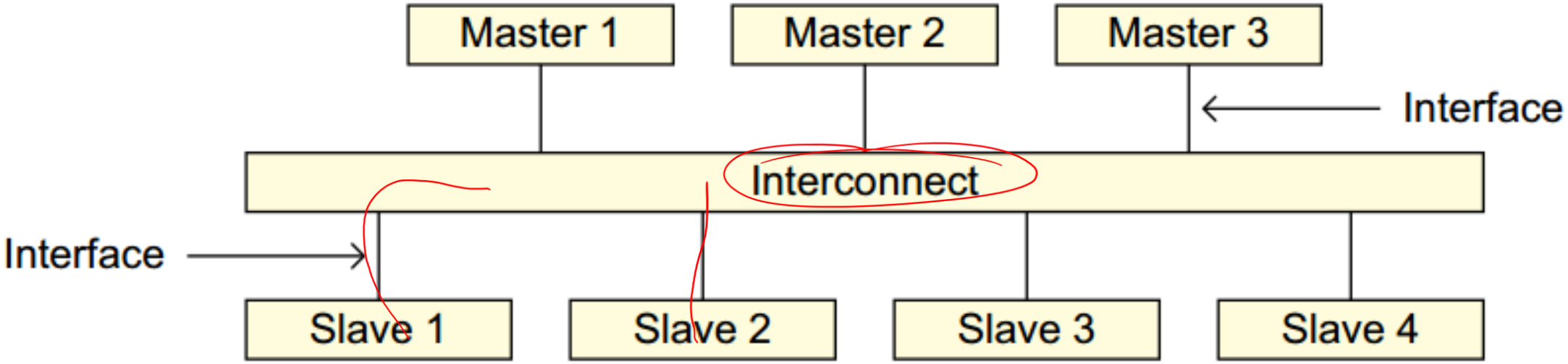
AXI

- Every AXI link contains two part: AXI master and AXI slave.
- AXI master initializes the transactions such as read and write. AXI slave is the one who responds to AXI master transactions.
- **Transaction:** Transfer of data from one point to another point



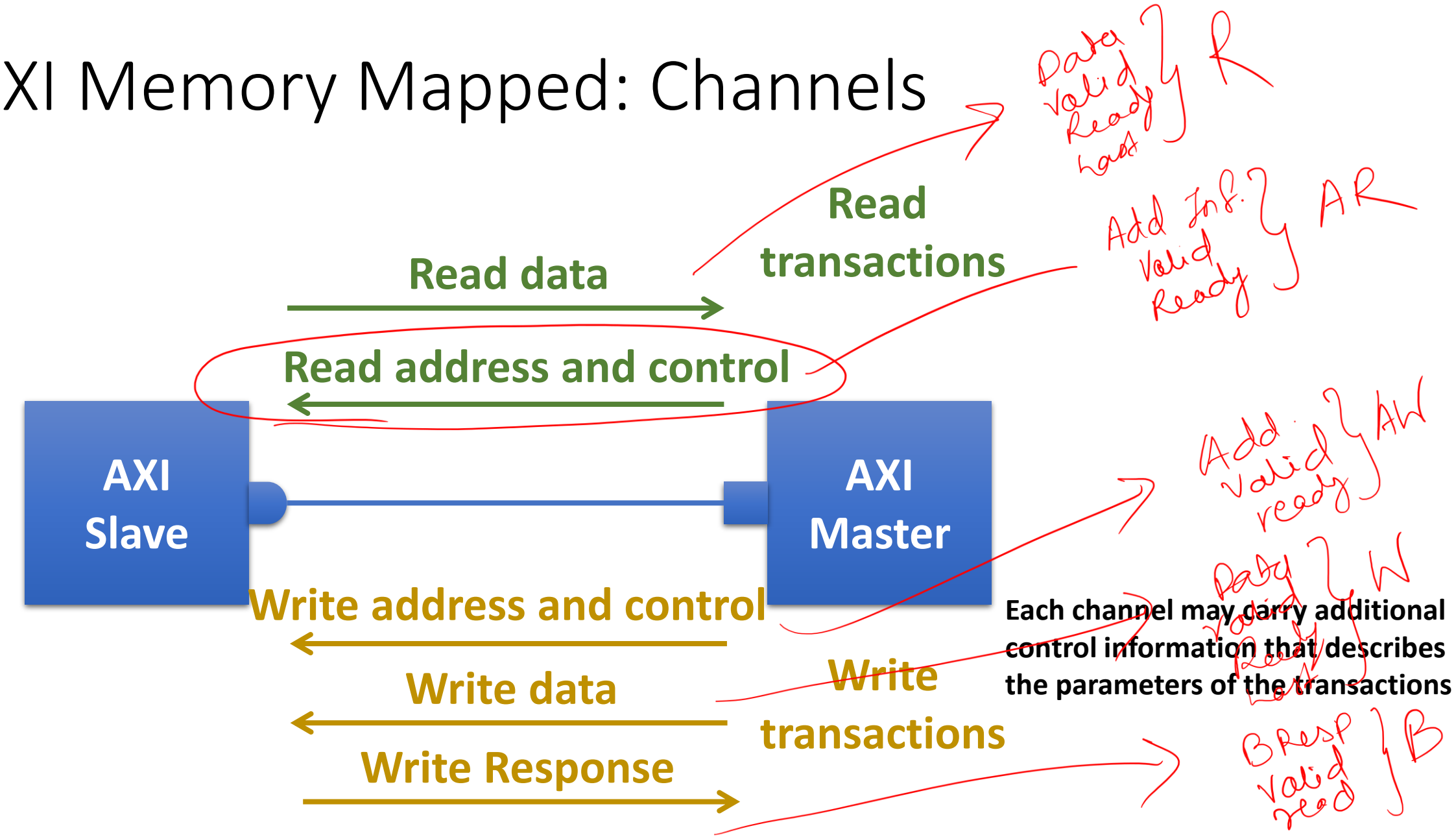
The AXI protocol provides a single interface definition, for the interfaces:

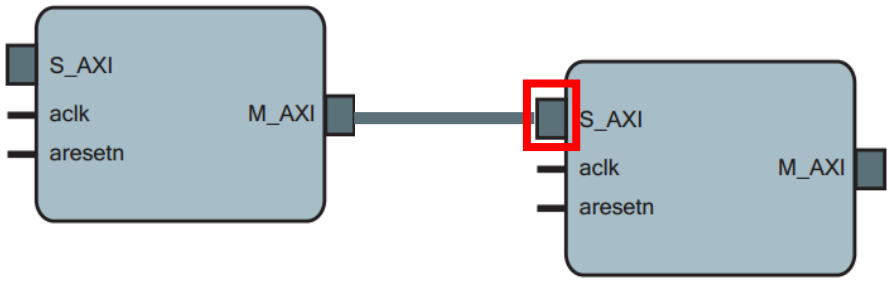
- between a master and the interconnect
- between a slave and the interconnect
- between a master and a slave.

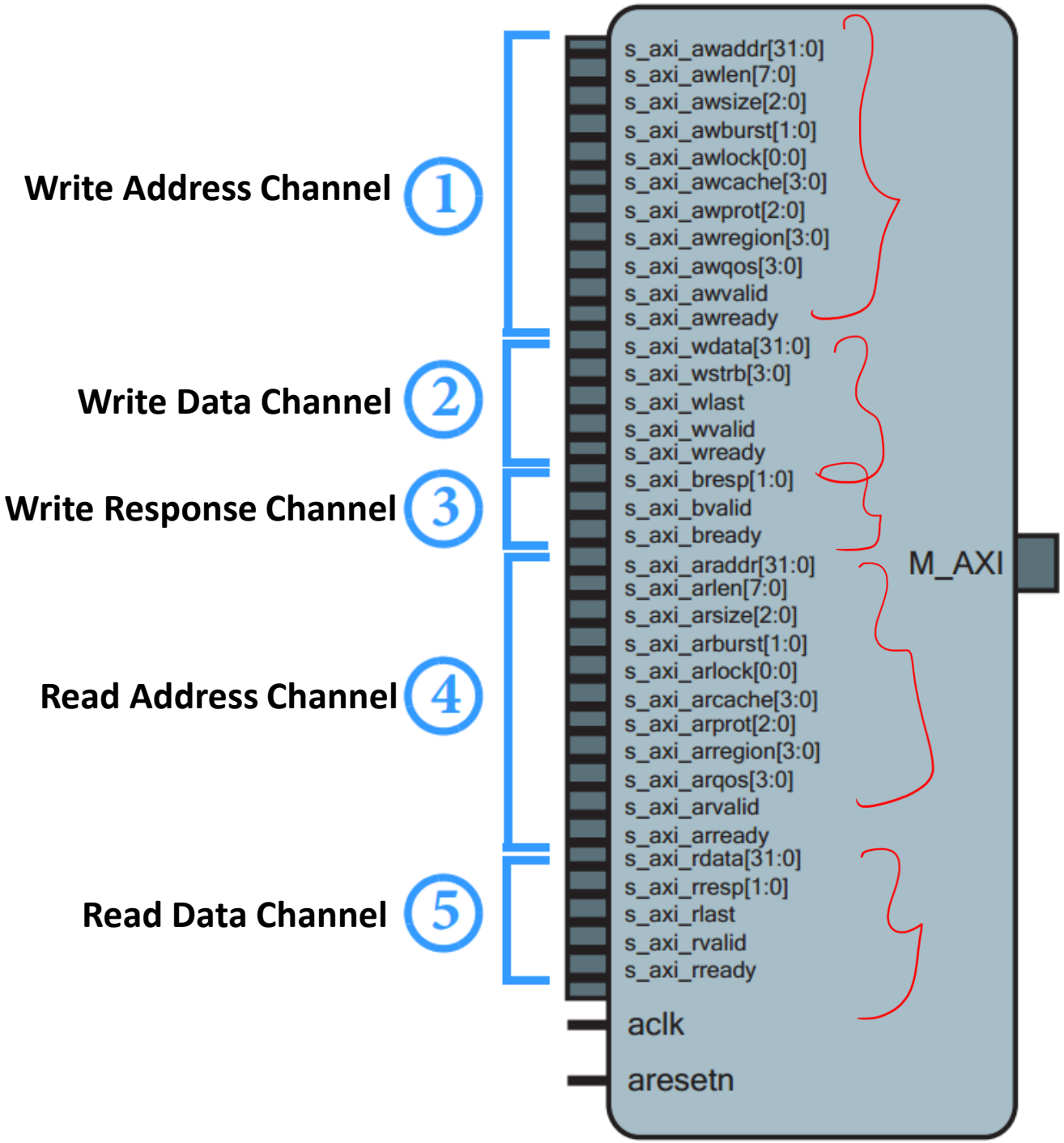


AXI Memory Mapped

AXI Memory Mapped: Channels




S_AXI
aclk
aresetn
M_AXI
S_AXI
aclk
aresetn
M_AXI

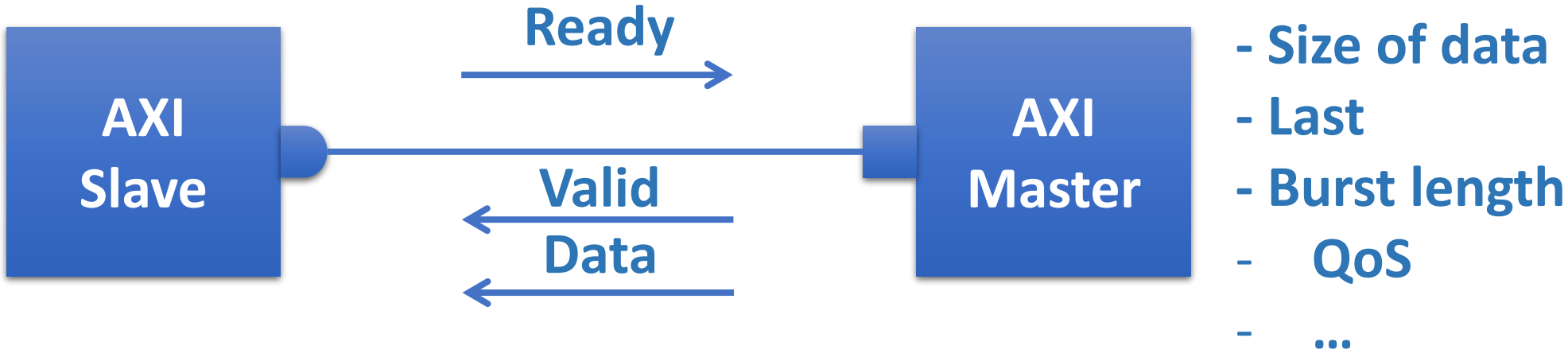

Write Address Channel
1
Write Data Channel
2
Write Response Channel
3
Read Address Channel
4
Read Data Channel
5
s_axi_awaddr[31:0]
s_axi_awlen[7:0]
s_axi_awsize[2:0]
s_axi_awburst[1:0]
s_axi_awlock[0:0]
s_axi_awcache[3:0]
s_axi_awprot[2:0]
s_axi_awregion[3:0]
s_axi_awqos[3:0]
s_axi_awvalid
s_axi_awready
s_axi_wdata[31:0]
s_axi_wstrb[3:0]
s_axi_wlast
s_axi_wvalid
s_axi_wready
s_axi_bresp[1:0]
s_axi_bvalid
s_axi_bready
s_axi_araddr[31:0]
s_axi_arlen[7:0]
s_axi_arsize[2:0]
s_axi_arburst[1:0]
s_axi_arlock[0:0]
s_axi_arcache[3:0]
s_axi_arprot[2:0]
s_axi_arregion[3:0]
s_axi_arqos[3:0]
s_axi_arvalid
s_axi_arready
s_axi_rdata[31:0]
s_axi_rresp[1:0]
s_axi_rlast
s_axi_rvalid
s_axi_rready
aclk
aresetn
M_AXI

AXI Memory Mapped

Transaction channel	Handshake pair
Write address channel	**AWVALID, AWREADY**
Write data channel	**WVALID, WREADY**
Write response channel	**BVALID, BREADY**
Read address channel	**ARVALID, ARREADY**
Read data channel	**RVALID, RREADY**

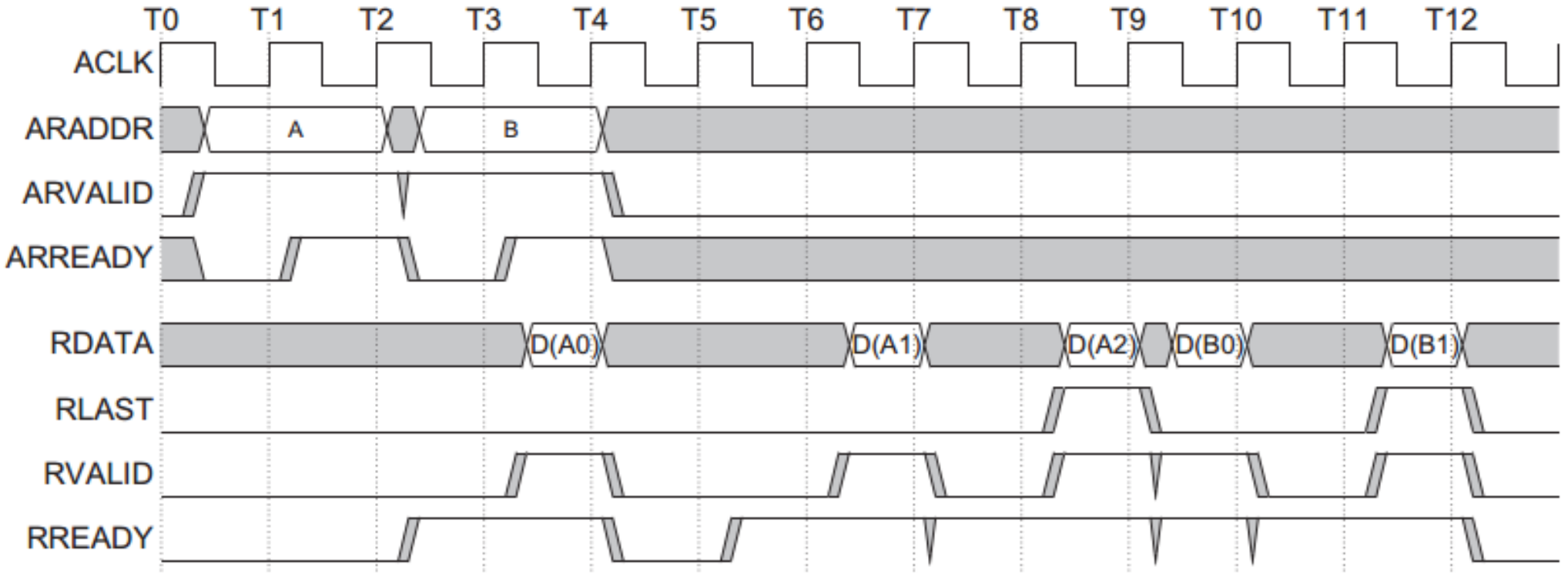
Read data channel: The slave can assert the RVALID signal only when it drives valid read data. When asserted, RVALID must remain asserted until the rising clock edge after the master asserts RREADY. The master interface uses the RREADY signal to indicate that it accepts the data.

AXI Memory Mapped



- Each of the independent channels consists of a set of information signals along with **VALID** and **READY** signals that provide a two-way handshake mechanism.
- The information source uses the **VALID** signal to show when valid address, data or control information is available on the channel.
- The destination uses the **READY** signal to show when it can accept the information.
- Both the read data channel and the write data channel also include a **LAST** signal to indicate the transfer of the final data item in a transaction.

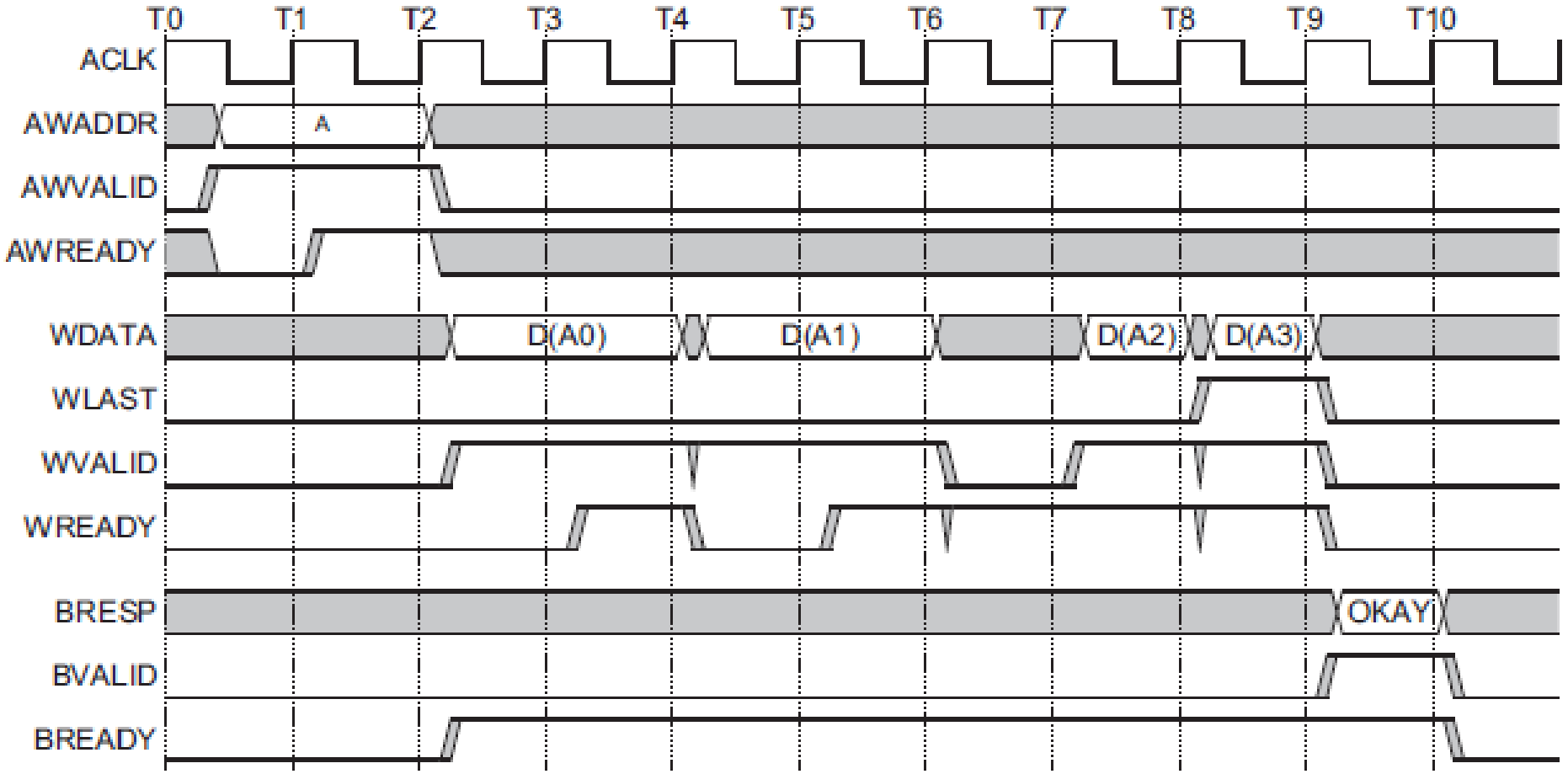
AXI Interface: Read



AXI – Burst Read

*Other control signals are not discussed here: ARID, ARLEN, ARSIZE, RID, RRESP,.....

AXI Interface: Write



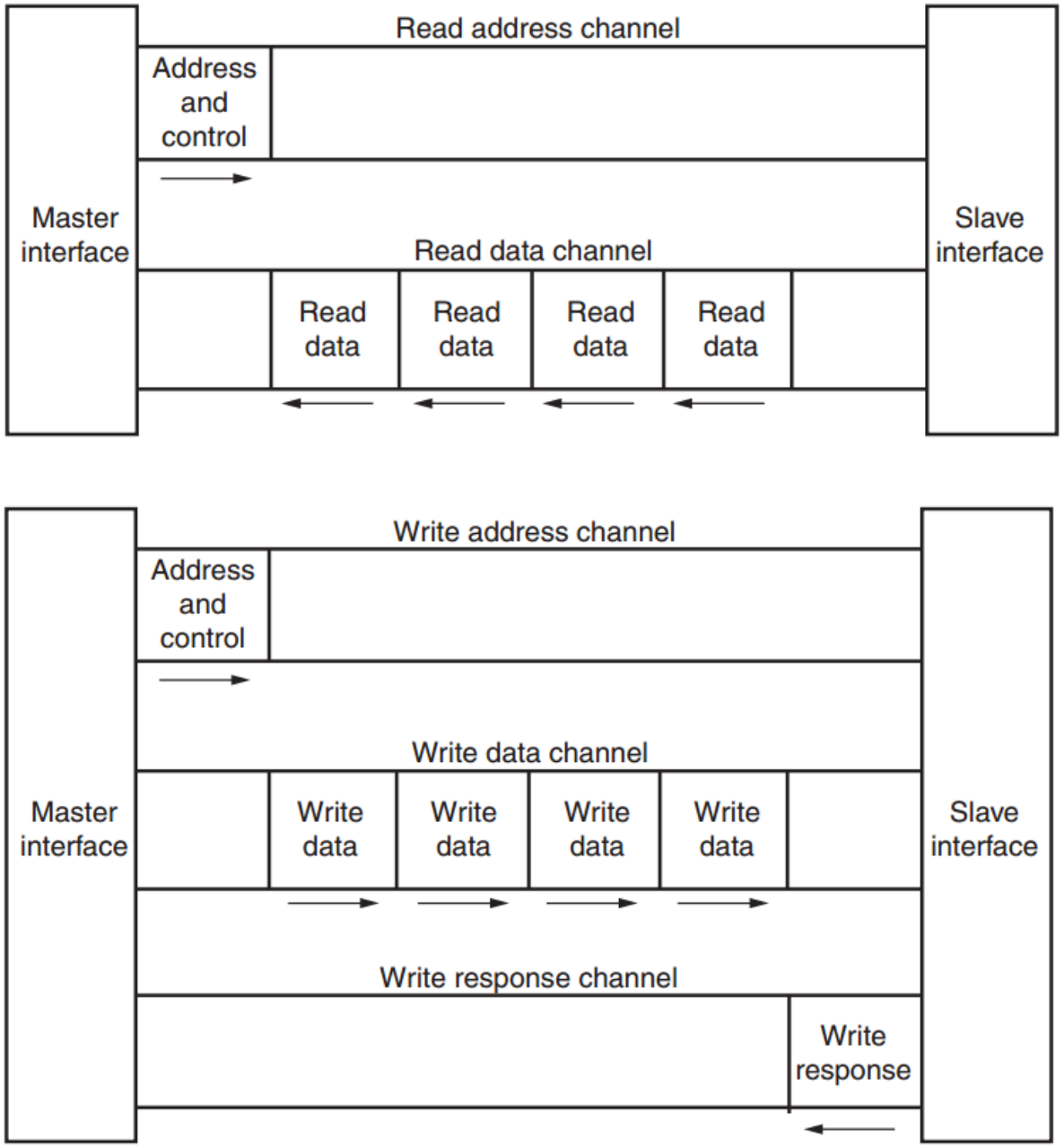
AXI Burst Write

AXI Memory Mapped: Channels

- AXI4 provides separate data and address/control connections for Reads and Writes, which allows simultaneous, bidirectional data transfer.
- Whether you're doing a read or write, an AXI4 transaction is the same. An address comes out, then the data is either written by the AXI master or provided by the AXI slave.
- The extra channel, Write Response, allows the AXI slave to tell the AXI master, in a write transaction, status on the transaction.

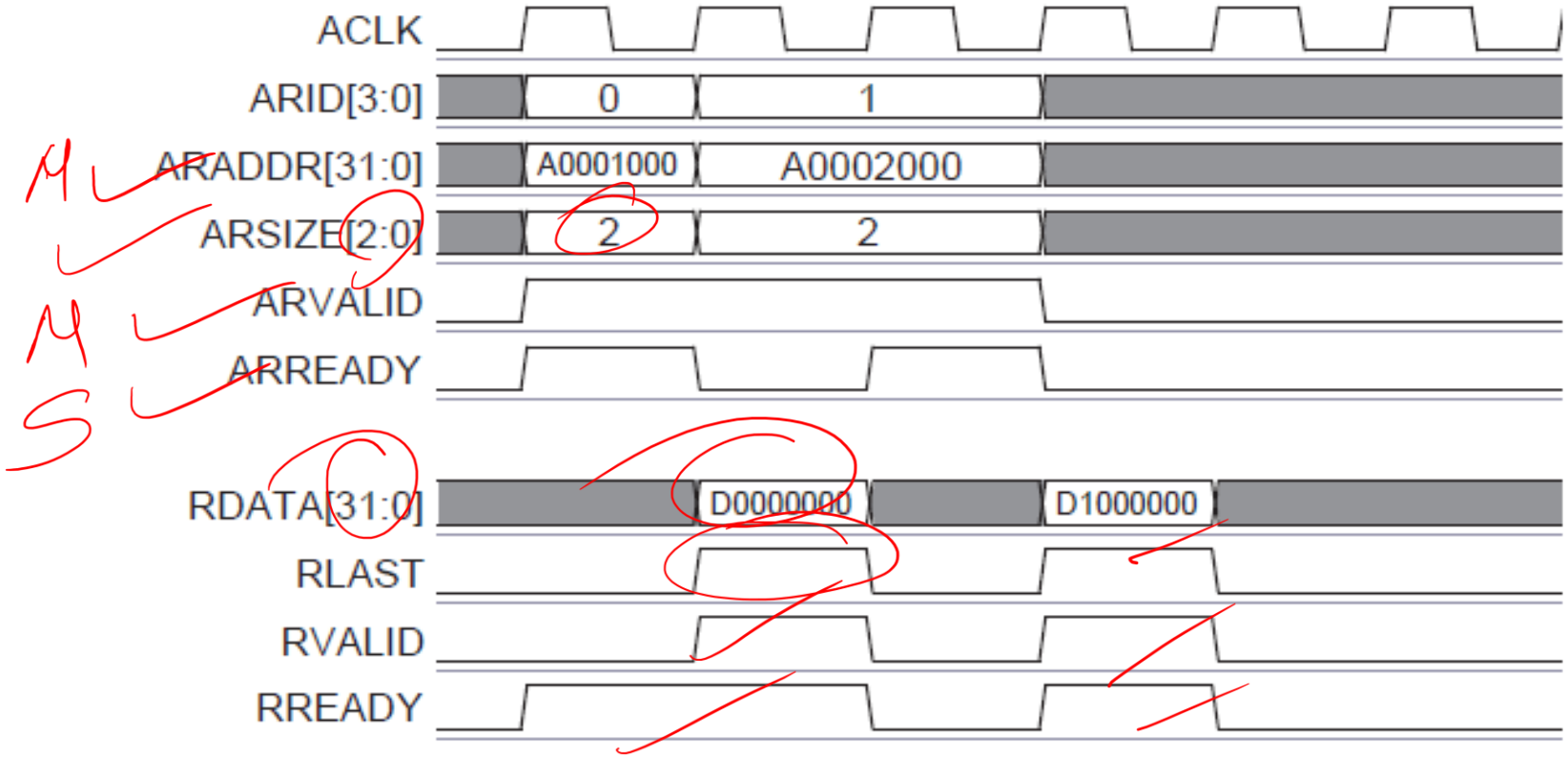
AXI Memory Mapped

- Bursts varying from 1 to 256 data transfers per burst
- Each burst transfer size can be 1,2,4,8..128 bytes
- **Byte lane strobe signal** for every eight data bits, indicating which bytes of the data are valid



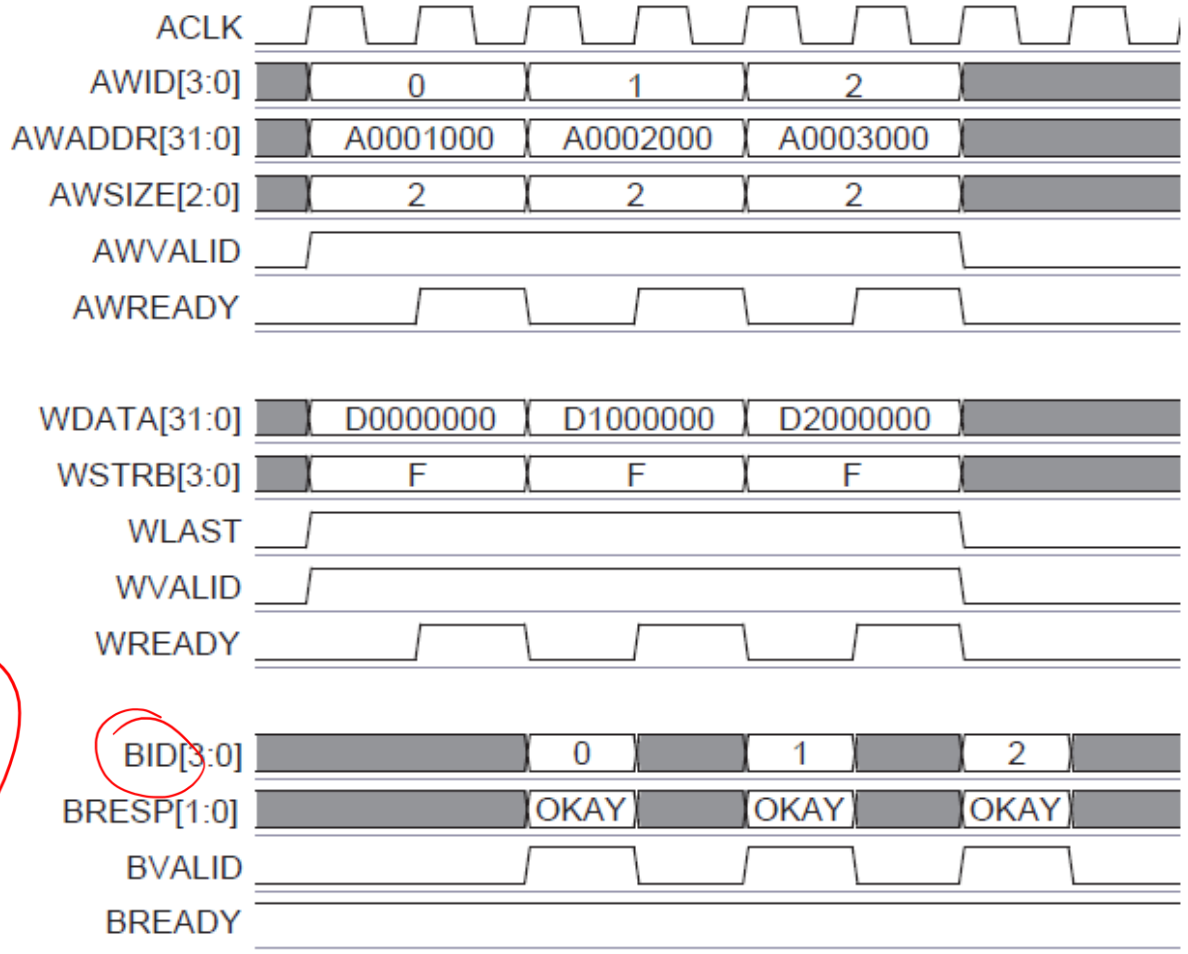
AXI Interface: Read

AxSIZE[2:0]	Bytes in transfer
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128



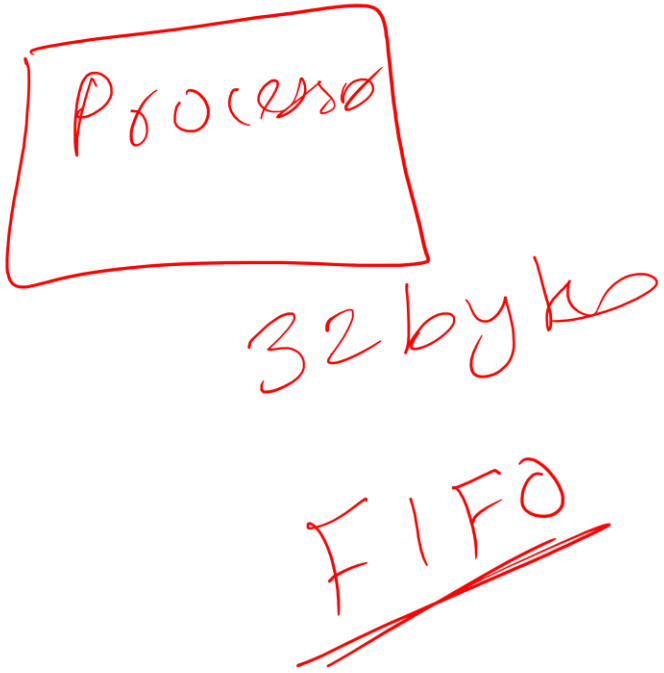
AXI Interface: Write

AxSIZE[2:0]	Bytes in transfer
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128



Example of AXI4 multiple write transactions

AXI Interface: Burst



AXI Interface: Burst

AXI Interface: Burst

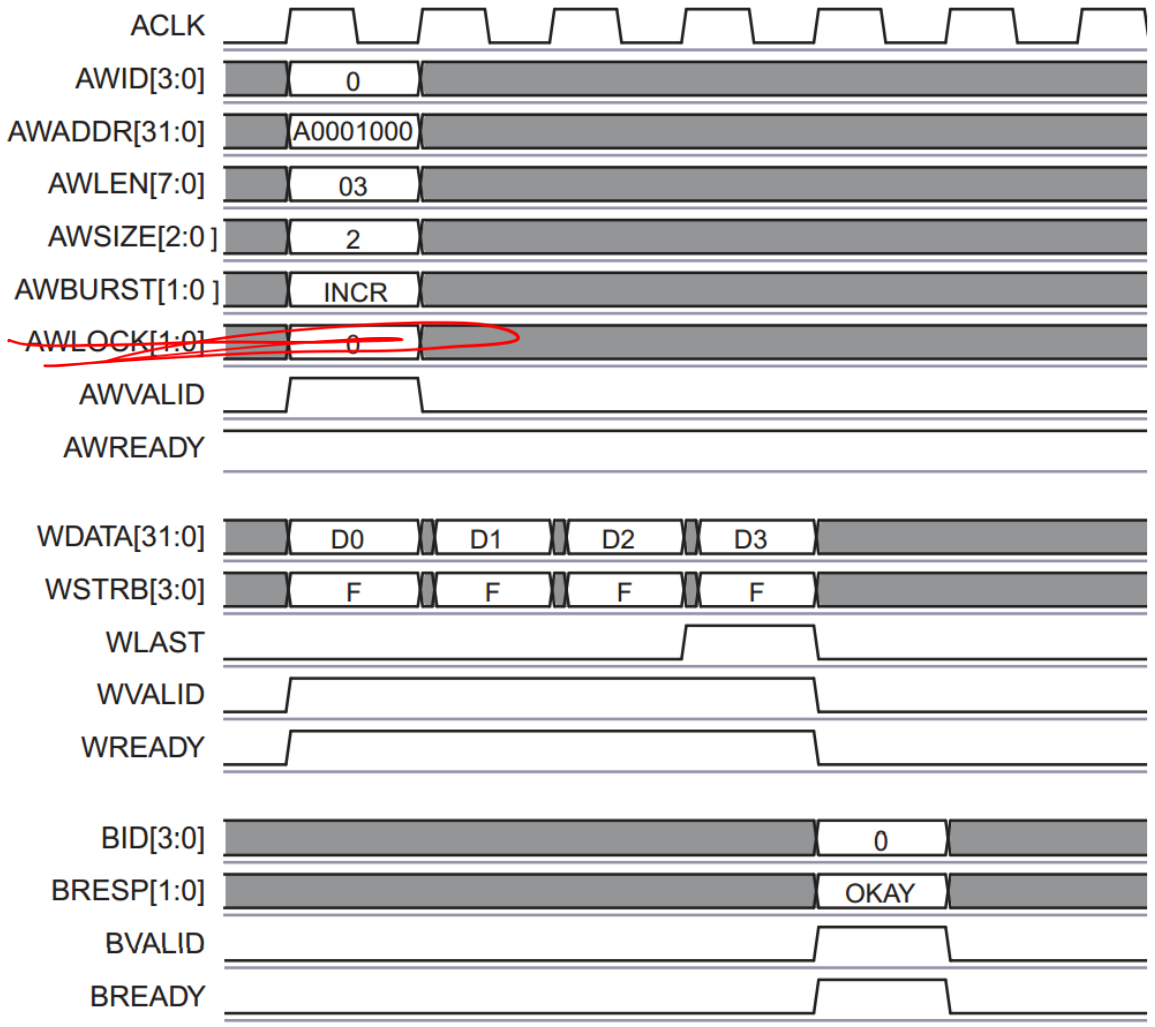
ARBURST[1:0] AWBURST[1:0]	Burst type	Description	Access
b00	FIXED	Fixed-address burst	FIFO-type
b01	INCR	Incrementing-address burst	Normal sequential memory
b10	WRAP	Incrementing-address burst that wraps to a lower address at the wrap boundary	Cache line
b11	Reserved	-	-

AXI Interface: Write

AxSIZE[2:0]	Bytes in transfer
0b000	1
0b001	2
0b010	4
0b011	8
0b100	16
0b101	32
0b110	64
0b111	128

The burst length for AXI4 is defined as,

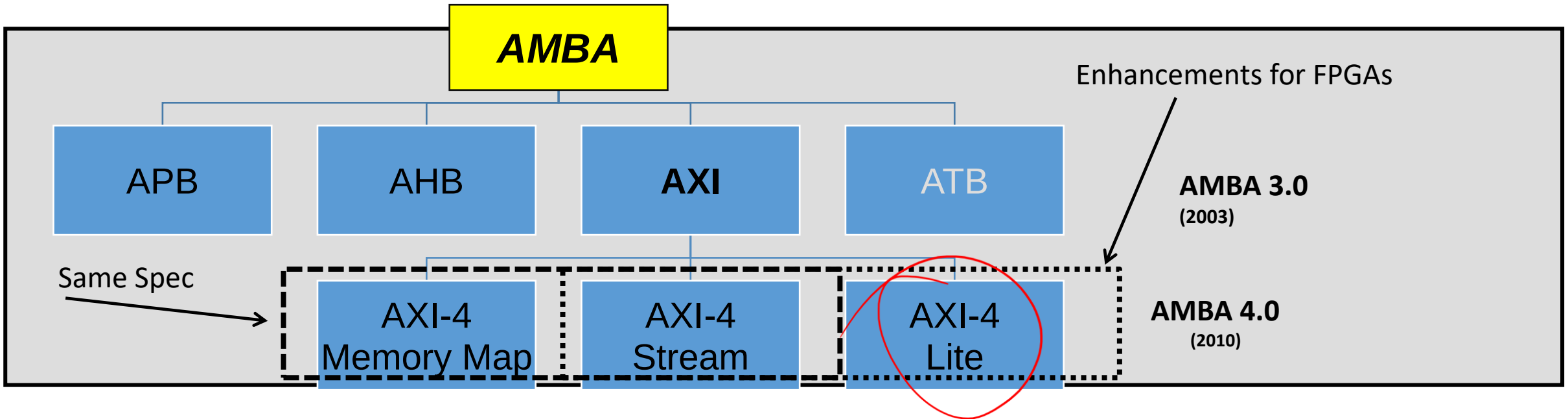
Burst_Length = **AxLEN[7:0]** + 1



Example of AXI4 write burst transaction

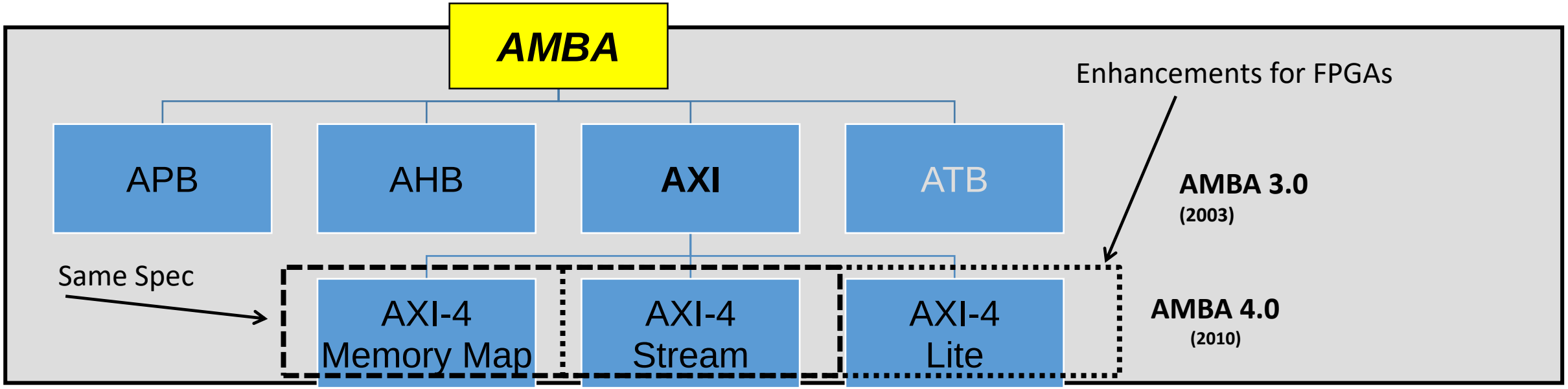
AXI

- **AXI4-memory mapped:**
  - Single address, multiple data
  - High-performance interface
  - Suited for memory mapped communication allowing bursts of up to 256 data transfer cycles per address phase



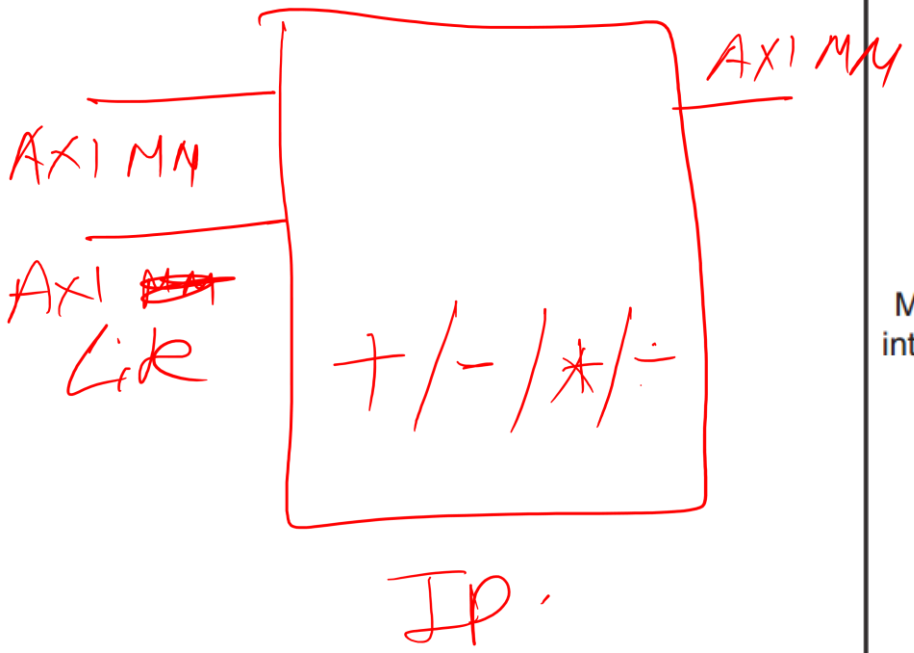
AXI

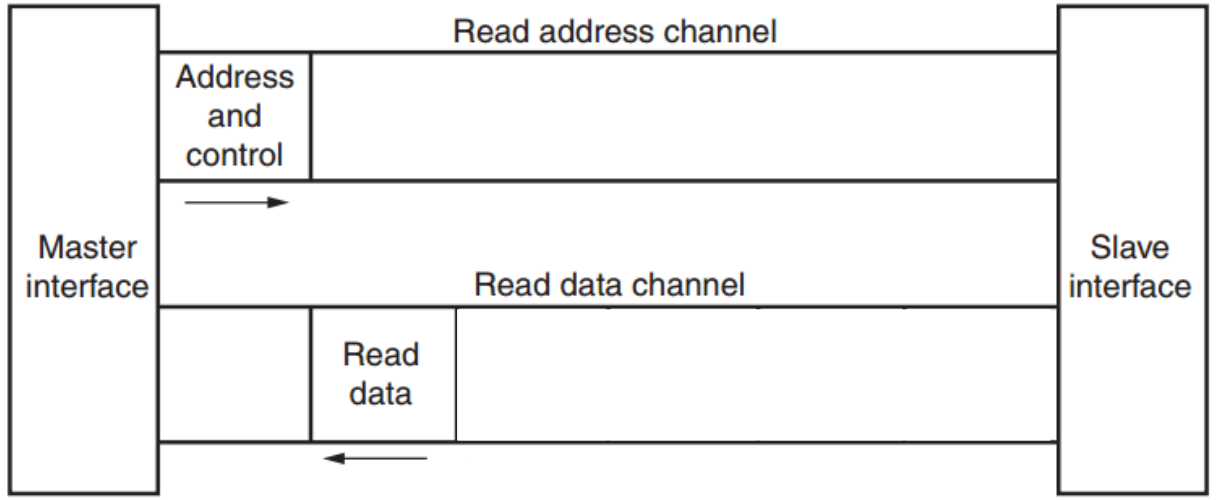
- **AXI4-Lite:**
  - Single address, single data: memory mapped single transactions
  - Does not support burst data -> Smaller logic footprint

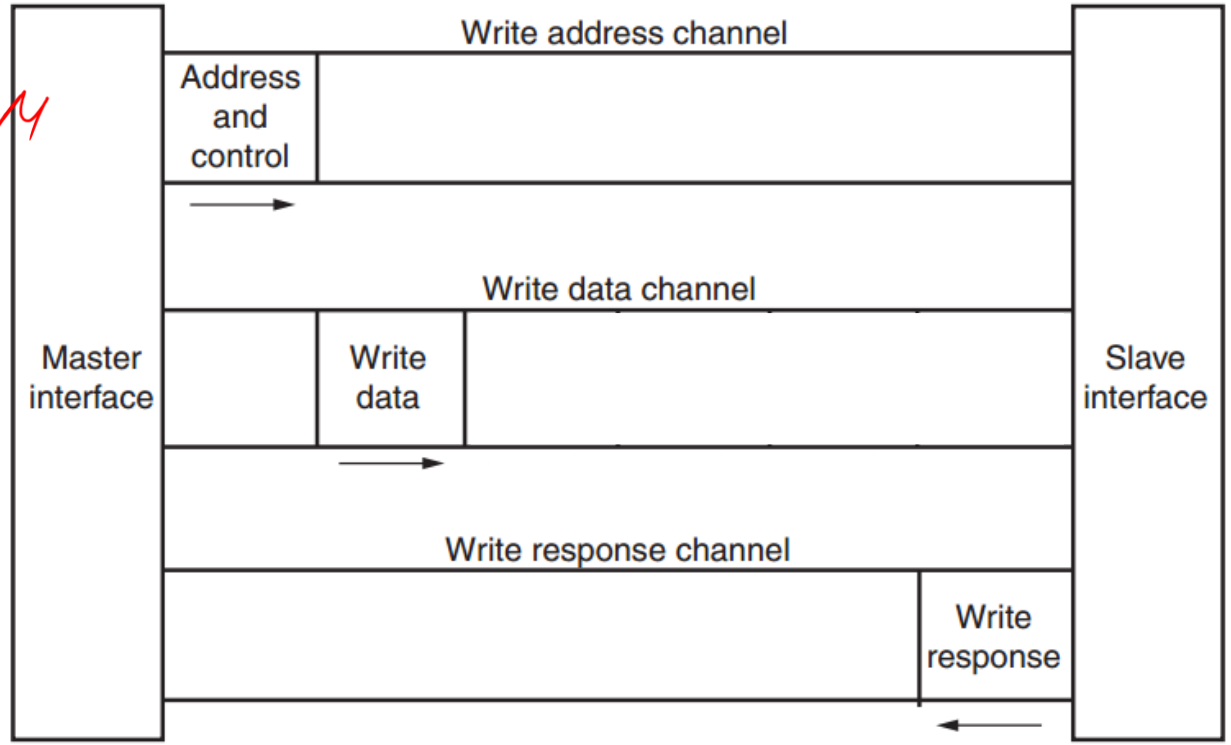


AXI Lite

- Bursting is not supported
- Subset of the AXI4 interface intended for communication with control registers and have small footprint

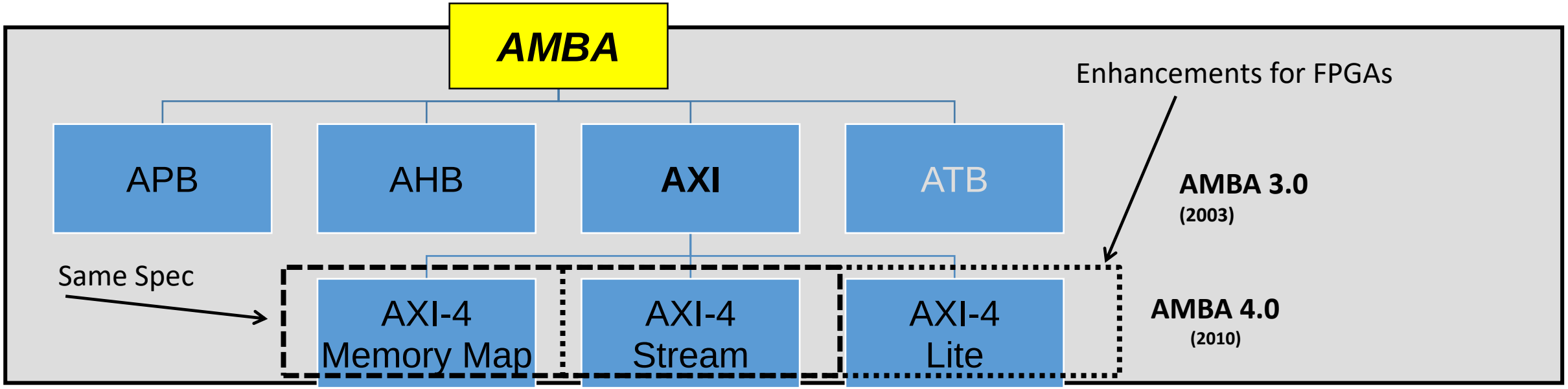






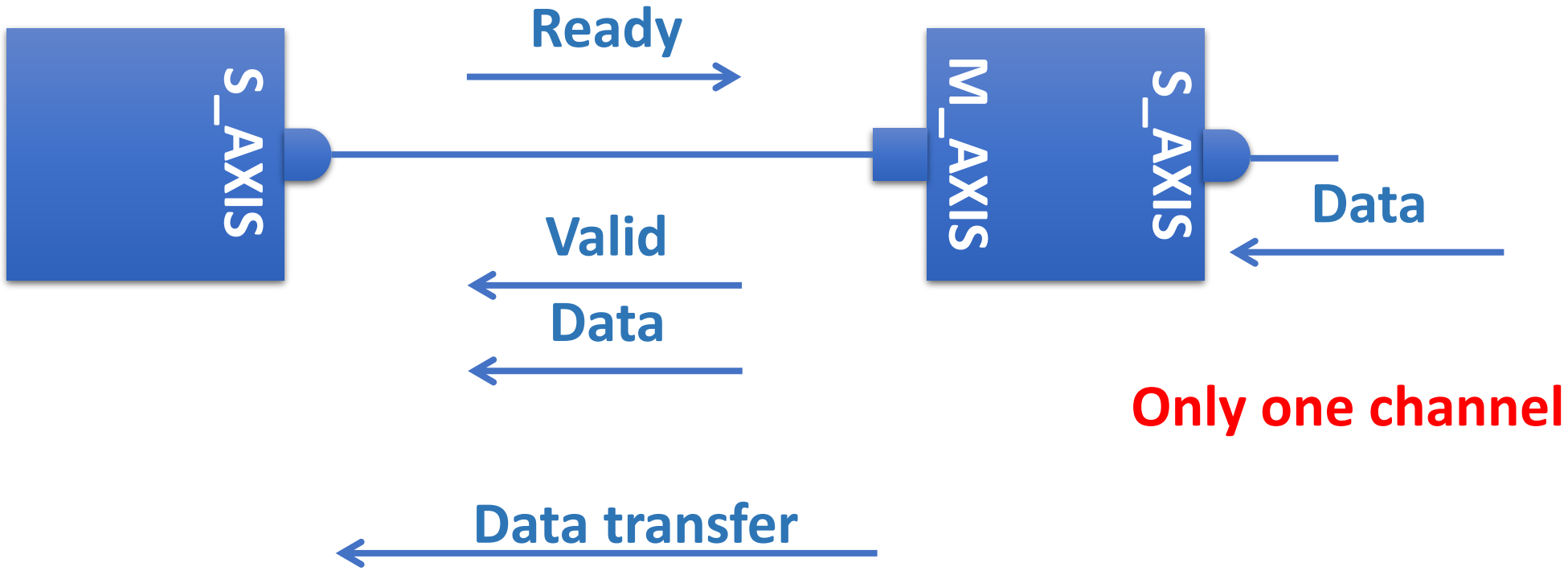
AXI

- **AXI4-Stream:**
  - No address phase means this is not memory mapped
  - Unlimited* data burst size

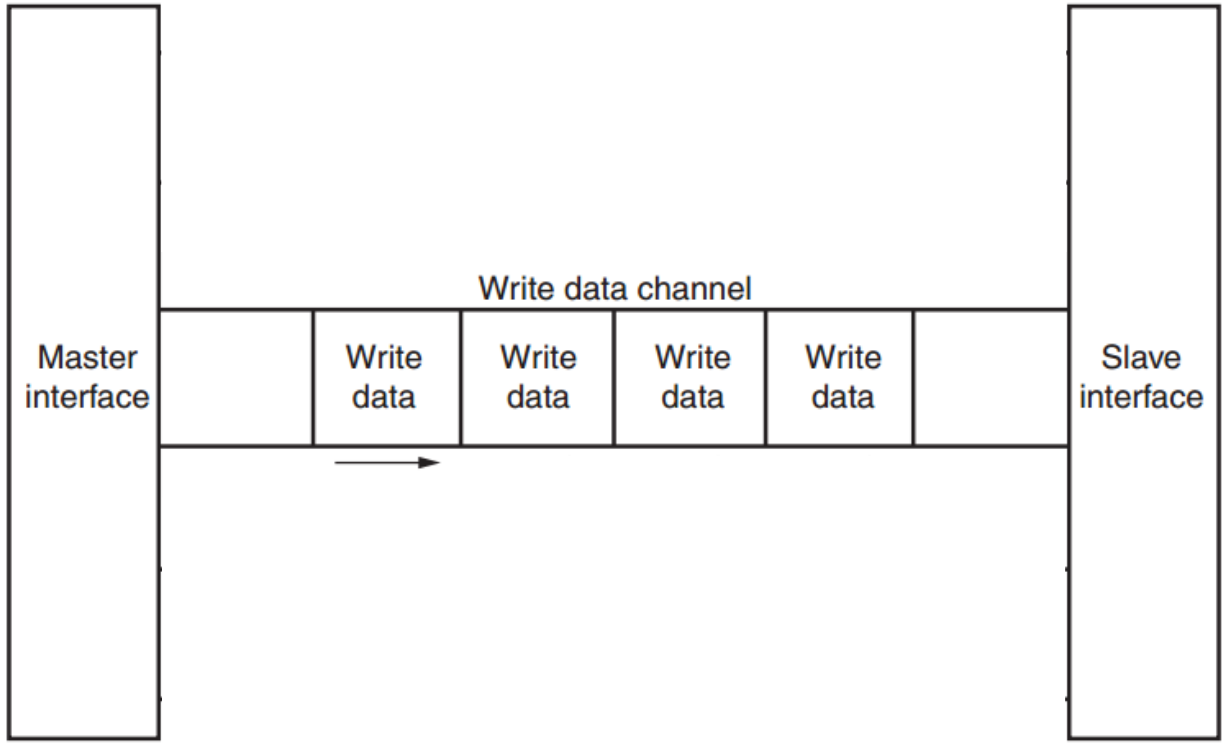


AXI Stream

- The AXI4-Stream protocol defines a single channel for transmission of streaming data (unlimited burst).

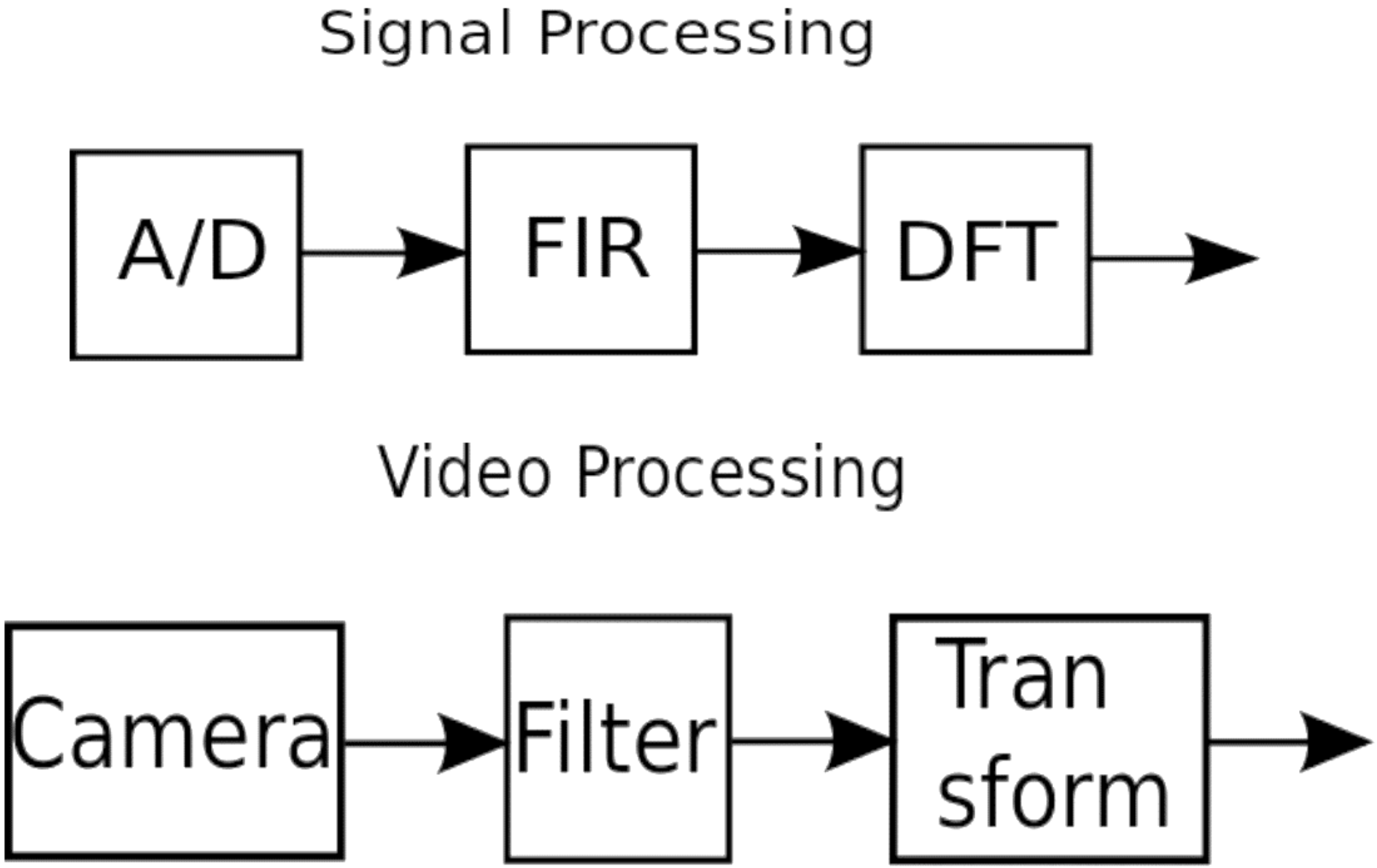


AXI Stream

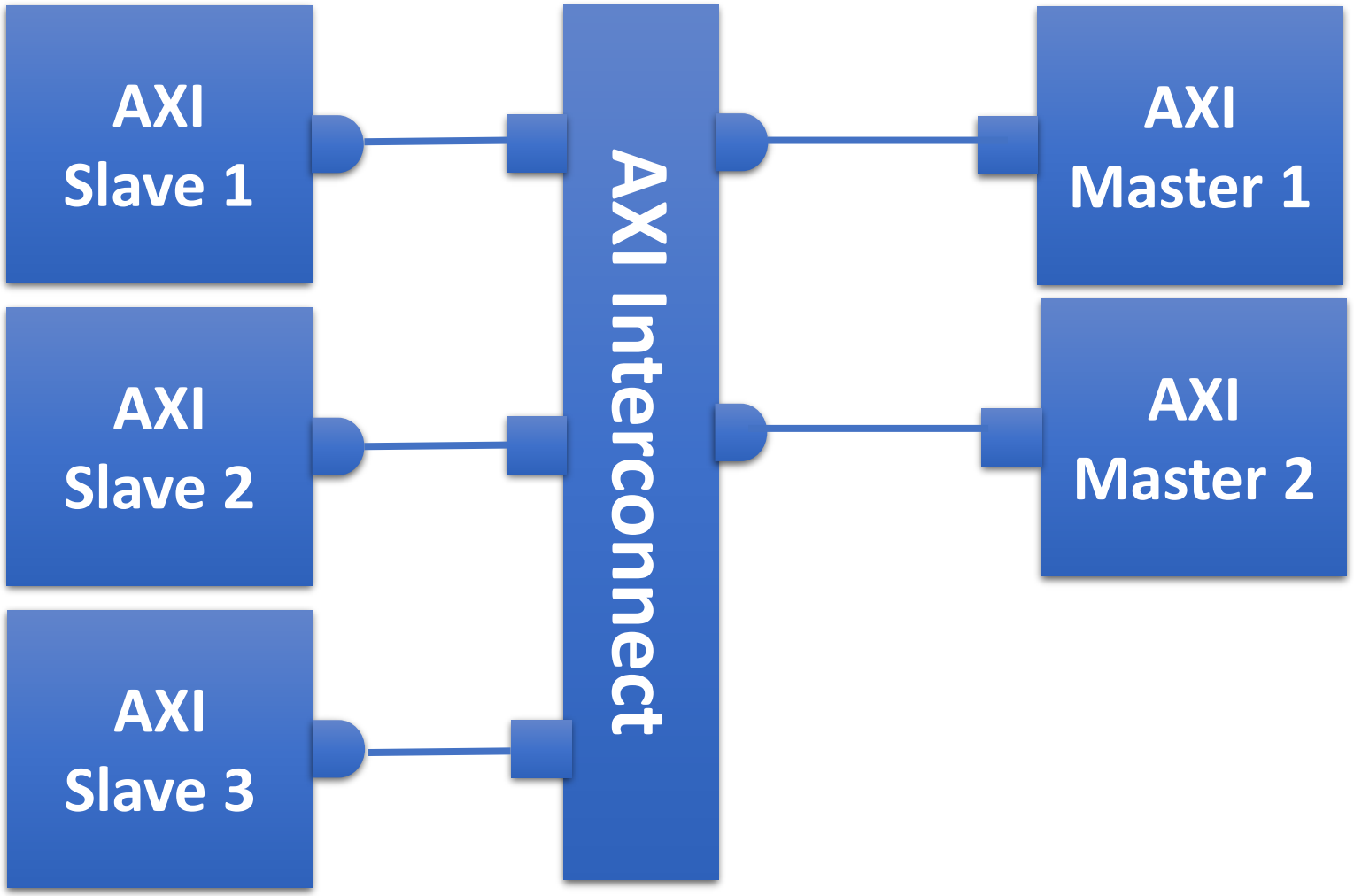


- The AXI4-Stream channel is modeled after the Write Data channel of the AXI4.
- Unlike AXI4, AXI4-Stream interfaces can burst an **unlimited** amount of data.

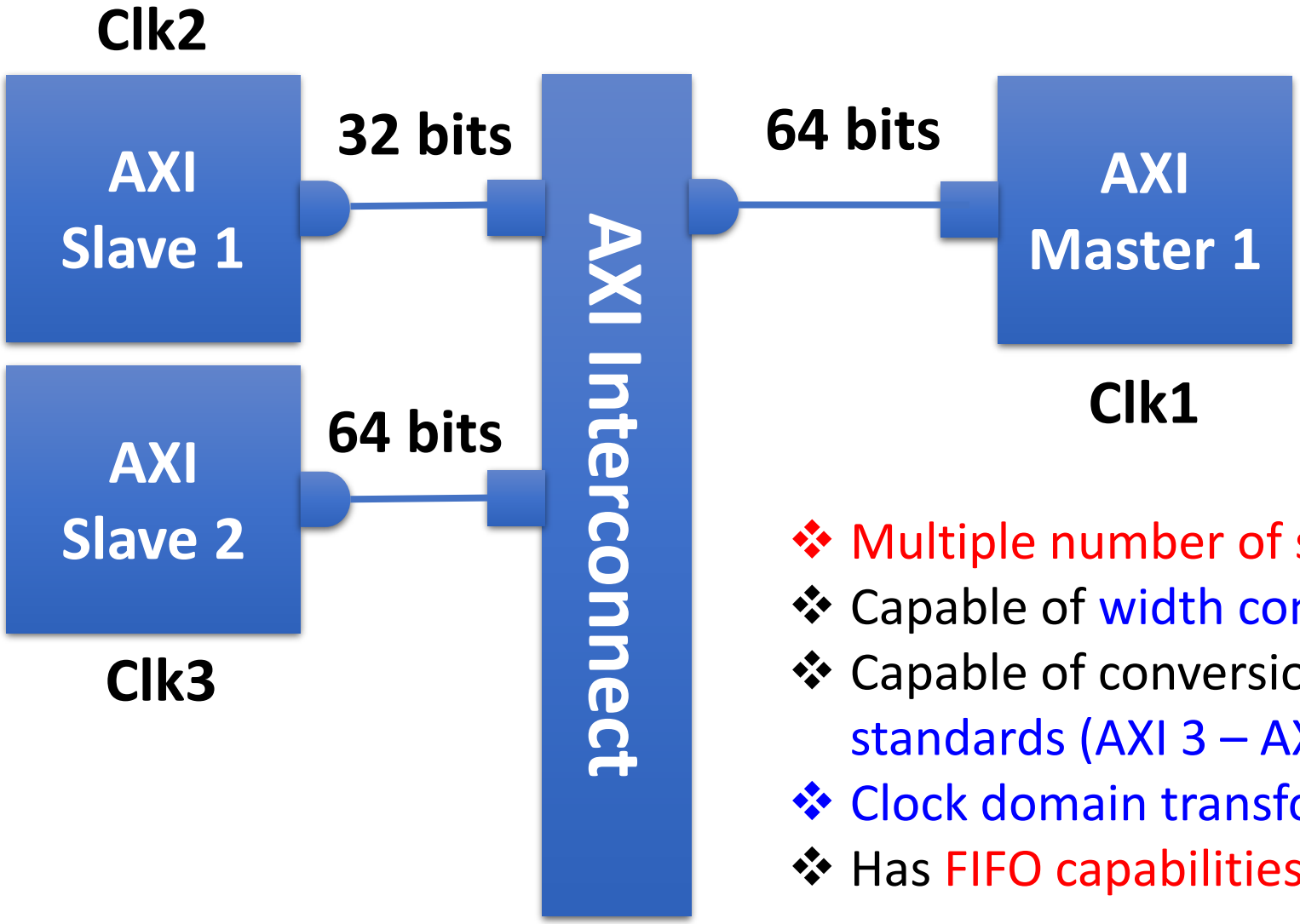
AXI Stream


Signal Processing
A/D
FIR
DFT
Video Processing
Camera
Filter
Tran
sform

AXI Interconnect

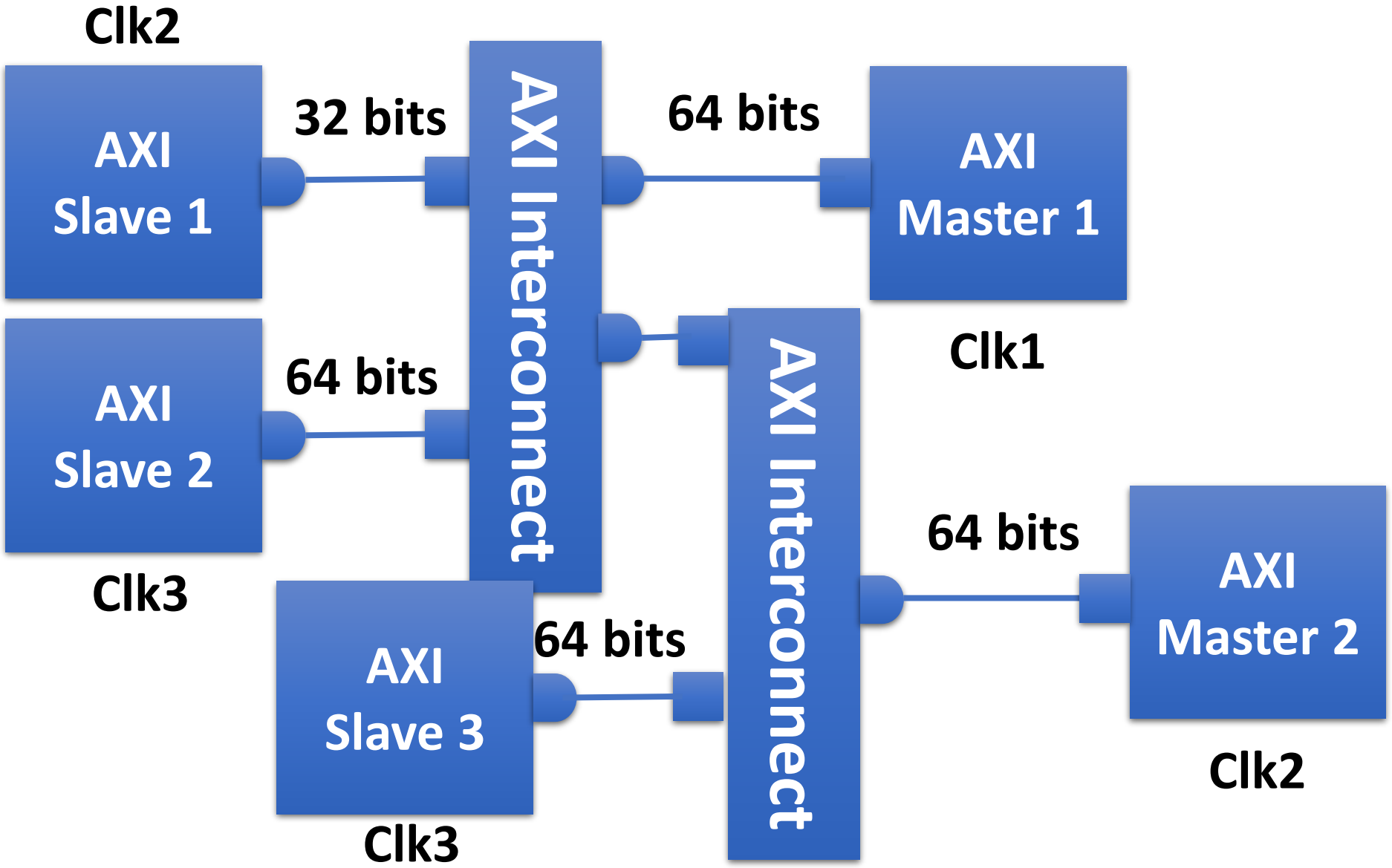


AXI Interconnect

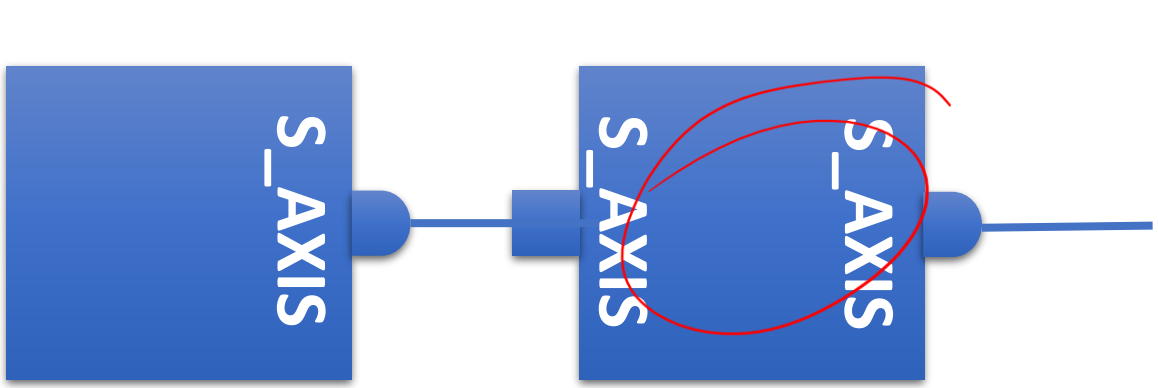


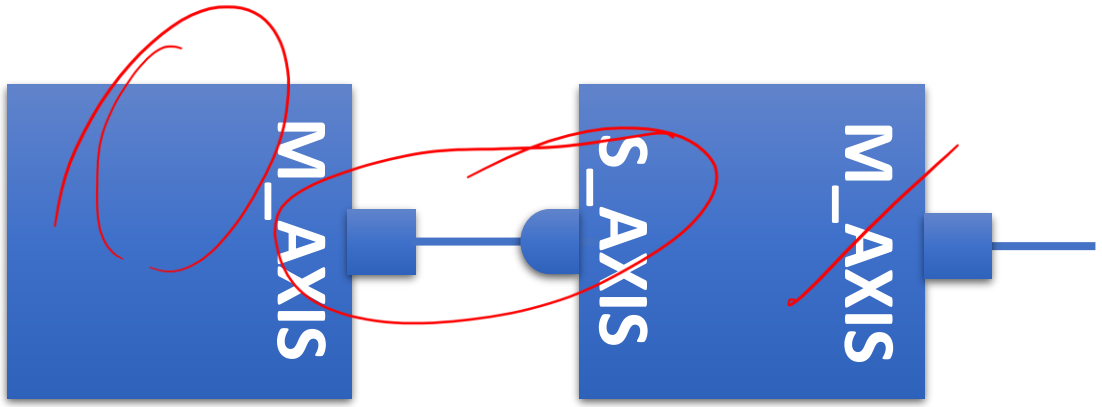
- Multiple number of slave and master ports
- Capable of width conversion
- Capable of conversion between different AXI standards (AXI 3 – AXI 4)
- Clock domain transformations
- Has FIFO capabilities, registers for pipelining

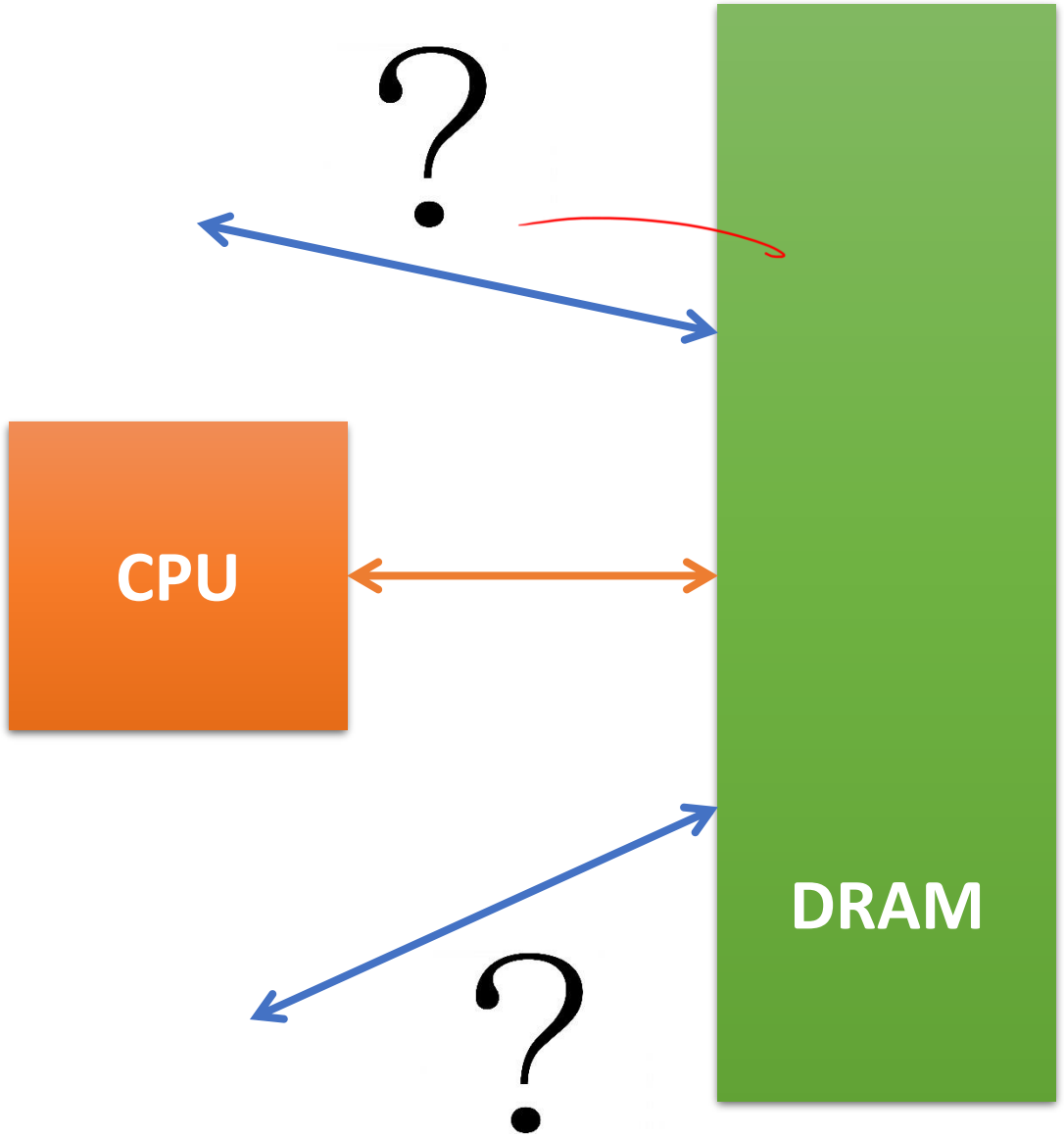
Hierarchical AXI Interconnect



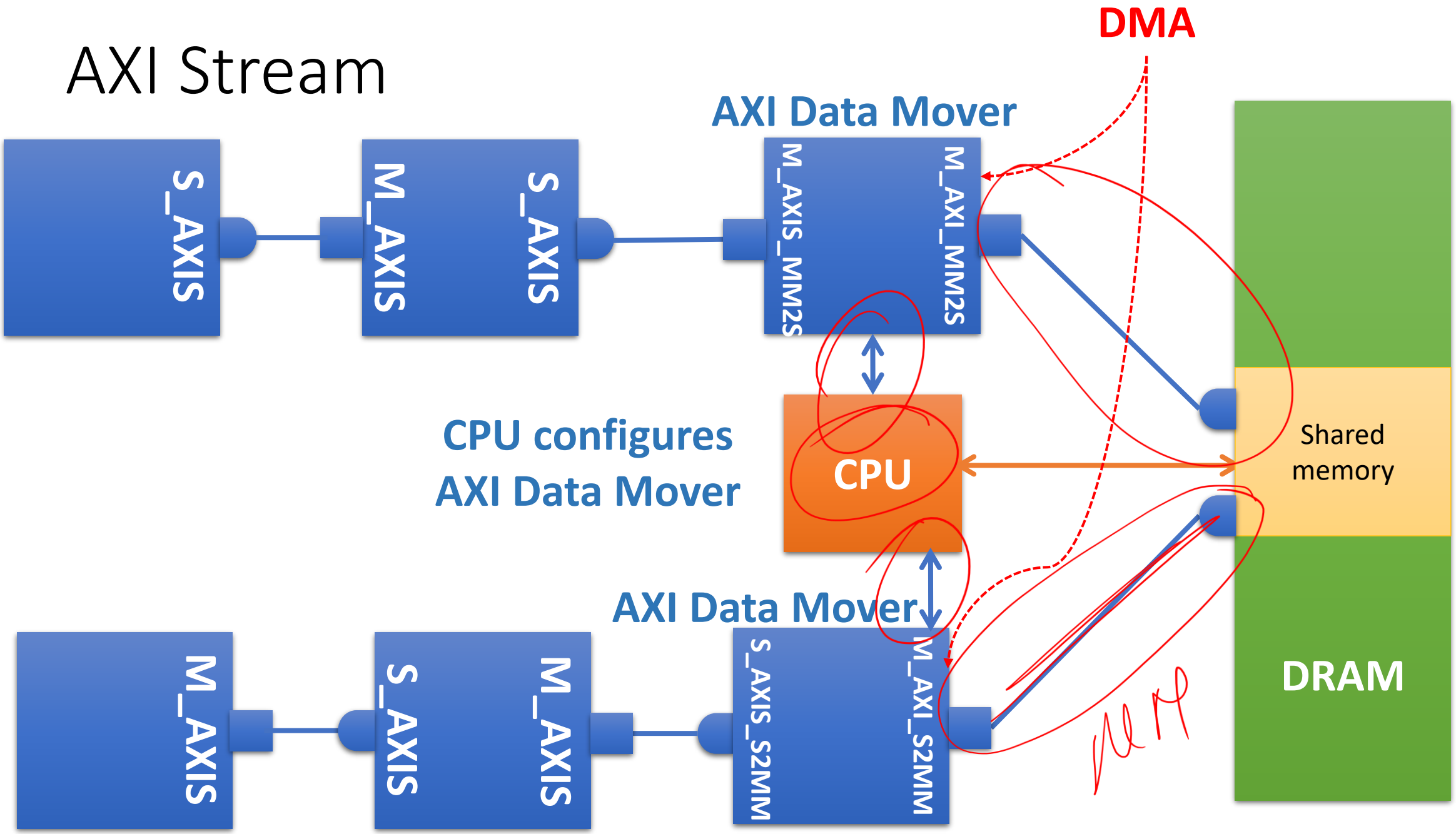
AXI Stream



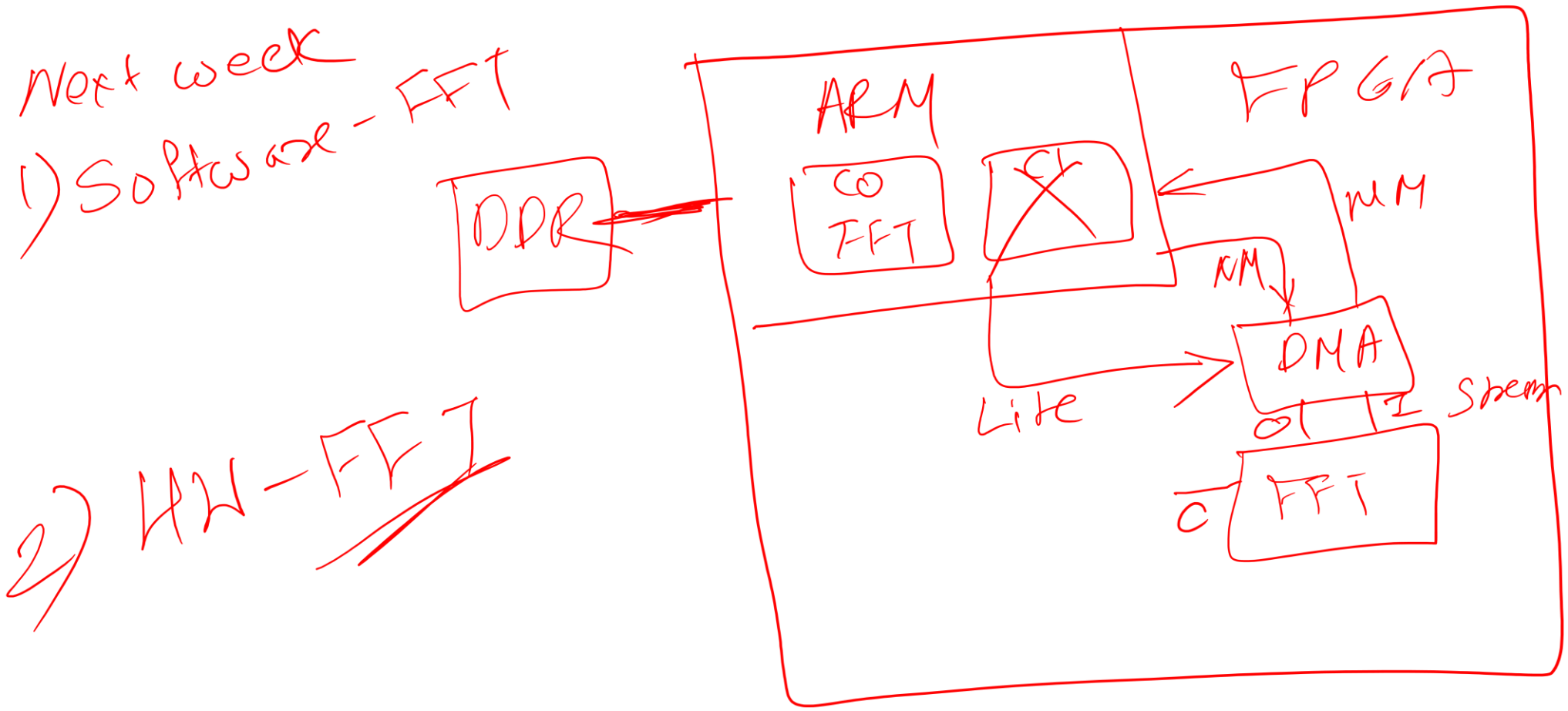


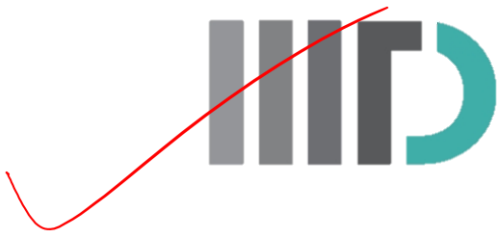


AXI Stream


DMA
AXI Data Mover
S_AXIS
M_AXIS
S_AXIS
M_AXIS_MM2S
M_AXI_MM2S
CPU configures
AXI Data Mover
CPU
Shared
memory
AXI Data Mover
M_AXIS
S_AXIS
M_AXIS
S_AXIS_S2MM
M_AXI_S2MM
DRAM

Overview of Upcoming labs



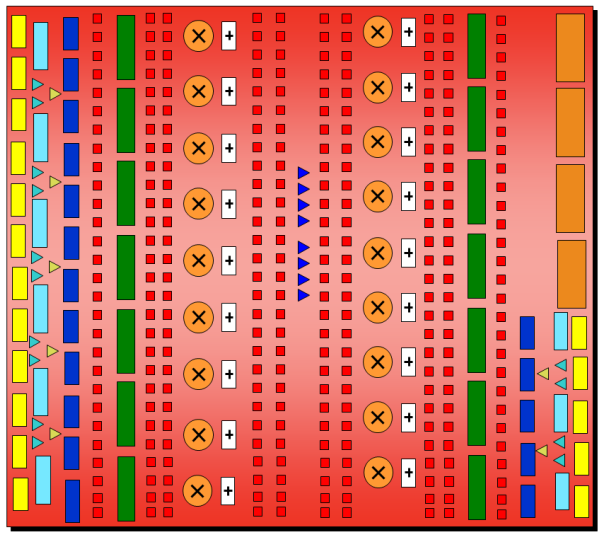


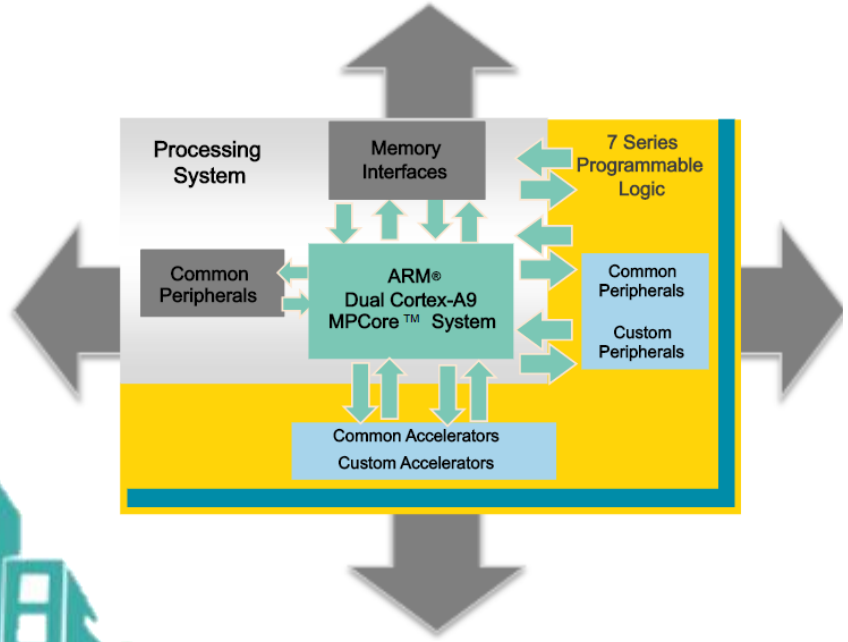
ECE | DEPARTMENT OF ELECTRONICS & COMMUNICATIONS ENGINEERING

A2A
Algorithms to Architecture

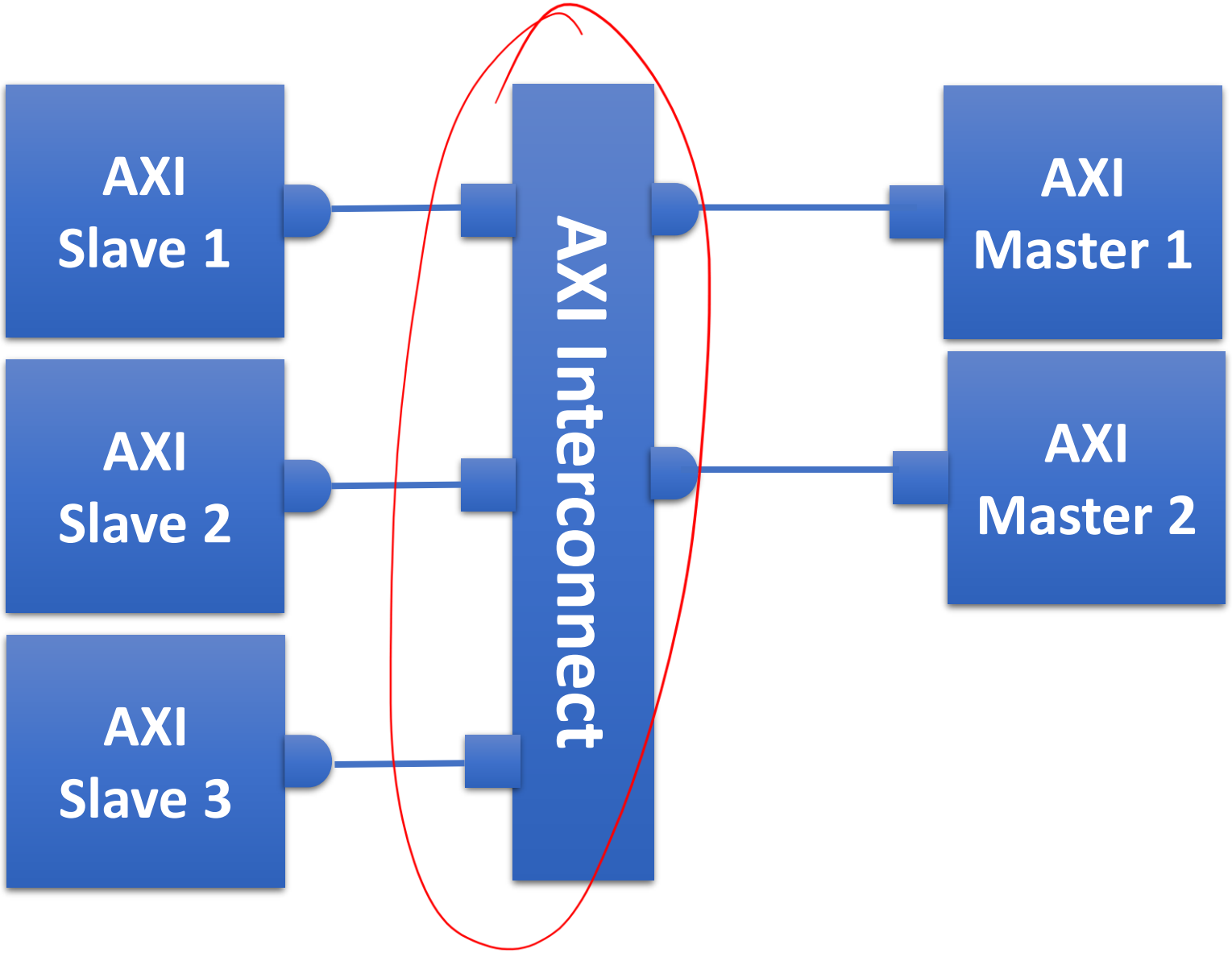
ECE 270: Embedded Logic Design

SDK
Software Development Kit
VIVADO™

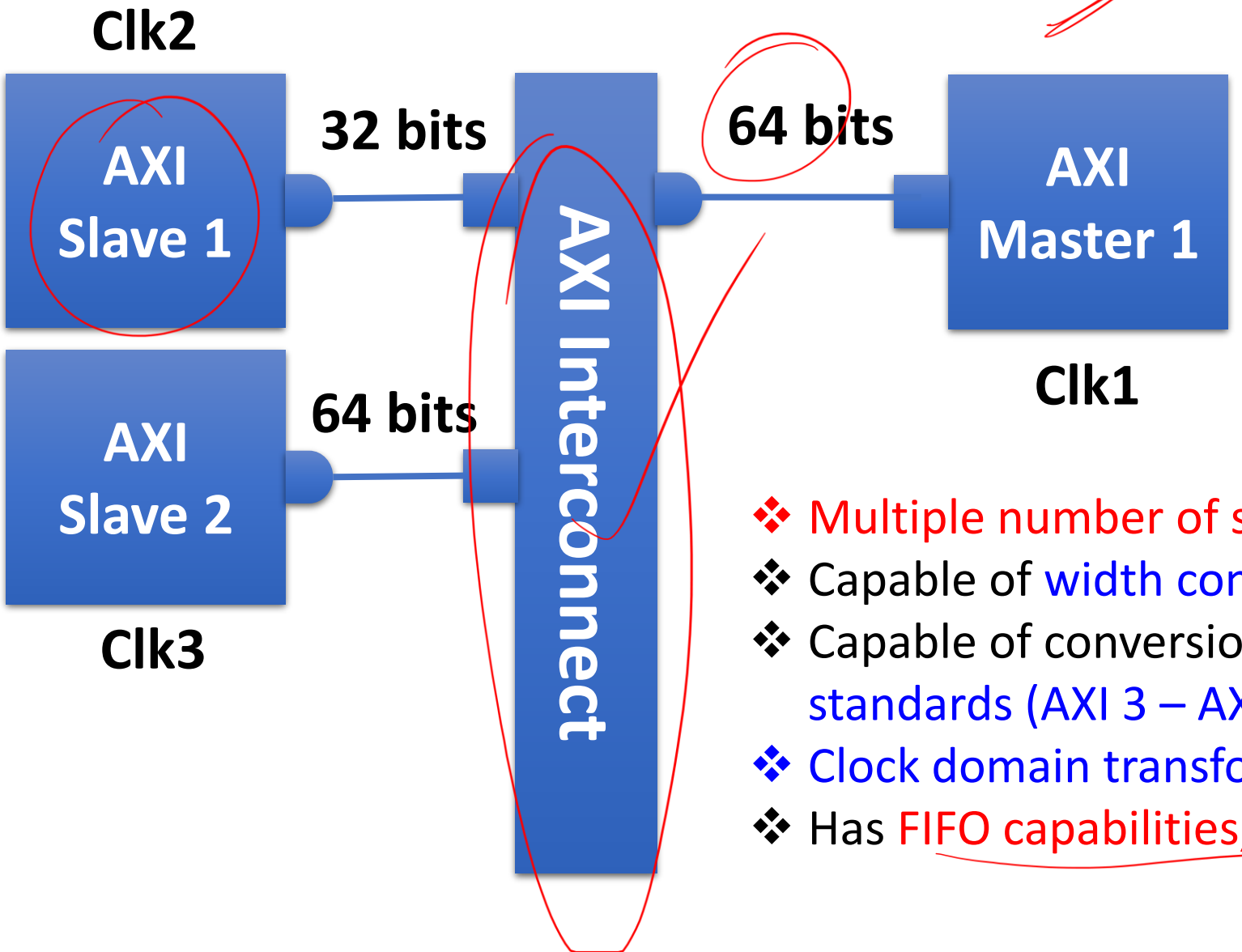




AXI Interconnect

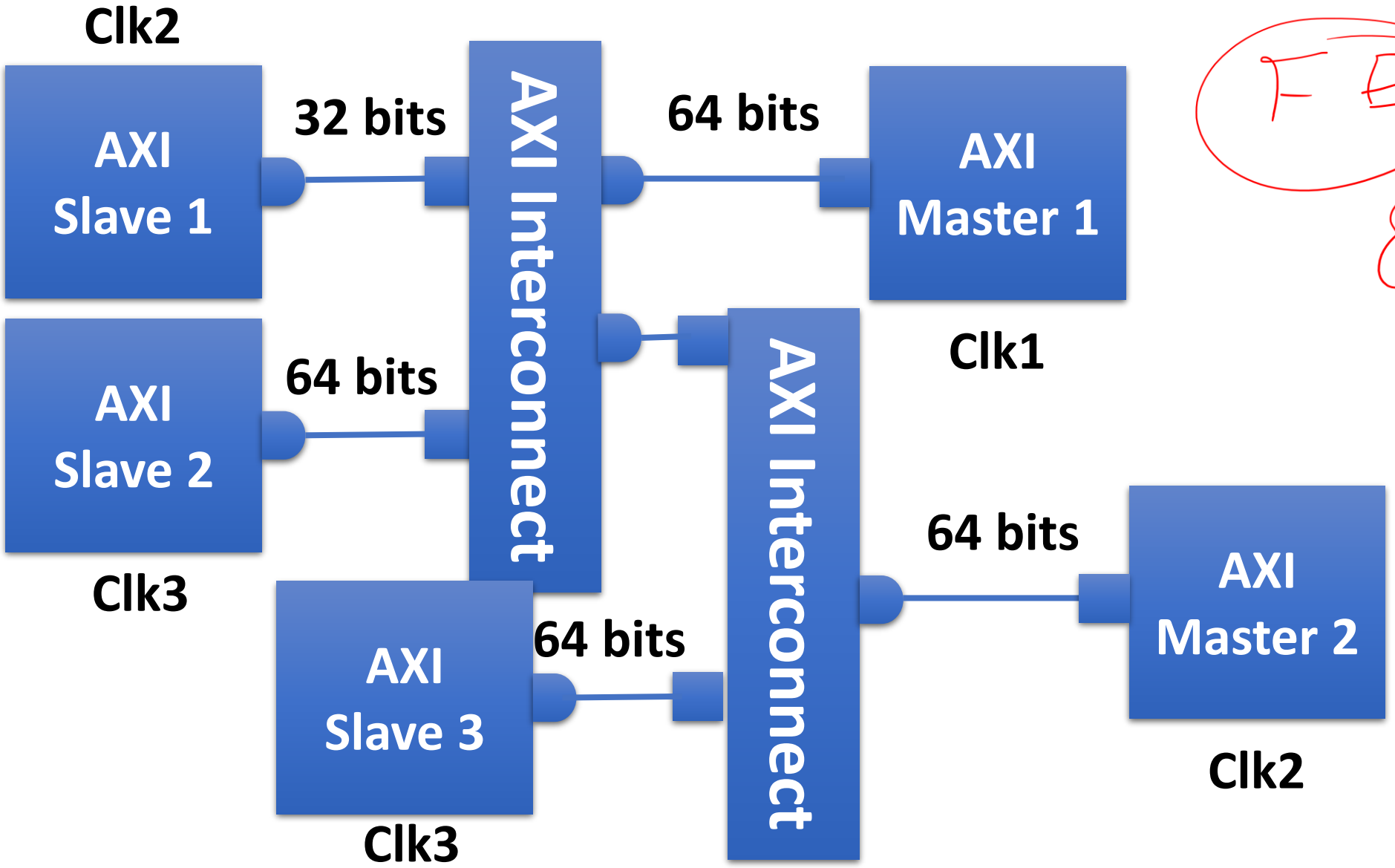


AXI Interconnect



- Multiple number of slave and master ports
- Capable of width conversion
- Capable of conversion between different AXI standards (AXI 3 – AXI 4)
- Clock domain transformations
- Has FIFO capabilities, registers for pipelining

Hierarchical AXI Interconnect



What is a system?

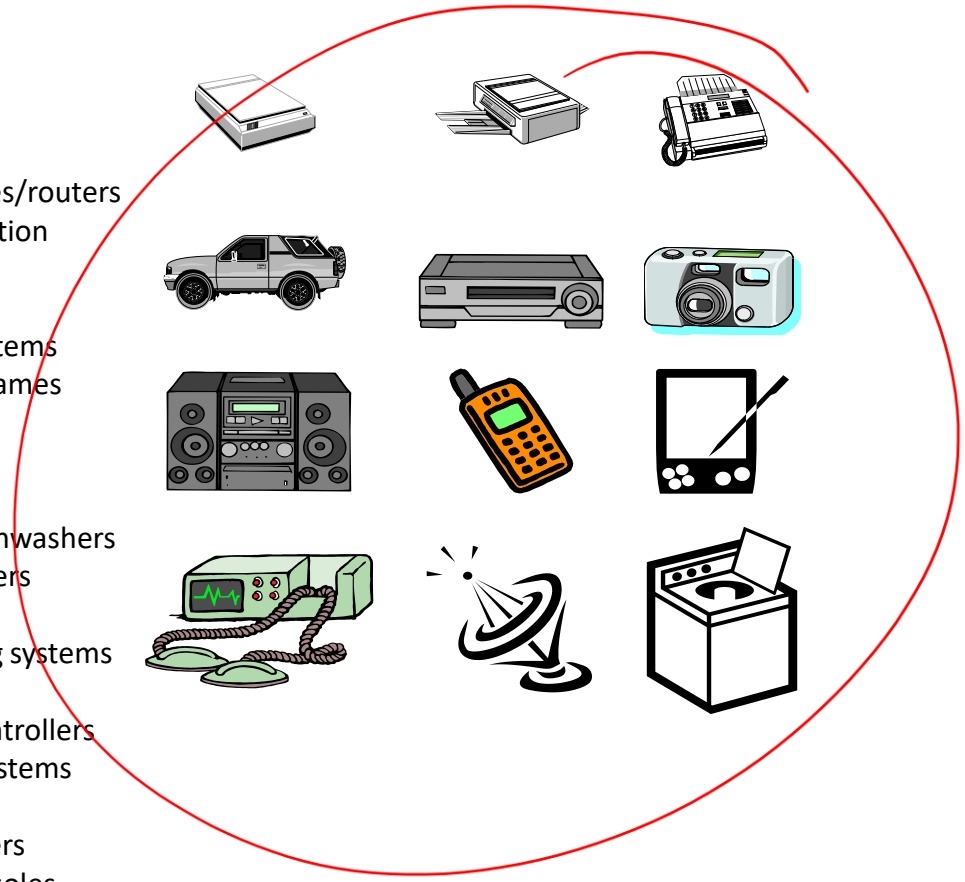
Embedded Logic Design

- A system is a way of working, organizing or doing one or many tasks according to a fixed plan, program or set of rules.
- It is an arrangement in which all its units assemble and work together according to the plan or program.
- **Embedded System** is a combination of hardware and software which together form a component of a larger machine.
- An embedded system is designed to run on its own without human intervention, and may be required to respond to events in real time.

A “short list” of embedded systems

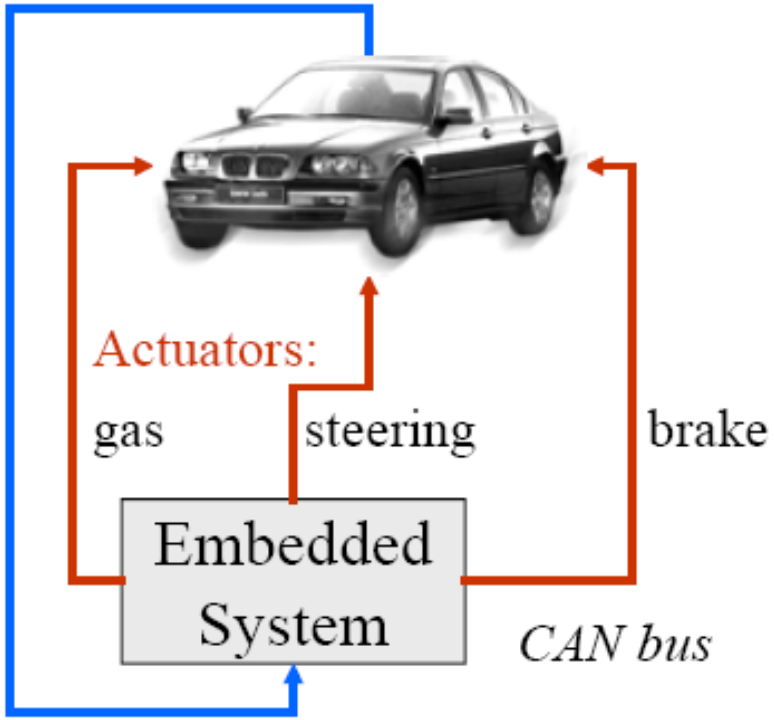
Anti-lock brakes
Auto-focus cameras
Automatic teller machines
Automatic toll systems
Automatic transmission
Avionic systems
Battery chargers
Camcorders
Cell phones
Cell-phone base stations
Cordless phones
Cruise control
Curbside check-in systems
Digital cameras
Disk drives
Electronic card readers
Electronic instruments
Electronic toys/games
Factory control
Fax machines
Fingerprint identifiers
Home security systems
Life-support systems
Medical testing systems

Modems
MPEG decoders
Network cards
Network switches/routers
On-board navigation
Pagers
Photocopiers
Point-of-sale systems
Portable video games
Printers
Satellite phones
Scanners
Smart ovens/dishwashers
Speech recognizers
Stereo systems
Teleconferencing systems
Televisions
Temperature controllers
Theft tracking systems
TV set-top boxes
VCR’s, DVD players
Video game consoles
Video phones
Washers and dryers

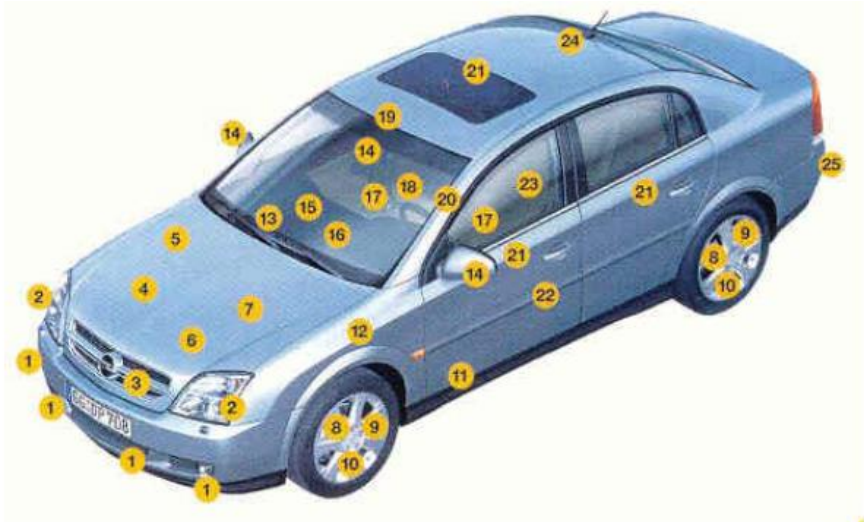


And the list goes on and on

Automobiles

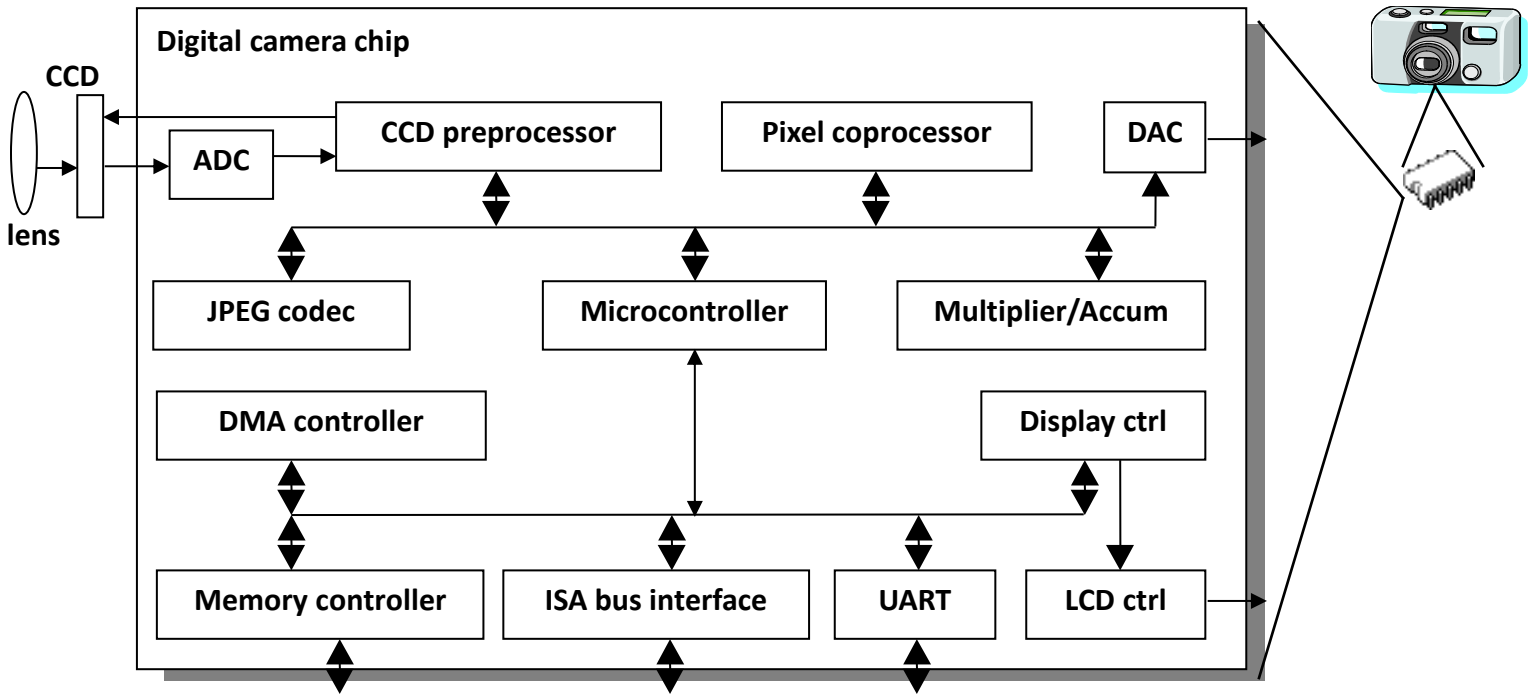


Sensors: Stereo-cameras, speedometer, accelerometers, signalling

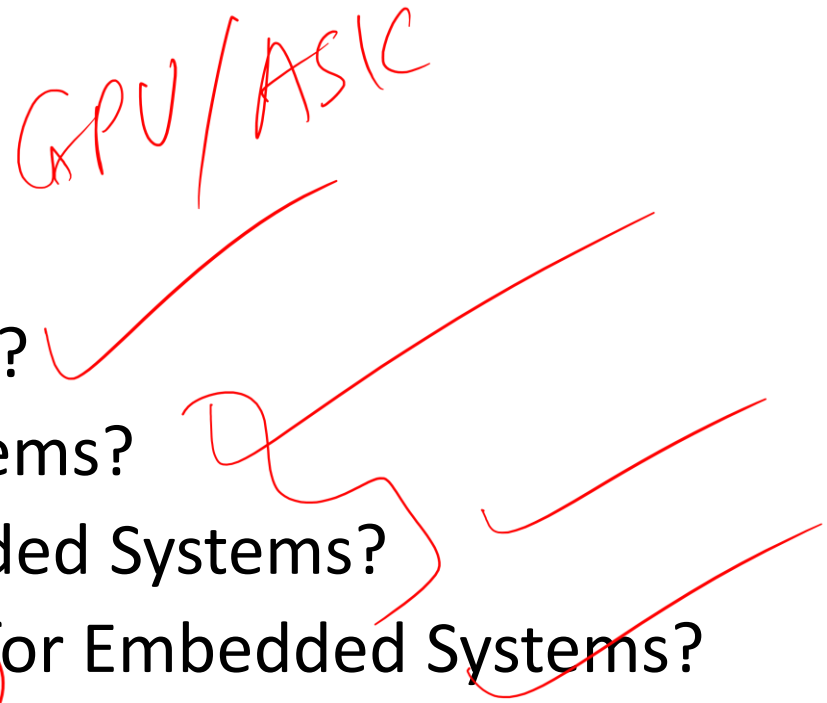


2002: Opel Vectra has over 40 sensors (25 types)

Digital Camera

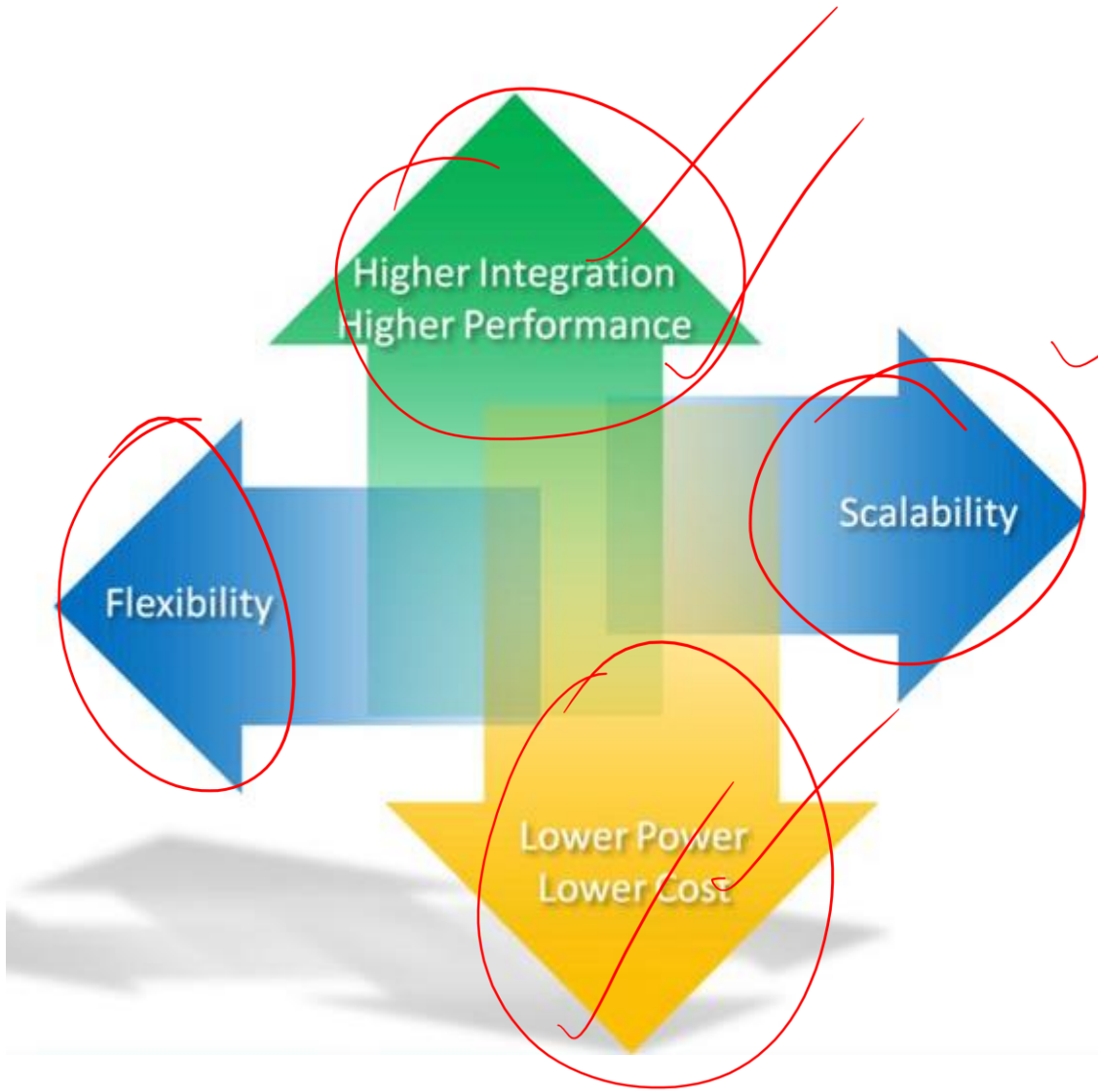


Embedded Logic Design



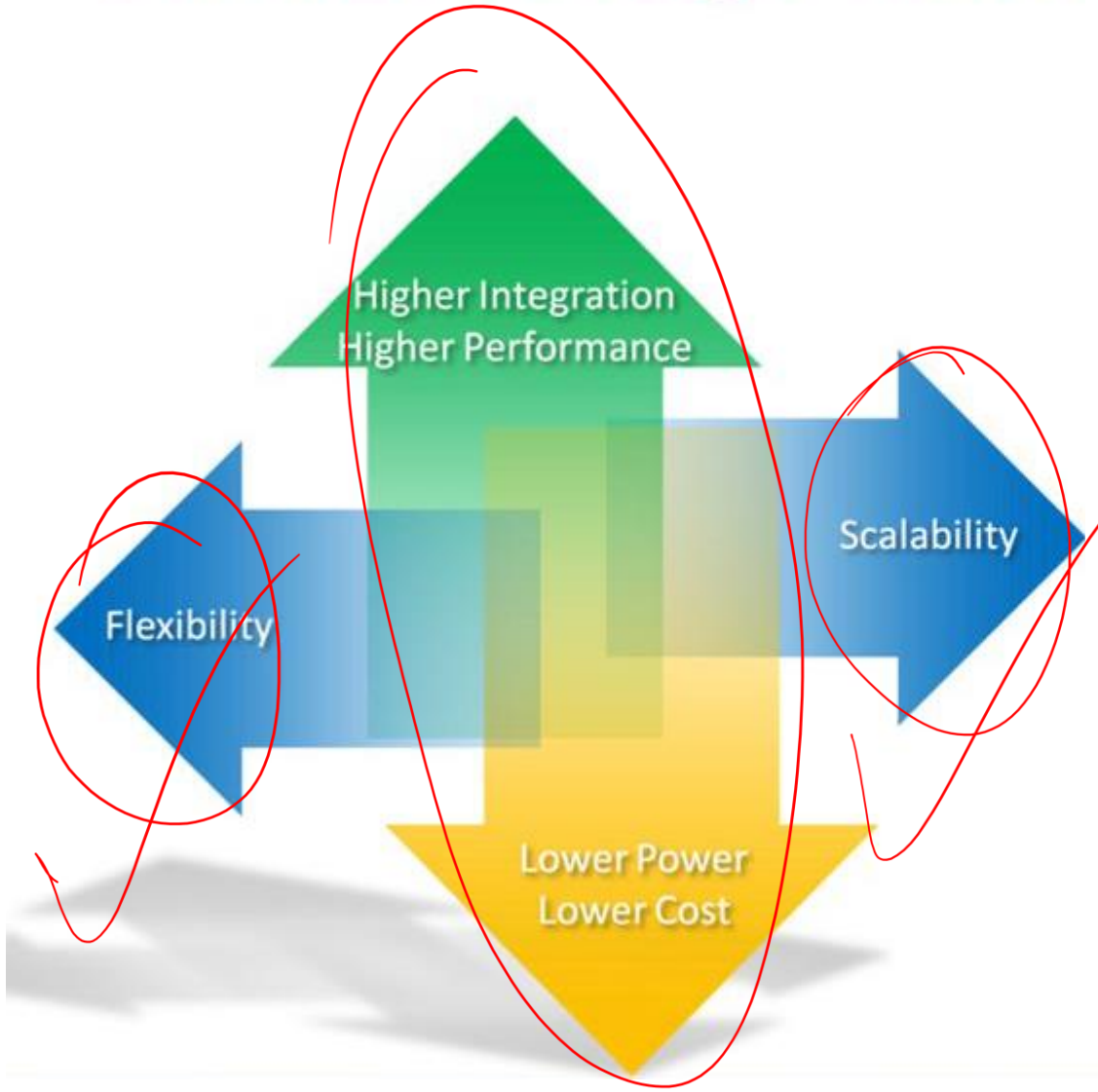
- What do you mean by word Embedded?
- Why FPGAs are part of Embedded Systems?
- What are other components of Embedded Systems?
- How to build FPGA based accelerators for Embedded Systems?

Demands of Today's Technology



- Over the years, industries are facing the same challenges and pressure: *Next generation system must improve the **performance** and should offer higher level of **integration**.*
- Furthermore, the *cost and power should be reduced*.
- These challenges have been addressed till now efficiently.

Demands of Today's Technology



- However, there is additional requirements of *flexibility* and *scalability* in the upcoming applications in order to tune your design to meet customer requirements efficiently.
- Need of single platform that can be scaled from low-end to high-end

ASIC Vs ASSP

	ASIC	ASSP
Performance	+	+
Power	+	+
Unit Cost	+	+
TCO	■	+

- TCO: Total cost of ownership
- Takes into account all the cost associated with development with such solution
- TCO is high for ASIC and depends on the how many units to be produced

+ positive, - negative, ■ neutral

ASIC Vs ASSP

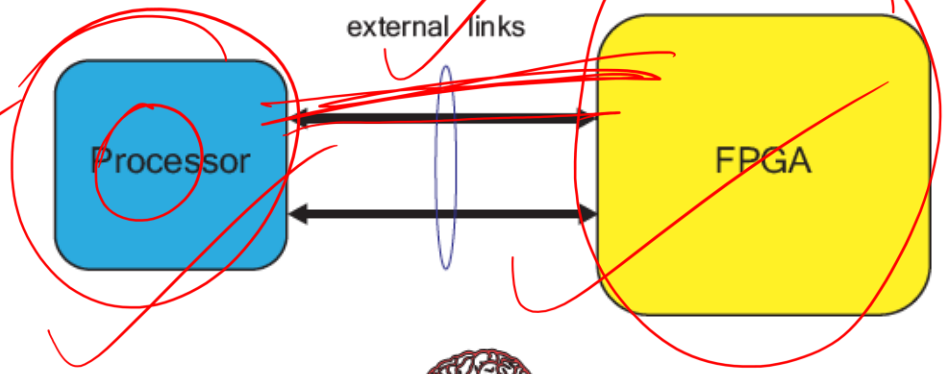
	ASIC	ASSP
Performance	+	+
Power	+	+
Unit Cost	+	+
TCO	■	+
Risk	-	+
TTM	-	+
Flexibility	-	-
Scalability	-	■

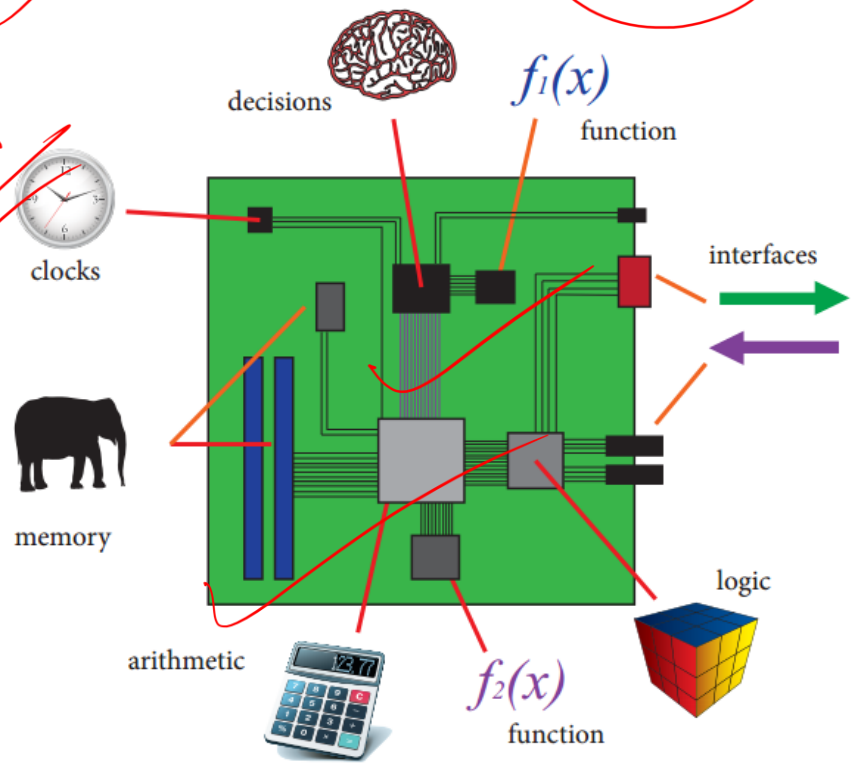
+ positive, - negative, ■ neutral

ASIC Vs ASSP Vs 2 Chip Solution

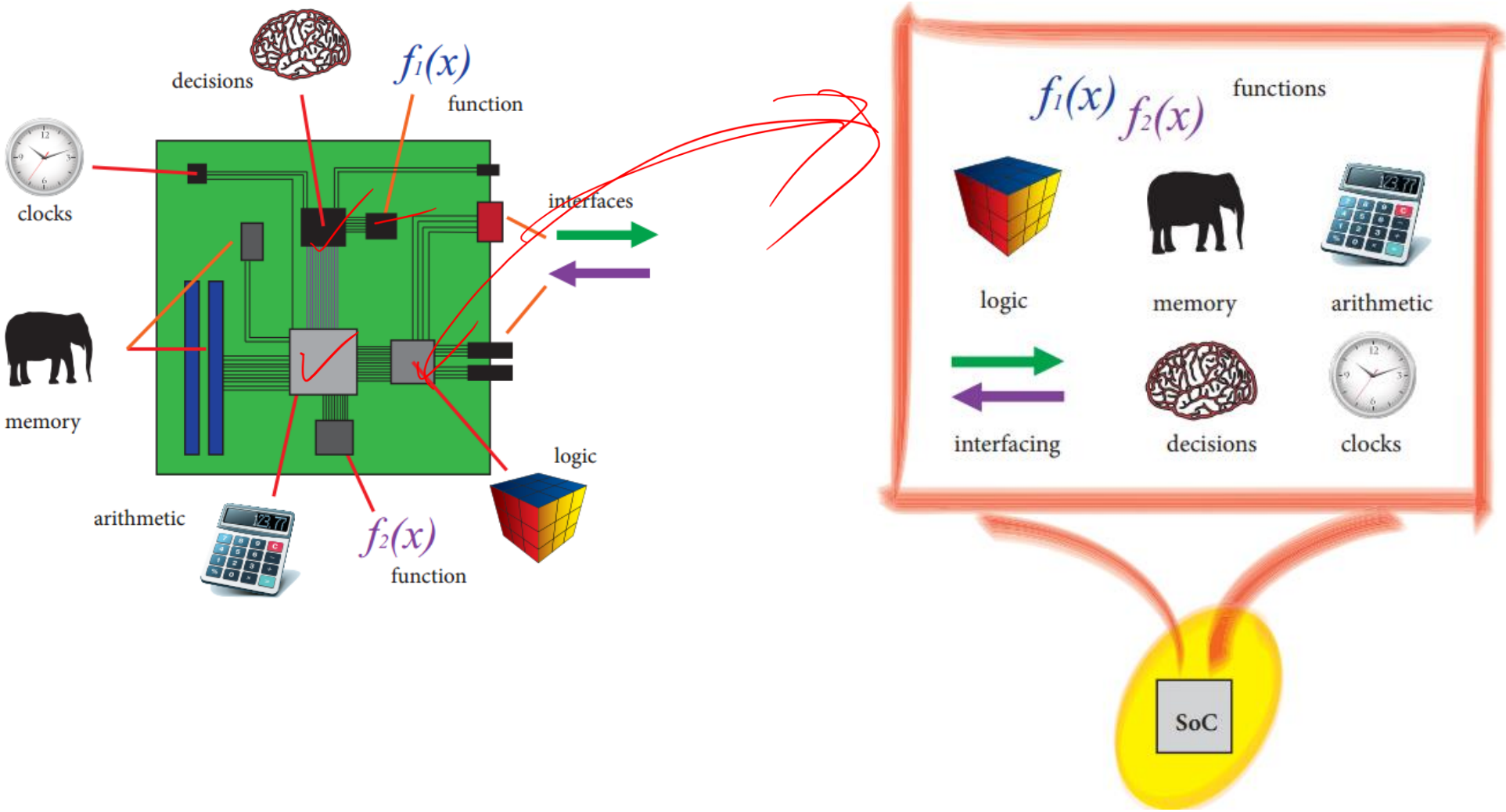
	ASIC	ASSP	2 Chip Solution
Performance	+	+	■
Power	+	+	-
Unit Cost	+	+	-
TCO	■	+	+
Risk	-	+	+
TTM	-	+	+
Flexibility	-	-	+
Scalability	-	■	+

+ positive, - negative, ■ neutral



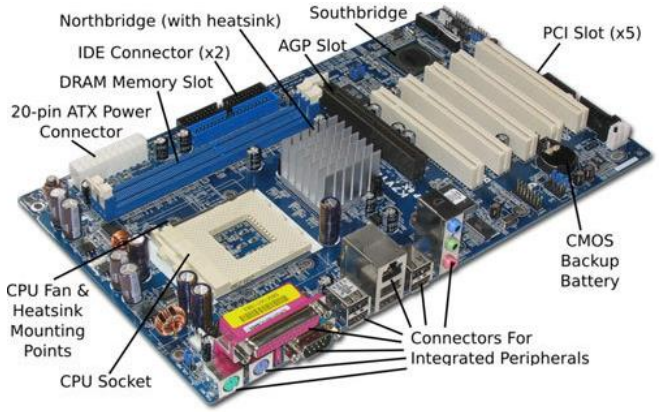


System on-a-board and System-on-chip

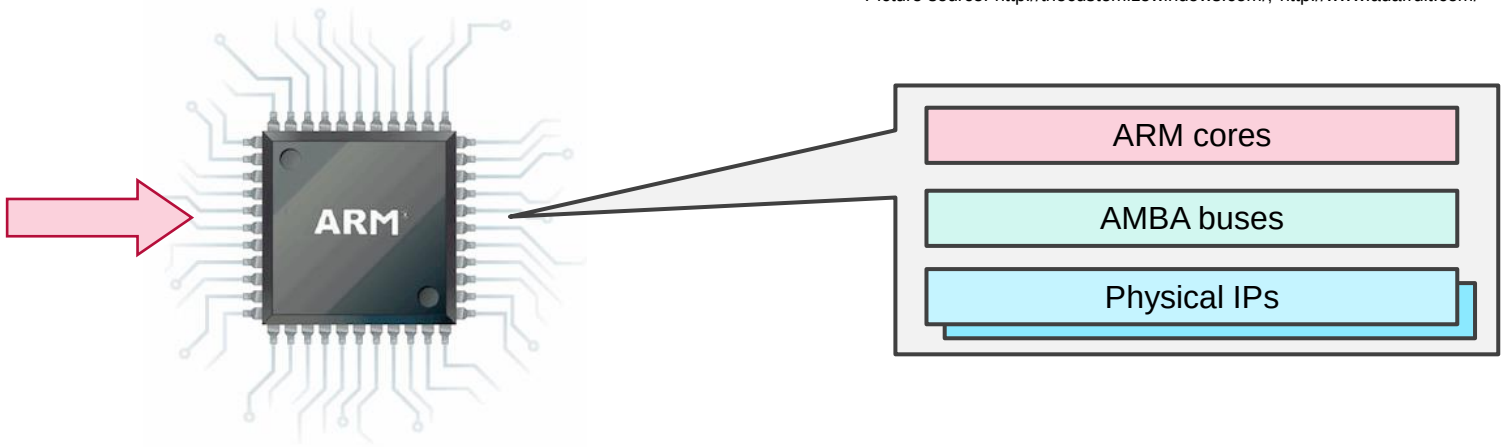


What is a System-on-Chip

Picture source: http://thecustomizewindows.com/, http://www.adafruit.com/



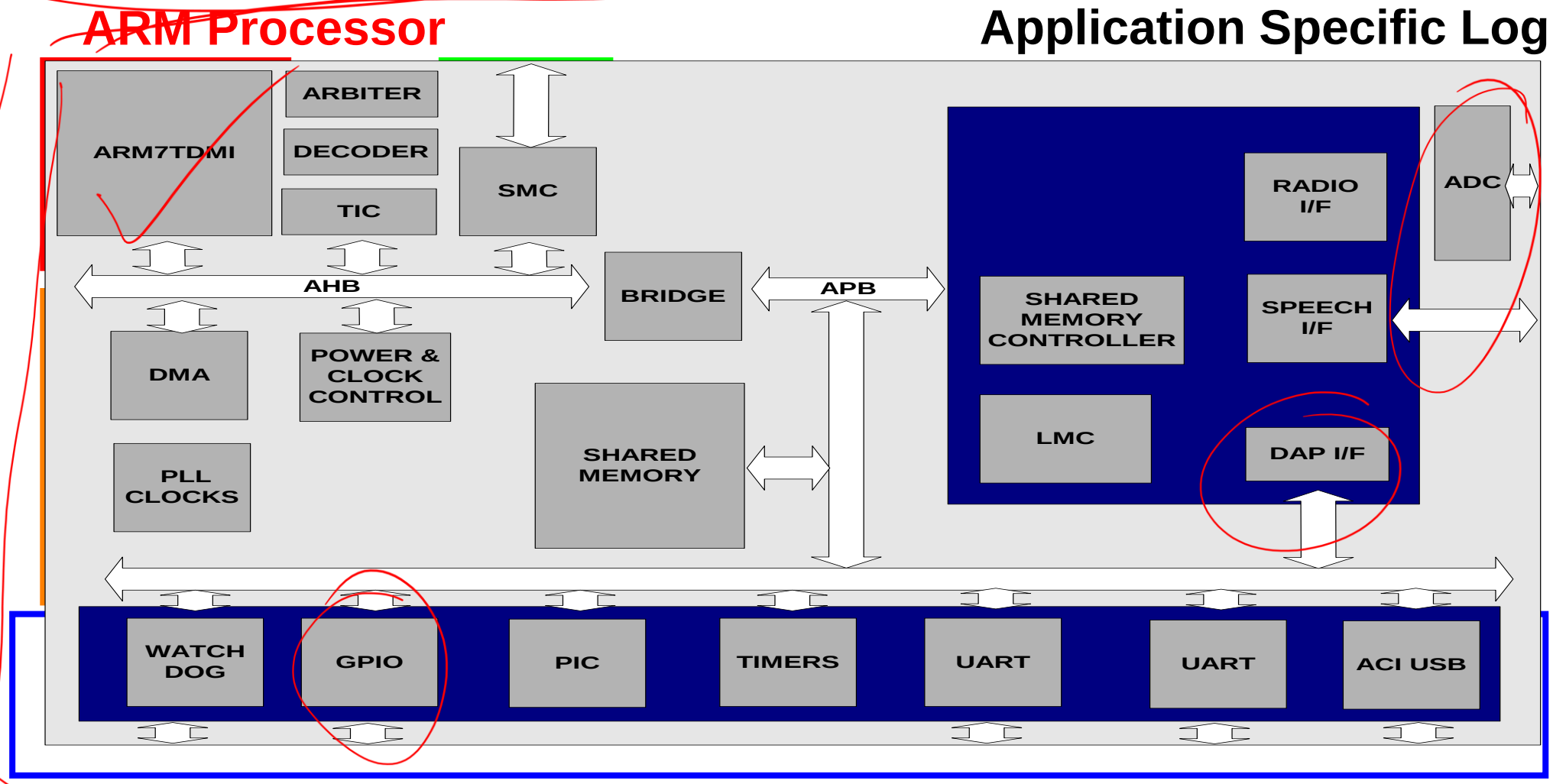
Mother board of a PC



System on a Chip

SoC is a single silicon chip that can be used to implement the functionality of an entire system, rather than using several different physical chips on a board.

Bluetooth SoC



Picture source: http://www.ecs.soton.ac.uk/, SoC Advance design Technique

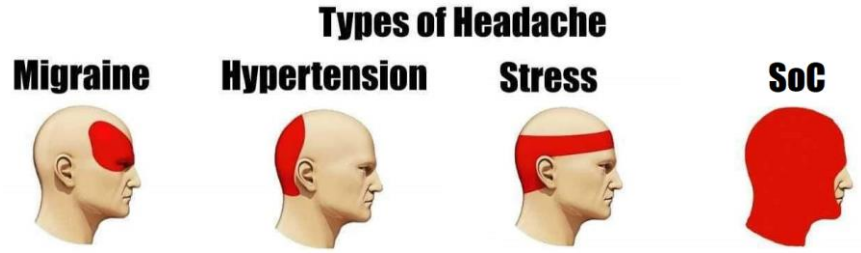
Advantages of SoC

- **Higher performance benefiting from:**
  - Less propagation delay since internal wires are shorter.
  - Less gate delay as internal transistors have lower electrical impedance.
- **Power efficiency benefiting from:**
  - Lower voltage required (typically < 2.0 volts) compared with external chip voltage (typically >3.0 volts).
  - Less capacitance.
- **Lighter footprint:**
  - Device size and weight is reduced.
- **Higher reliability:**
  - All encapsulated in a single chip package, less interference from the external world.
- **Low cost:**
  - The cost per unit is reduced since a single chip design can be fabricated in a large volumes.

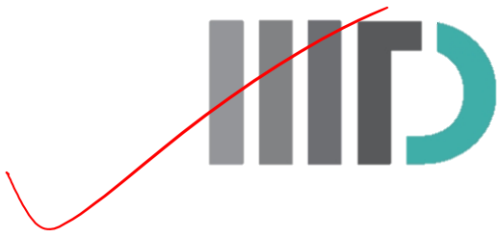
Credits: ARM Univ. Prog.

Limitations of ASIC Based SoC

- **Less flexibility**
  - Unlike a PC or a laptop, which allows you to upgrade a single component, such as RAM or graphic card, a SoC cannot be easily upgraded after manufacture. Though external components can be added, it is no longer SoC
- **Application Specific**
  - Most SoCs are created for particular applications thus they are not easily adapted to other applications.
- **Complexity**
  - A SoC design usually requires advanced skills compared with board-level development.



Credits: ARM Univ. Prog.

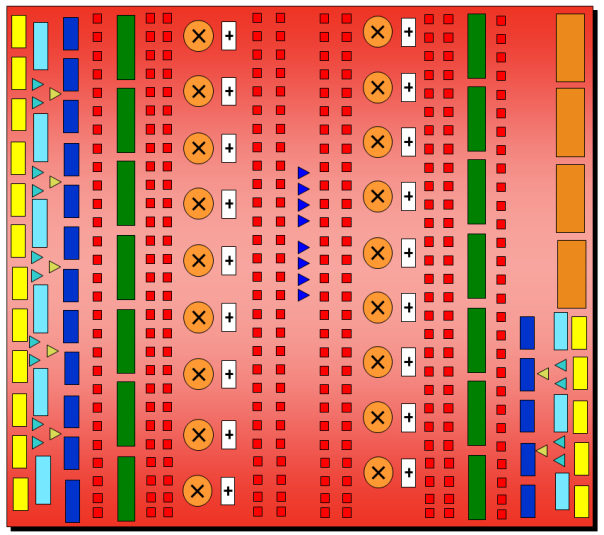


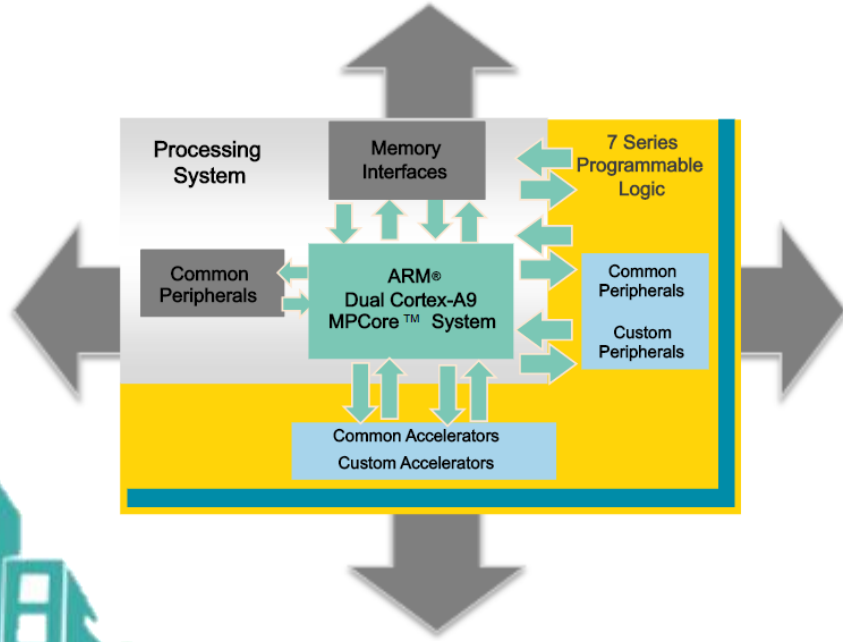
ECE | DEPARTMENT OF ELECTRONICS & COMMUNICATIONS ENGINEERING

A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design

SDK
Software Development Kit
VIVADO™

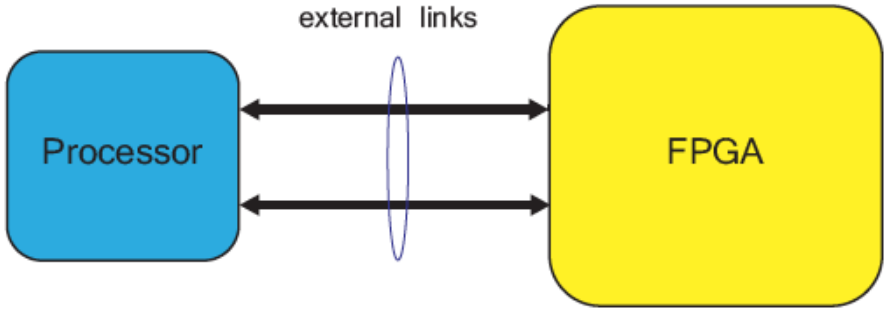


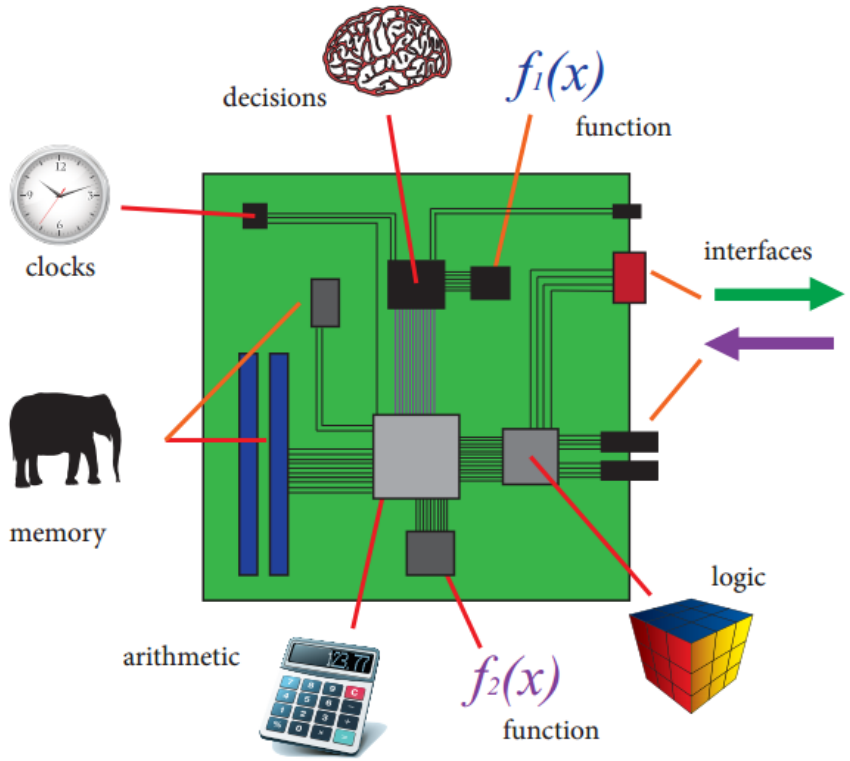


ASIC Vs ASSP Vs 2 Chip Solution

	ASIC	ASSP	2 Chip Solution
Performance	+	+	■
Power	+	+	-
Unit Cost	+	+	-
TCO	■	+	+
Risk	-	+	+
TTM	-	+	+
Flexibility	-	-	+
Scalability	-	■	+

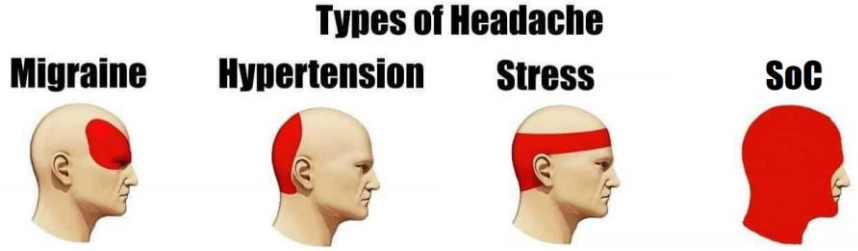
+ positive, - negative, ■ neutral





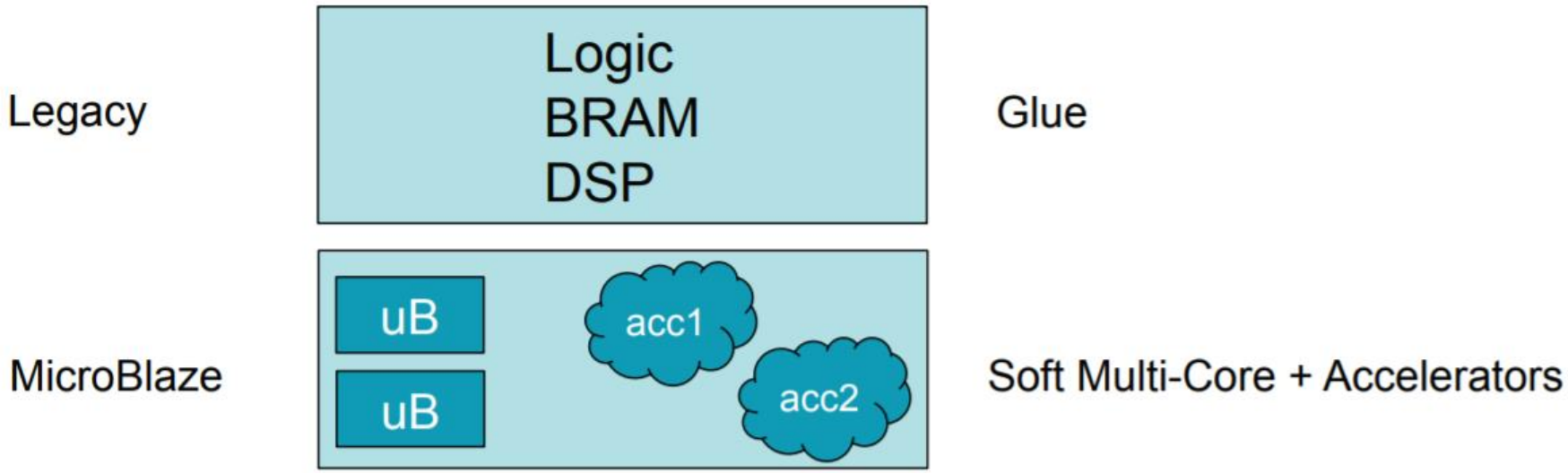
Limitations of ASIC Based SoC

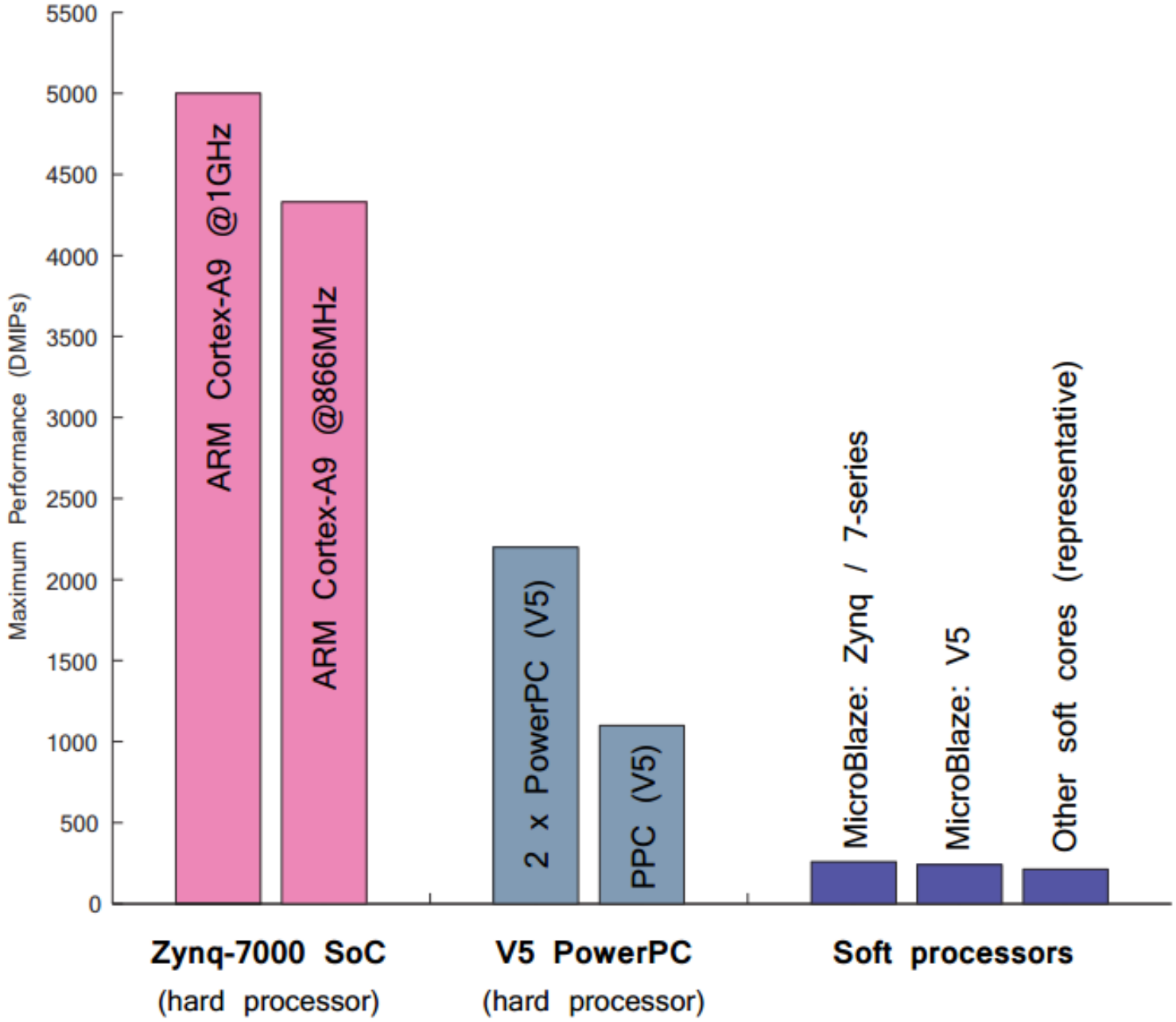
- **Less flexibility**
  - Unlike a PC or a laptop, which allows you to upgrade a single component, such as RAM or graphic card, a SoC cannot be easily upgraded after manufacture. Though external components can be added, it is no longer SoC
- **Application Specific**
  - Most SoCs are created for particular applications thus they are not easily adapted to other applications.
- **Complexity**
  - A SoC design usually requires advanced skills compared with board-level development.



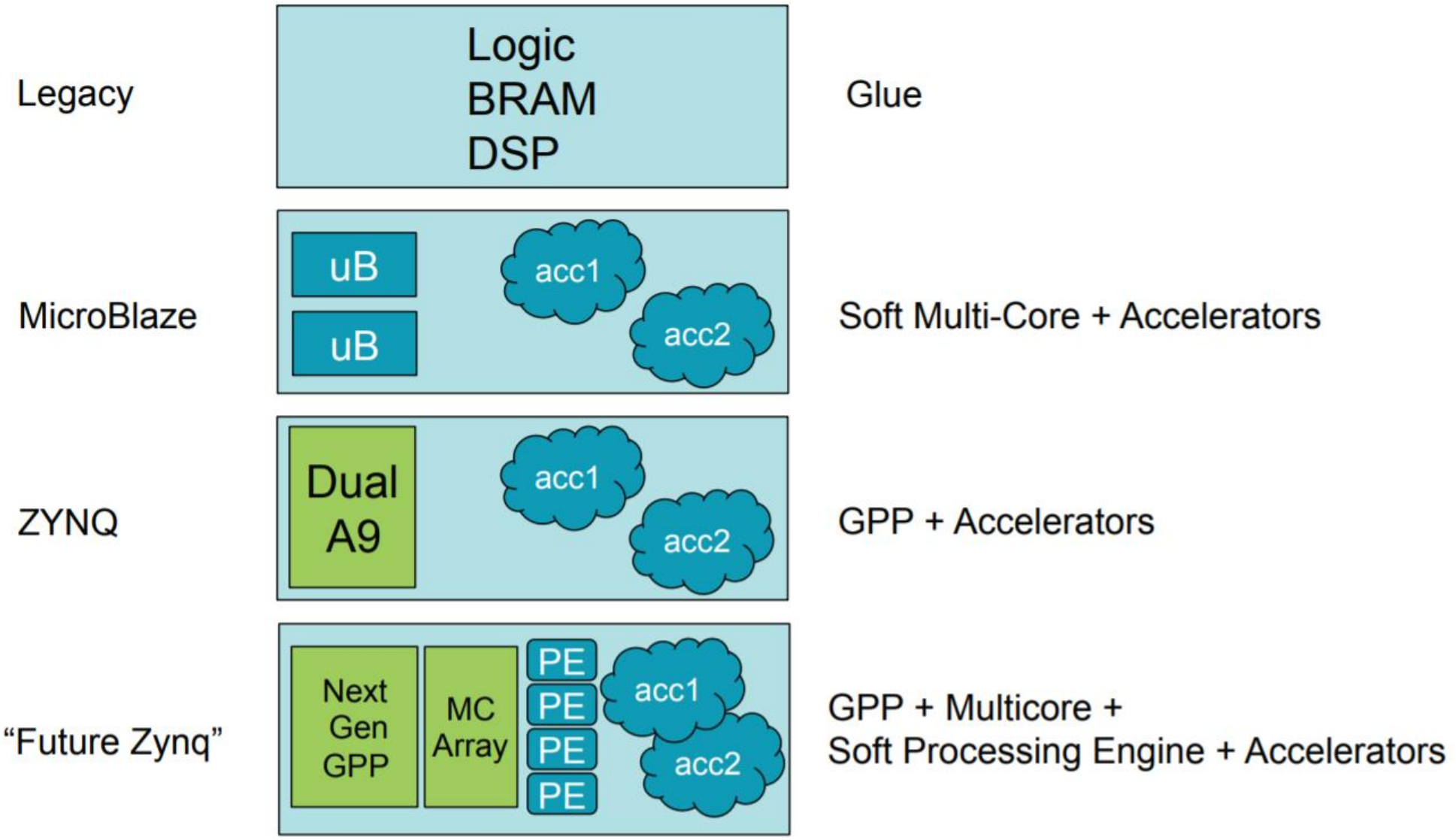
Credits: ARM Univ. Prog.

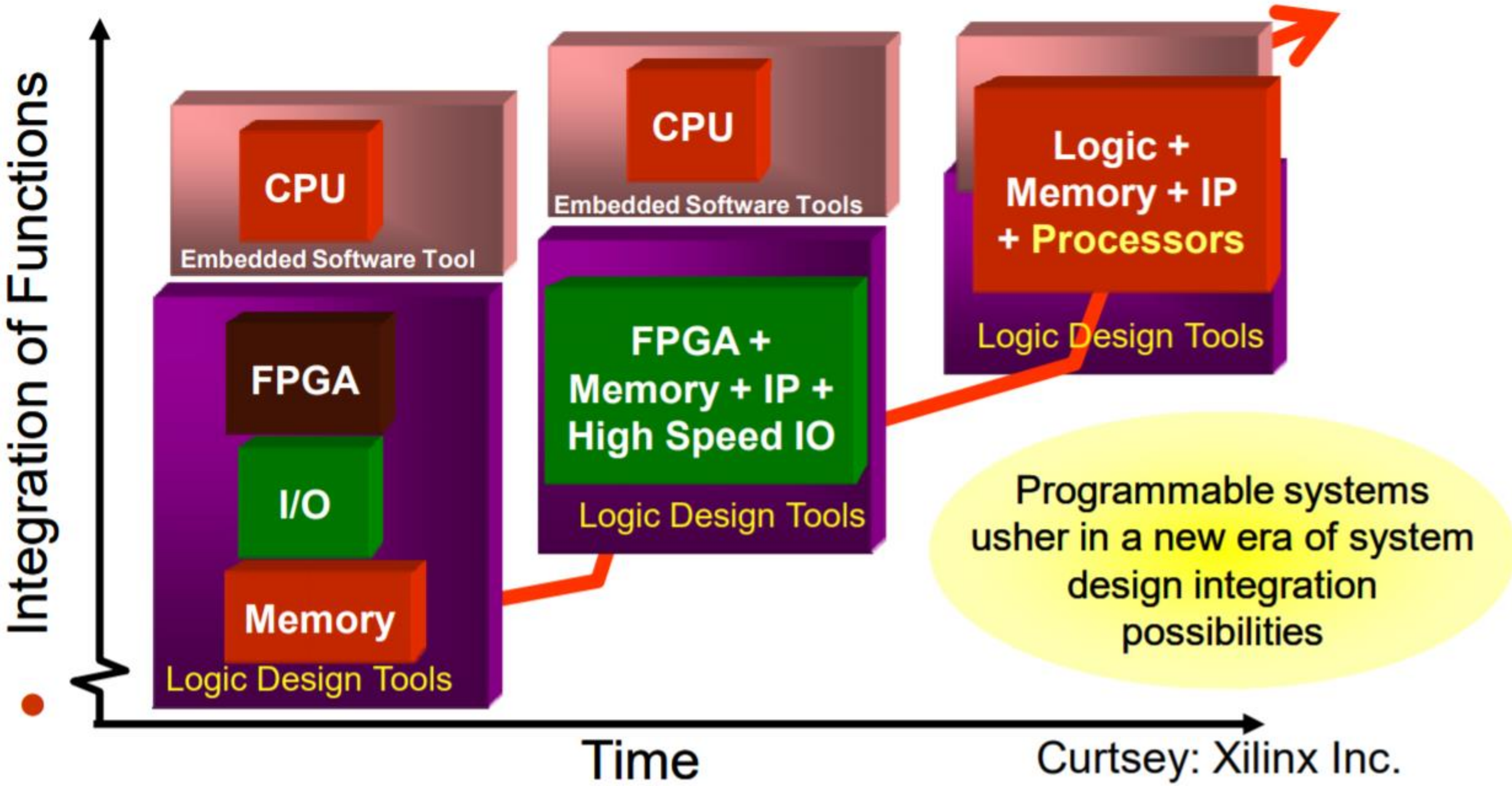
Zynq SoC


Legacy
Logic
BRAM
DSP
Glue
MicroBlaze
uB
uB
acc1
acc2
Soft Multi-Core + Accelerators

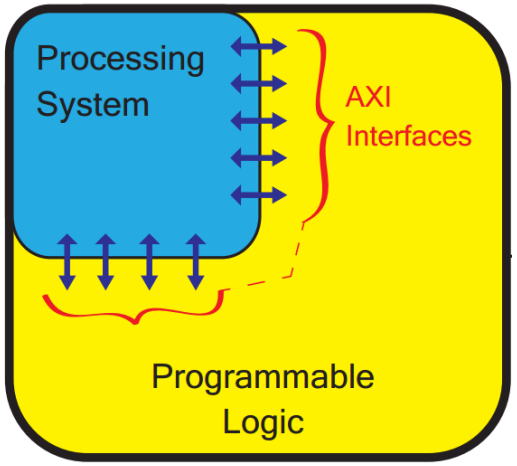

Maximum Performance (DMIPs)
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
ARM Cortex-A9 @1GHz
ARM Cortex-A9 @866MHz
2 x PowerPC (V5)
PPC (V5)
MicroBlaze: Zynq / 7-series
MicroBlaze: V5
Other soft cores (representative)
Zynq-7000 SoC
(hard processor)
V5 PowerPC
(hard processor)
Soft processors

Zynq SoC


Legacy
Logic
BRAM
DSP
Glue
MicroBlaze
uB
uB
acc1
acc2
Soft Multi-Core + Accelerators
ZYNQ
Dual
A9
acc1
acc2
GPP + Accelerators
"Future Zynq"
Next
Gen
GPP
MC
Array
PE
PE
PE
PE
acc1
acc2
GPP + Multicore +
Soft Processing Engine + Accelerators


Integration of Functions
CPU
Embedded Software Tool
FPGA
I/O
Memory
Logic Design Tools
CPU
Embedded Software Tools
FPGA +
Memory + IP +
High Speed IO
Logic Design Tools
Logic +
Memory + IP
+ Processors
Logic Design Tools
Programmable systems
usher in a new era of system
design integration
possibilities
Time
Curtsey: Xilinx Inc.

Zynq SoC

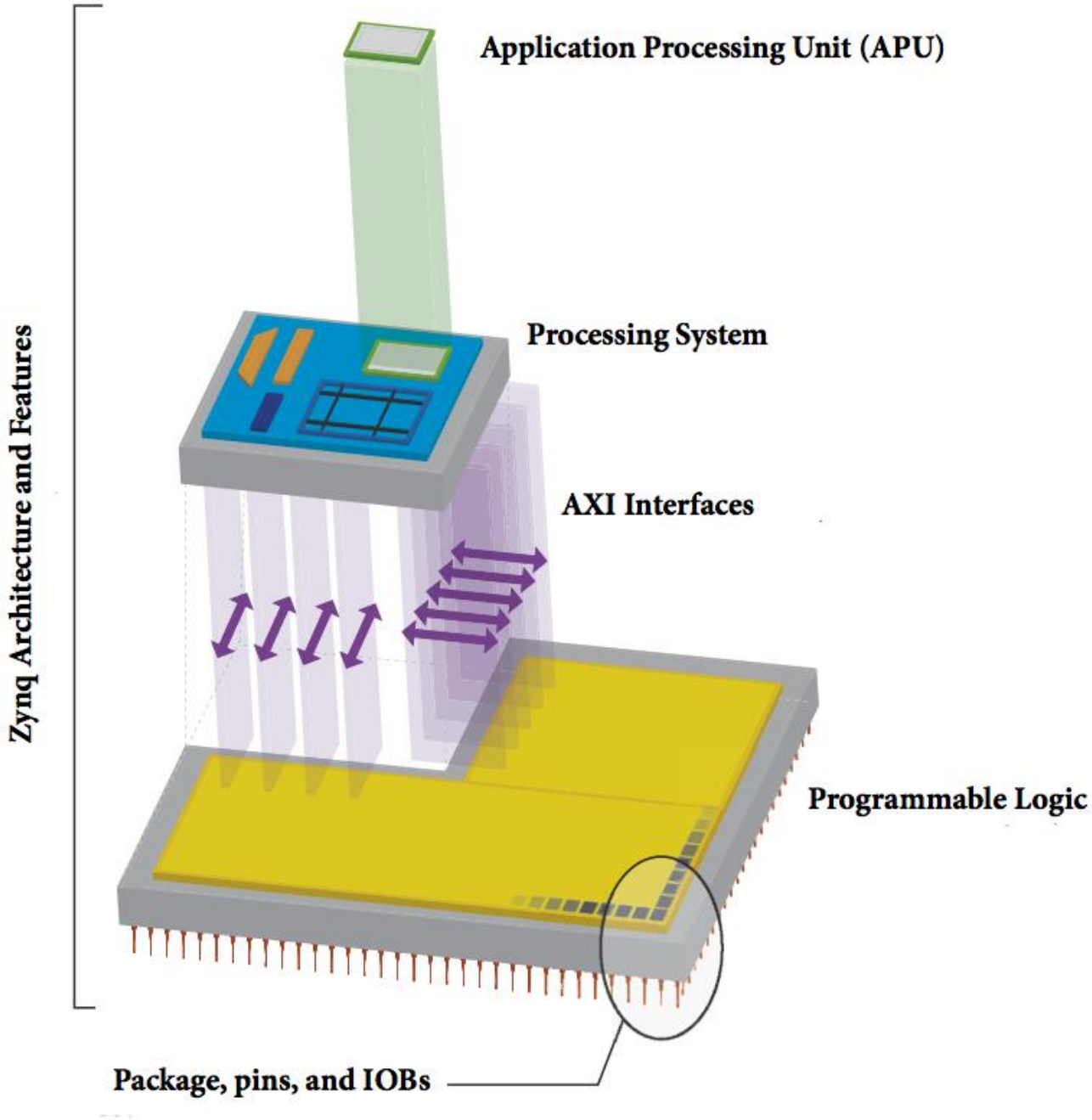


	ASIC	ASSP	2 Chip Solution	Zynq-7000
Performance	+	+	■	+
Power	+	+	-	+
Unit Cost	+	+	-	■
TCO	■	+	+	+
Risk	-	+	+	+
TTM	-	+	+	+
Flexibility	-	-	+	+
Scalability	-	■	+	+

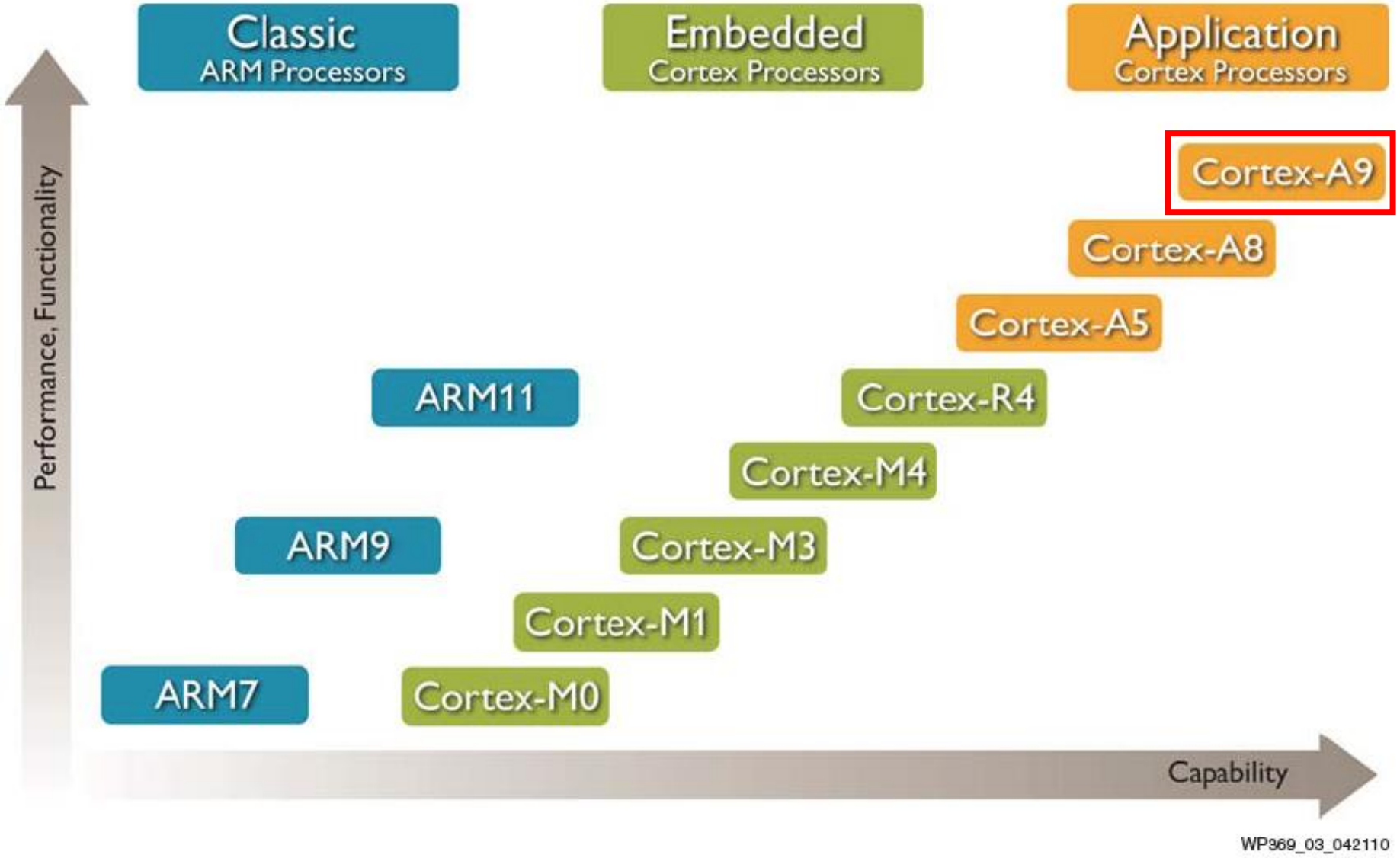
+ positive, - negative, ■ neutral

Zynq

- Not an ordinary FPGA
- Not an ordinary microprocessor
- Unique blend of two technologies with powerful interconnect
- In Zynq, the ARM Cortex-A9 is an application grade processor, capable of running full operating systems such as Linux, while the programmable logic is based on Xilinx 7-series FPGA architecture



ARM Processor Roadmap



ARM

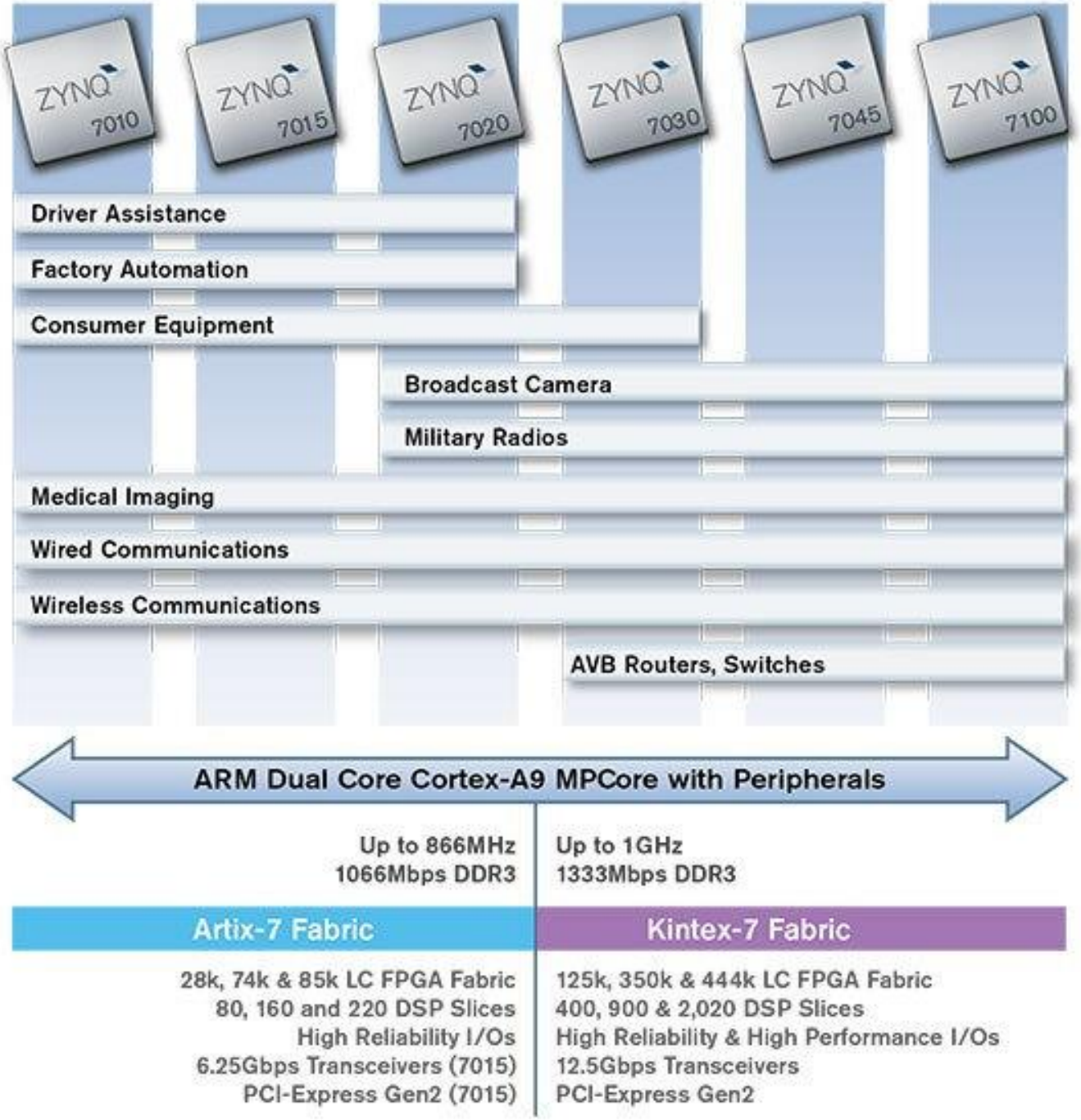
Apple A5:

iPhone 4S iPad 2

MOTOROLA XOOM™

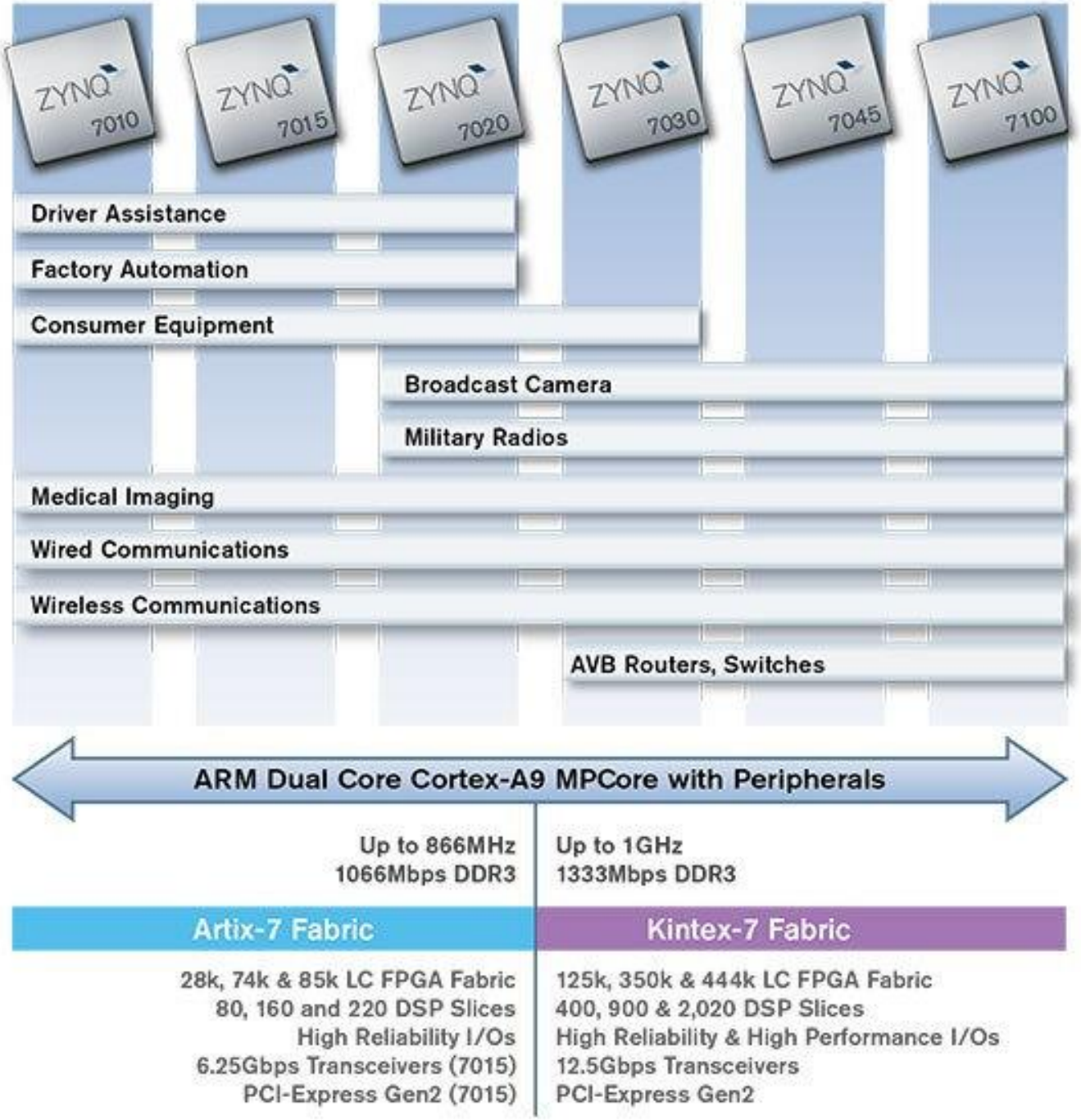
Zynq

- All programmable SoC (APSoC)
- Offers software and hardware and IO Programmability on single SoC
- Is there any link between Zynq and Zinc?

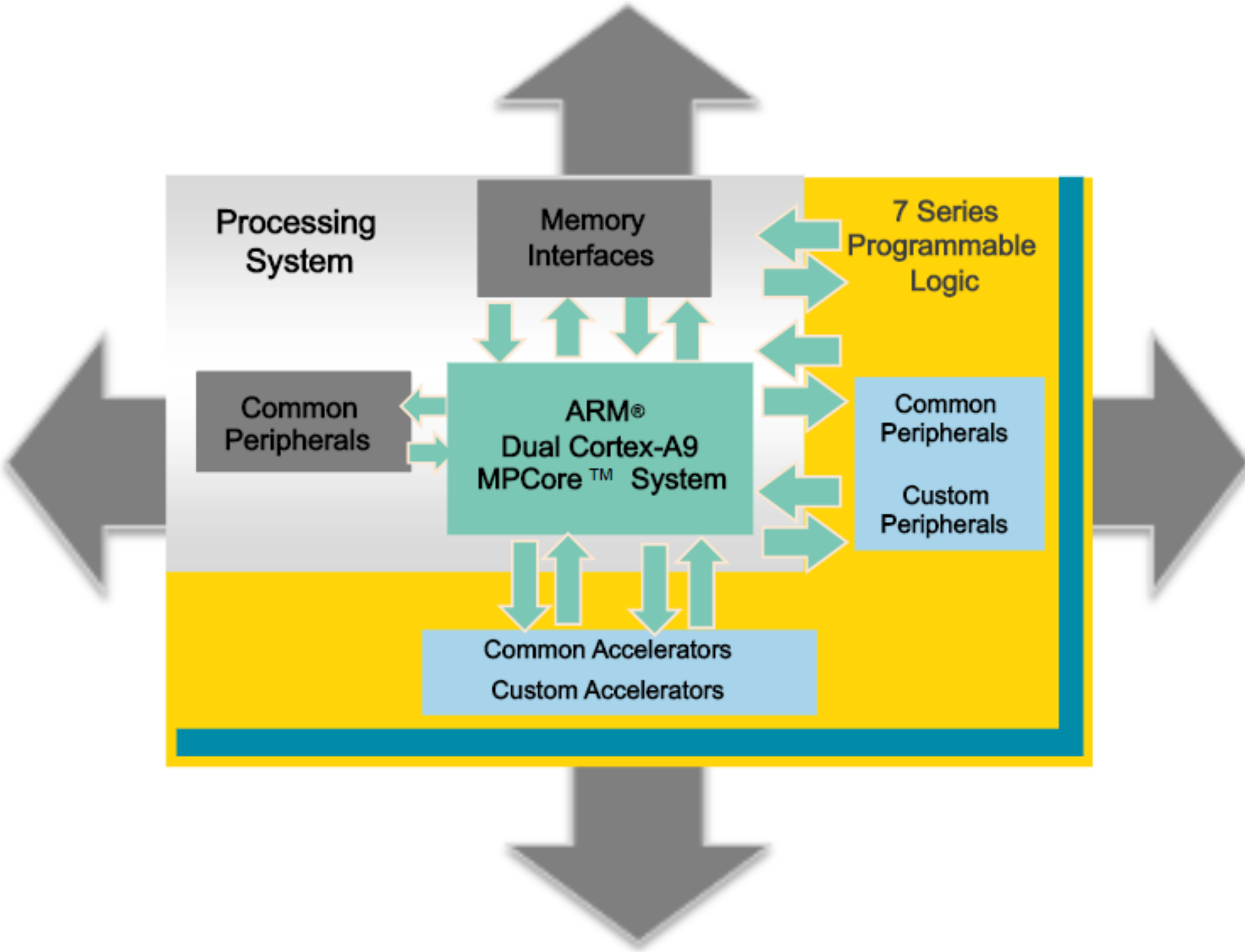


Zynq

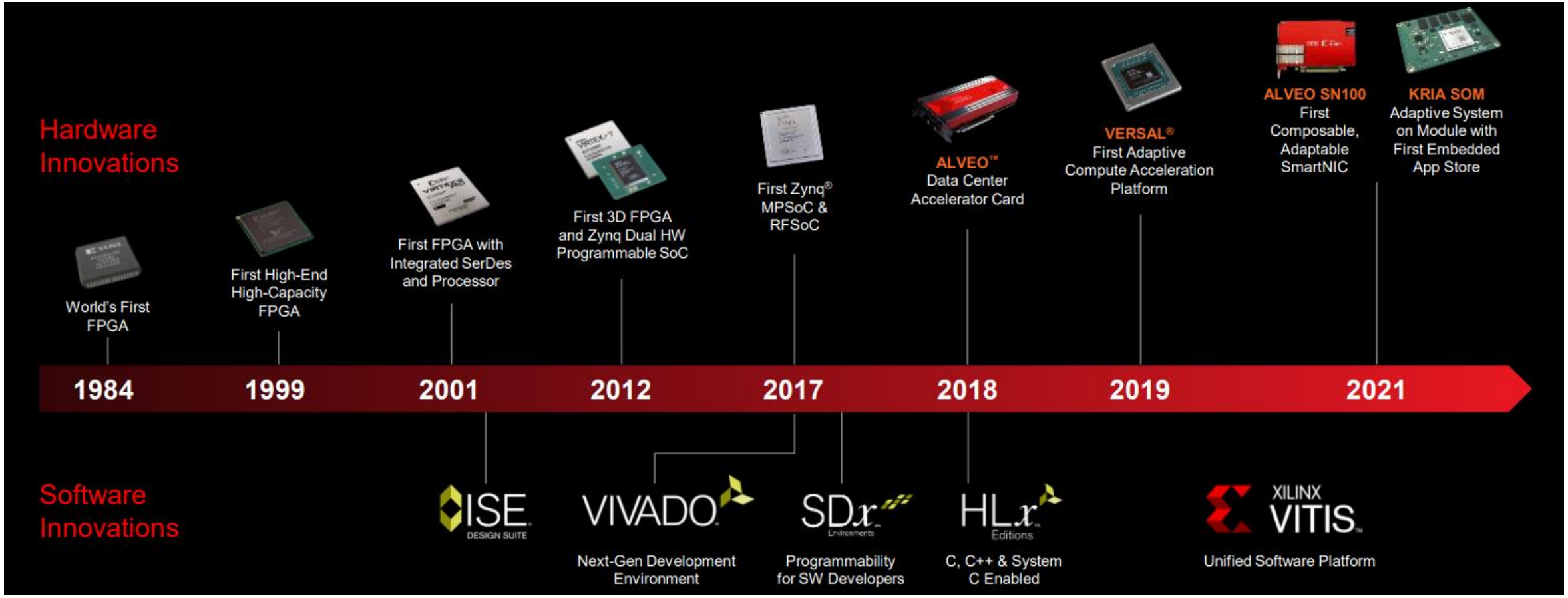
- All programmable SoC (APSoC)
- Offers software and hardware and IO Programmability on single SoC
- Is there any link between Zynq and Zinc?
- Zynq devices are intended to be flexible and form a compelling platform for a wide variety of applications, just as the metal zinc can be mixed with various other metals to form alloys with differing desirable properties.



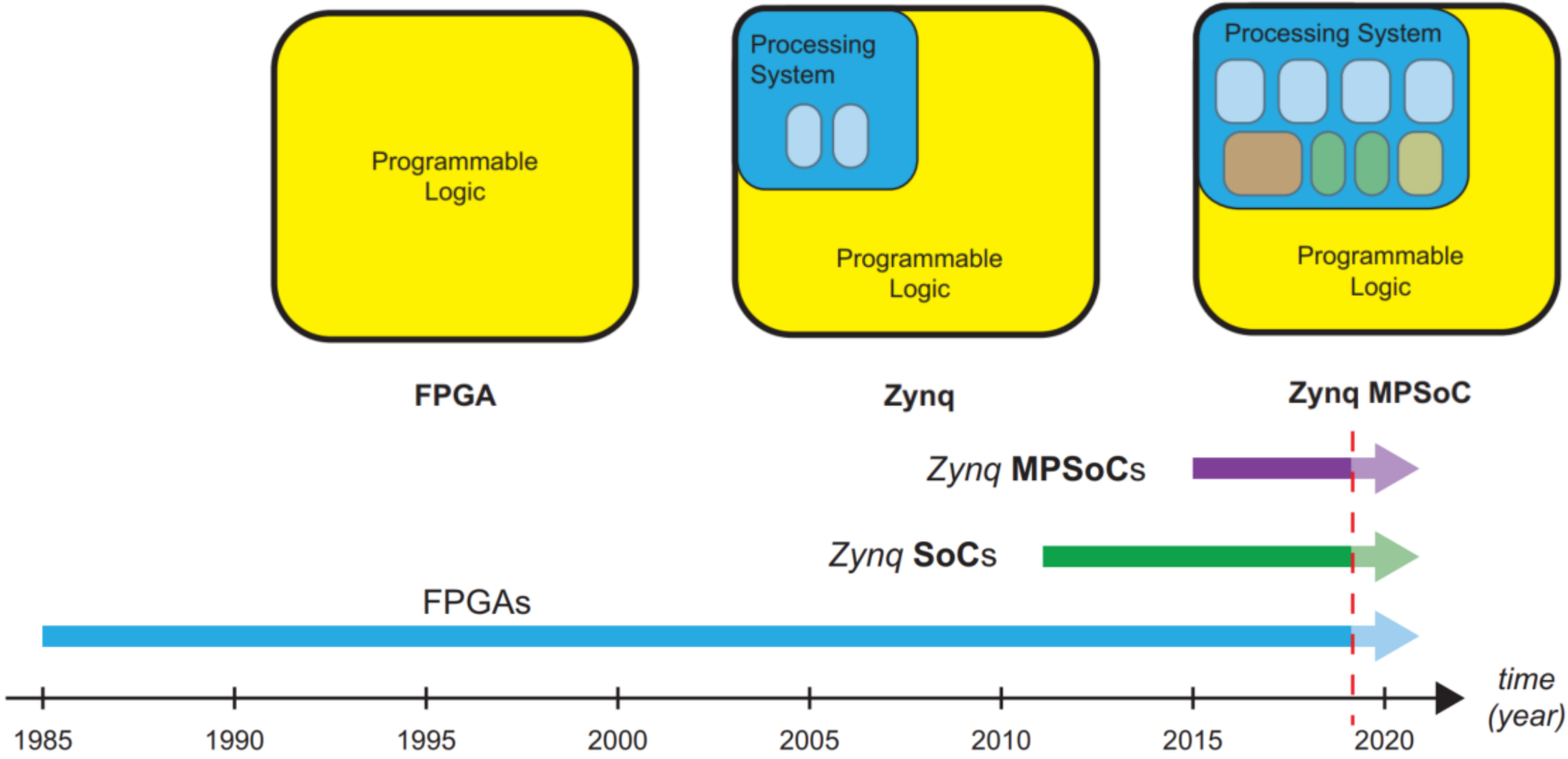
Zynq SoC

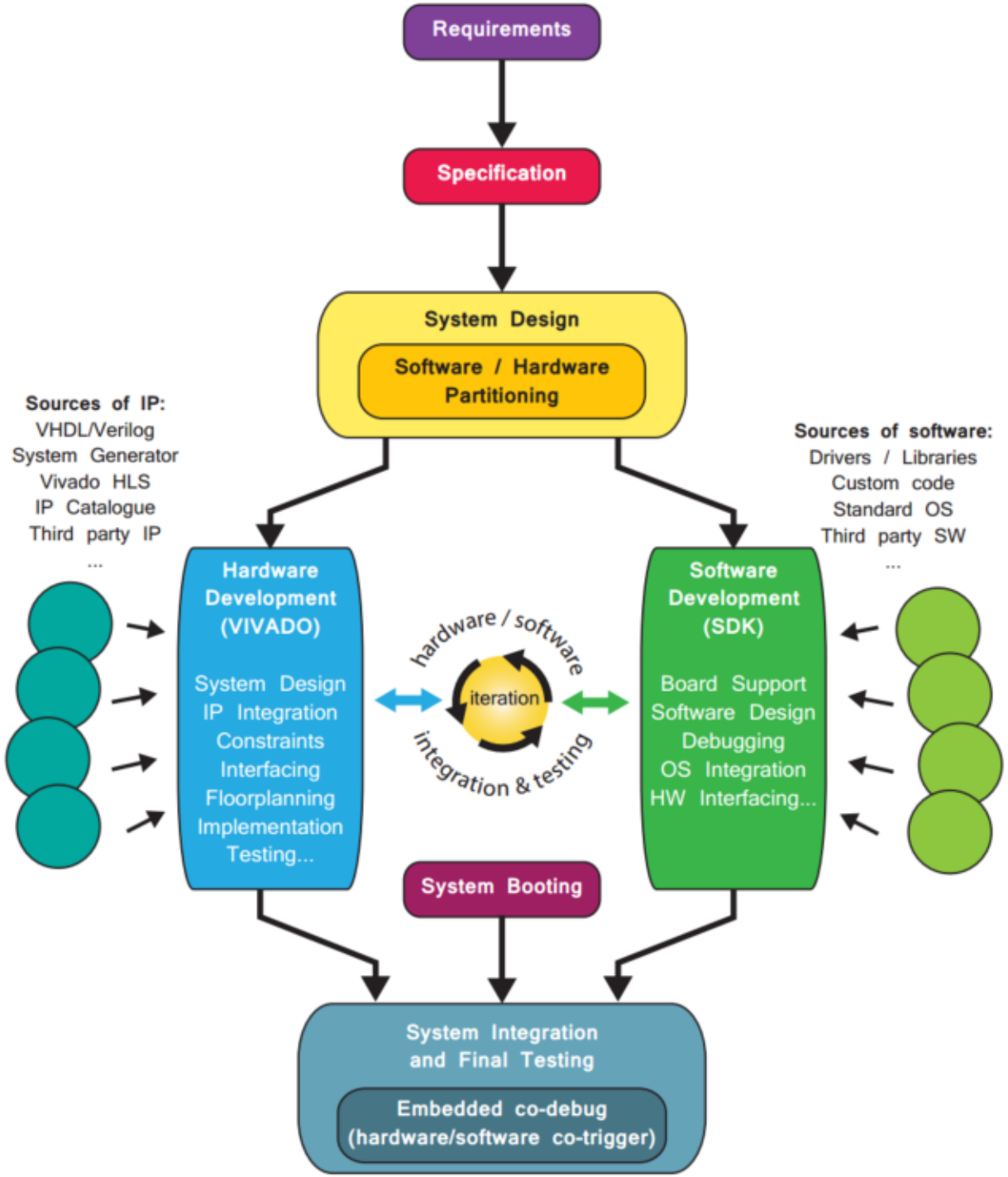

Processing System
Memory Interfaces
7 Series Programmable Logic
Common Peripherals
ARM®
Dual Cortex-A9
MPCore ™ System
Common Peripherals
Custom Peripherals
Common Accelerators
Custom Accelerators

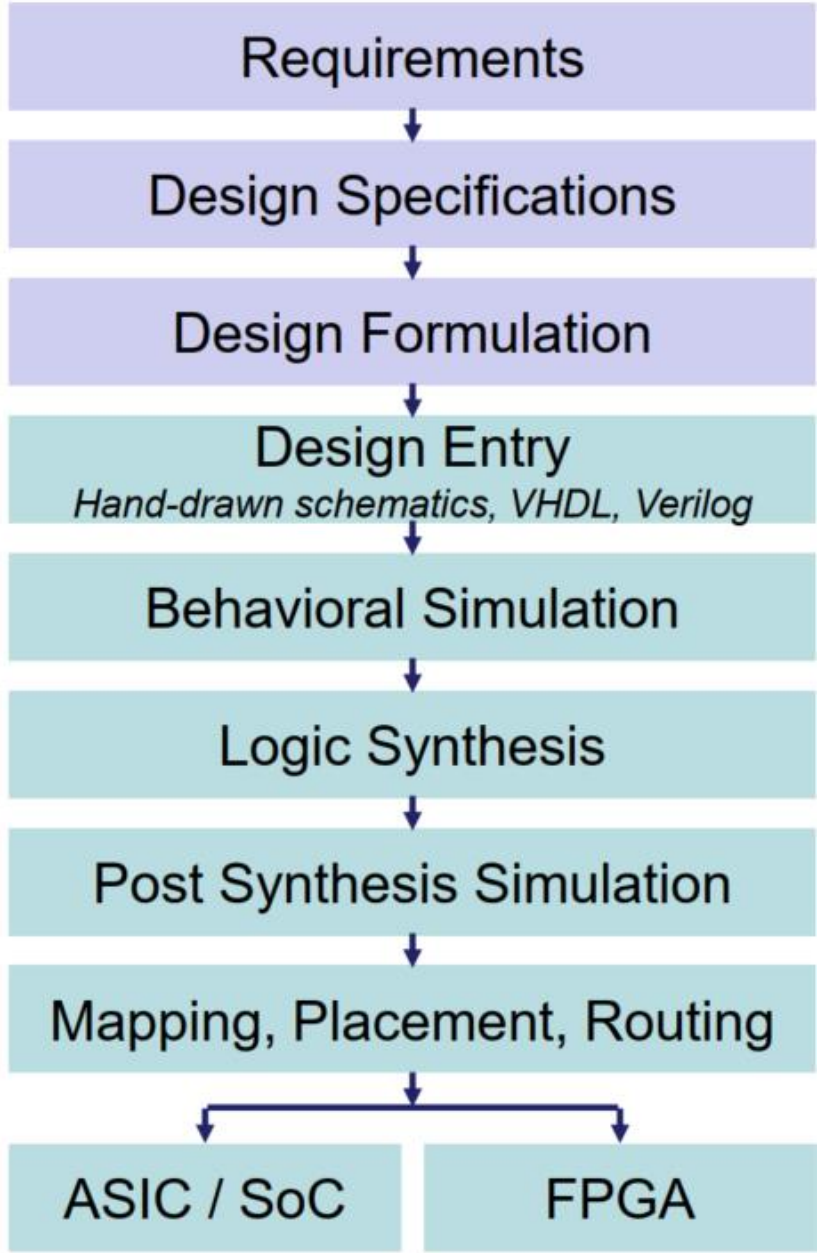
Zynq Evolution



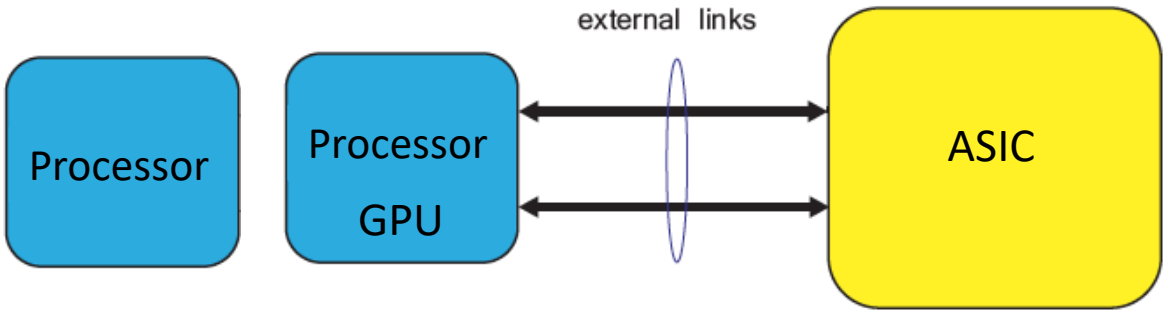
Zynq Evolution

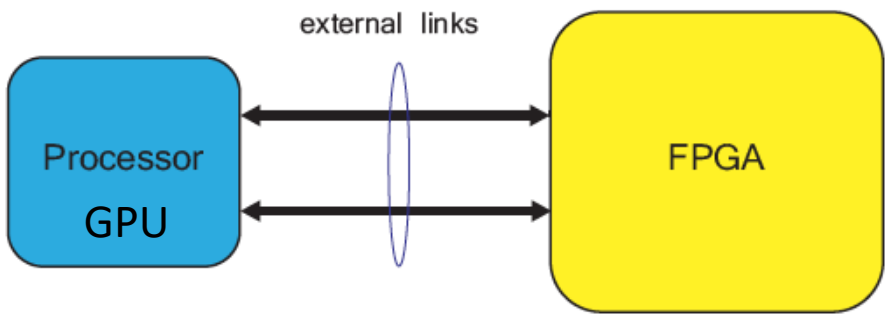


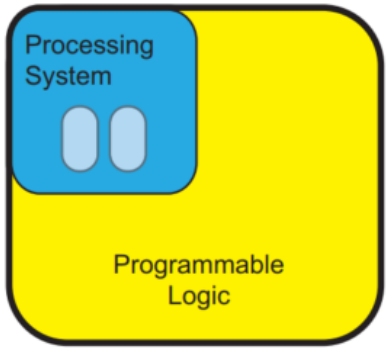

Requirements
Specification
System Design
Software / Hardware Partitioning
Sources of IP:
VHDL/Verilog
System Generator
Vivado HLS
IP Catalogue
Third party IP
...
Sources of software:
Drivers / Libraries
Custom code
Standard OS
Third party SW
...
Hardware Development (VIVADO)
System Design
IP Integration
Constraints
Interfacing
Floorplanning
Implementation
Testing...
hardware / software
iteration
integration & testing
Software Development (SDK)
Board Support
Software Design
Debugging
OS Integration
HW Interfacing...
System Booting
System Integration and Final Testing
Embedded co-debug (hardware/software co-trigger)


Requirements
Design Specifications
Design Formulation
Design Entry
Hand-drawn schematics, VHDL, Verilog
Behavioral Simulation
Logic Synthesis
Post Synthesis Simulation
Mapping, Placement, Routing
ASIC / SoC
FPGA

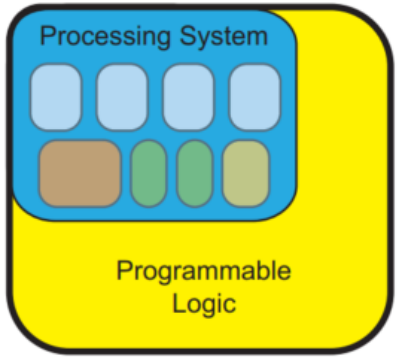
SoC Architectures



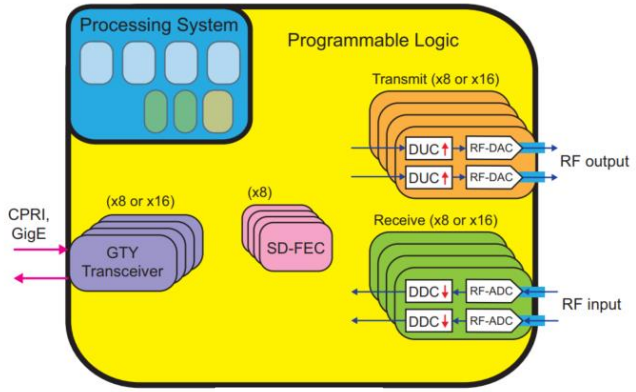




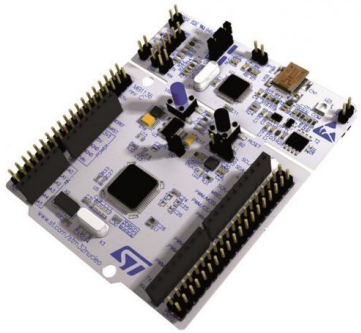
APSOC



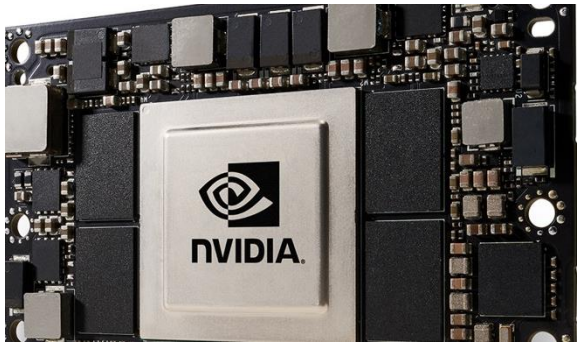
MPSoC



RFSOC



Heterogenous All Programmable SoC



Zynq Book by Xilinx