

# Real Time Embedded Systems

**"System On Programmable Chip"**

**NIOS II – Avalon Bus**

René Beuchat

Laboratoire d'Architecture des Processeurs

*rene.beuchat@epfl.ch*

## Goal :

- To understand the architecture of an embedded system on FPGA
- To be able to design a specific interface
- To be able to construct a full system based on a standard softcore bus in a FPGA and using blocs modules
- To understand, use and program a softcore processor

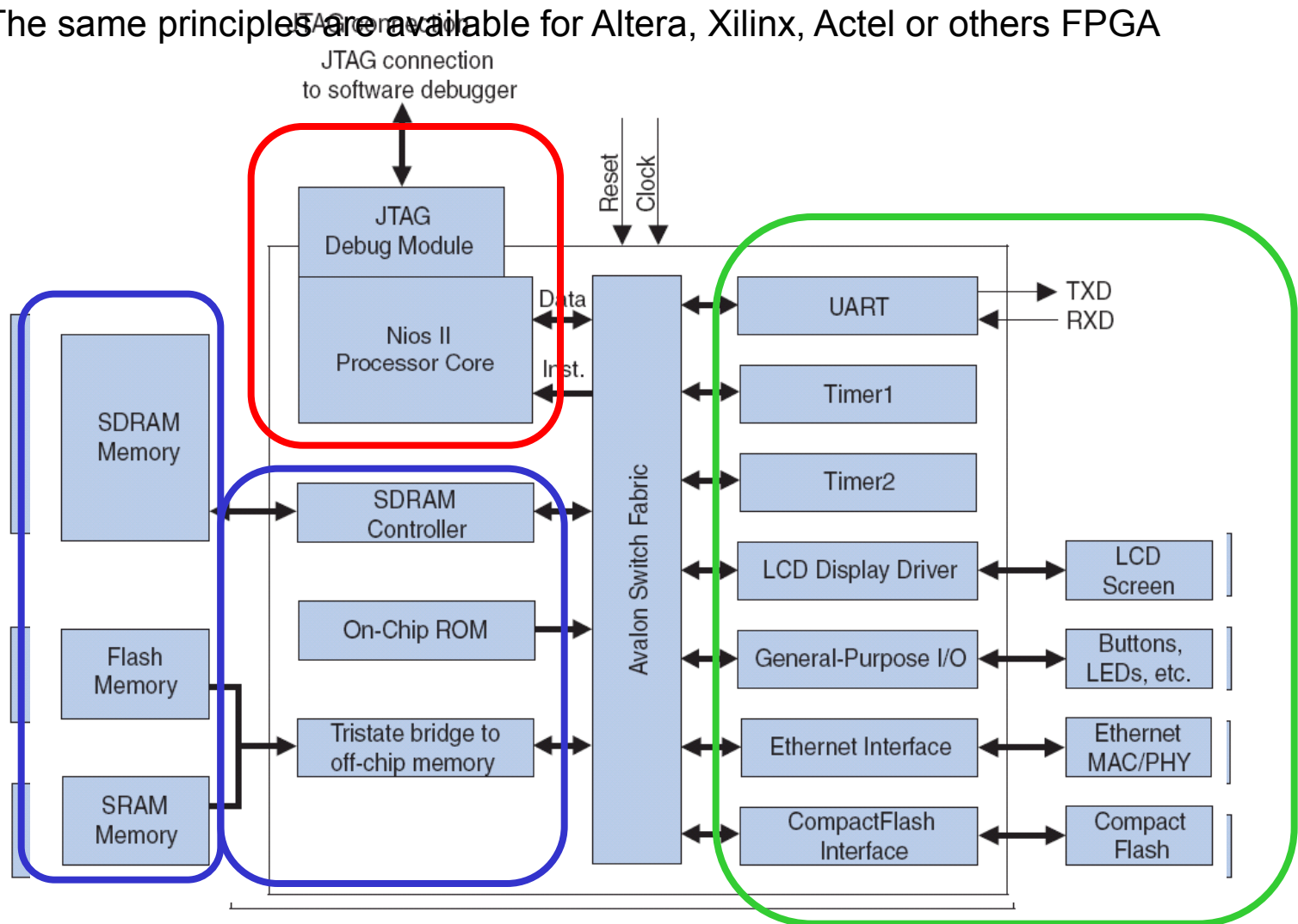
## Contents

- NIOS II a softcore processor
- System On FPGA
- Avalon Bus
- Design of a specific slave programmable interface on Avalon
  
- Reference:  
<http://www.altera.com/literature/lit-nio2.jsp>

- Softcore Processor from Altera
  - A processor implemented with Logic Elements (LUT+DFF) in a FPGA
  - A processor synthesized by a compiler and placed & routed on the FPGA
  - A processor described by a HDL language(VHDL/Verilog/...)
- 32 bits Architecture
- 3 versions
- 256 instructions available for user implementation

# NIOS II – Embedded system NIOSII/Avalon Architecture

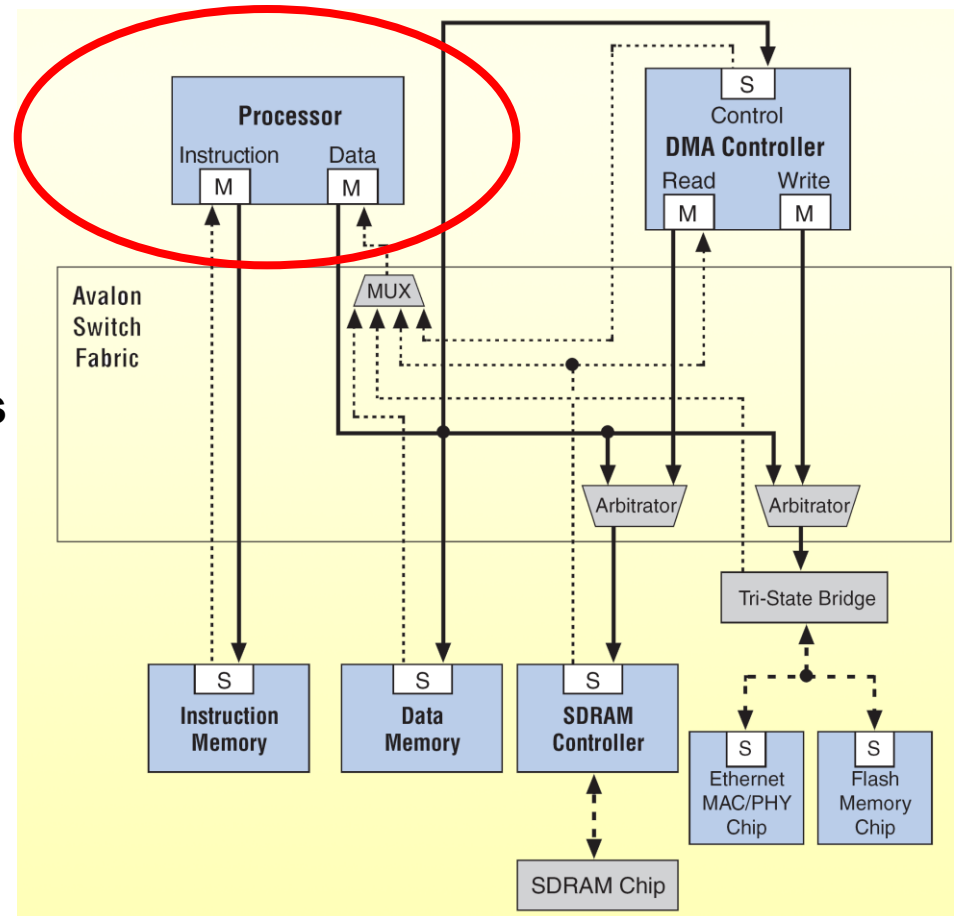
Note: The same principles are available for Altera, Xilinx, Actel or others FPGA



# AVALON Switch Fabric

## Some Avalon specifications :

- Multi-Master
- Arbitrage « slave-side »
- Concurrent Master-Slave Access
- Synchronous transfers

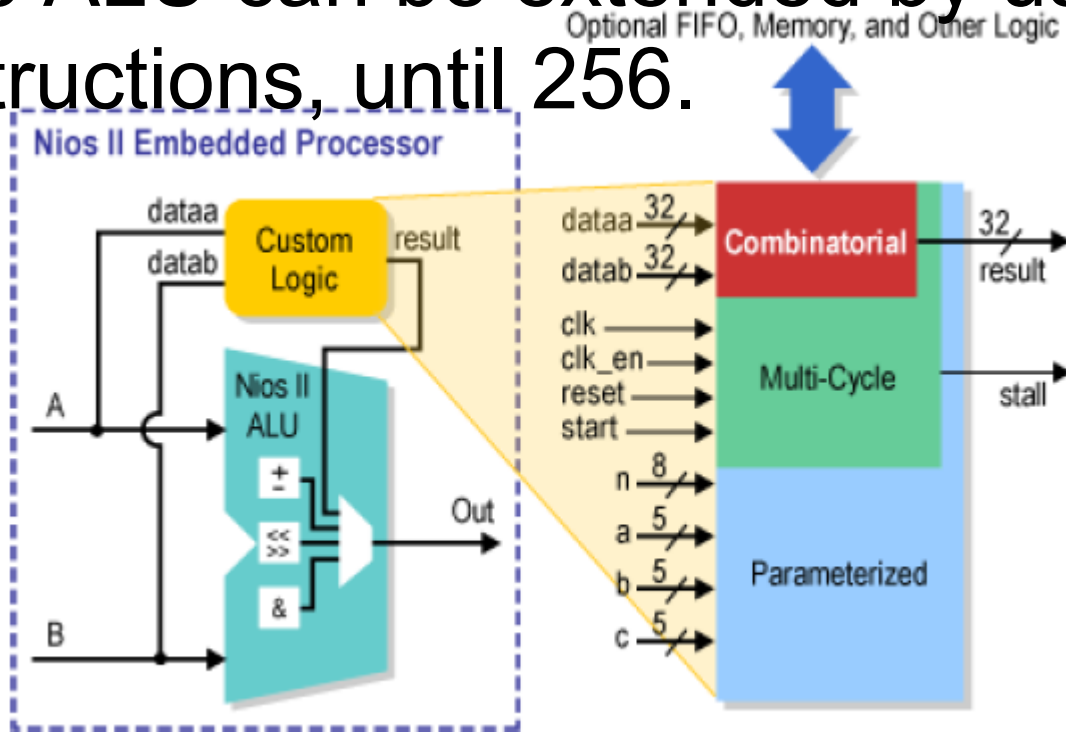


## 3 processors architectures

	<b>Nios II /f Fast</b>	<b>Nios II /s Standard</b>	<b>Nios II /e Economy</b>
<b>Pipeline</b>	6 Stage	5 Stage	None
<b>Multiplier *</b>	1 Cycle	3 Cycle	None
<b>Branch Prediction</b>	Dynamic	Static	None
<b>Instruction Cache</b>	Configurable	Configurable	None
<b>Data Cache</b>	Configurable	None	None

# Nios II Processor, user instructions

- The ALU can be extended by user own instructions, until 256.

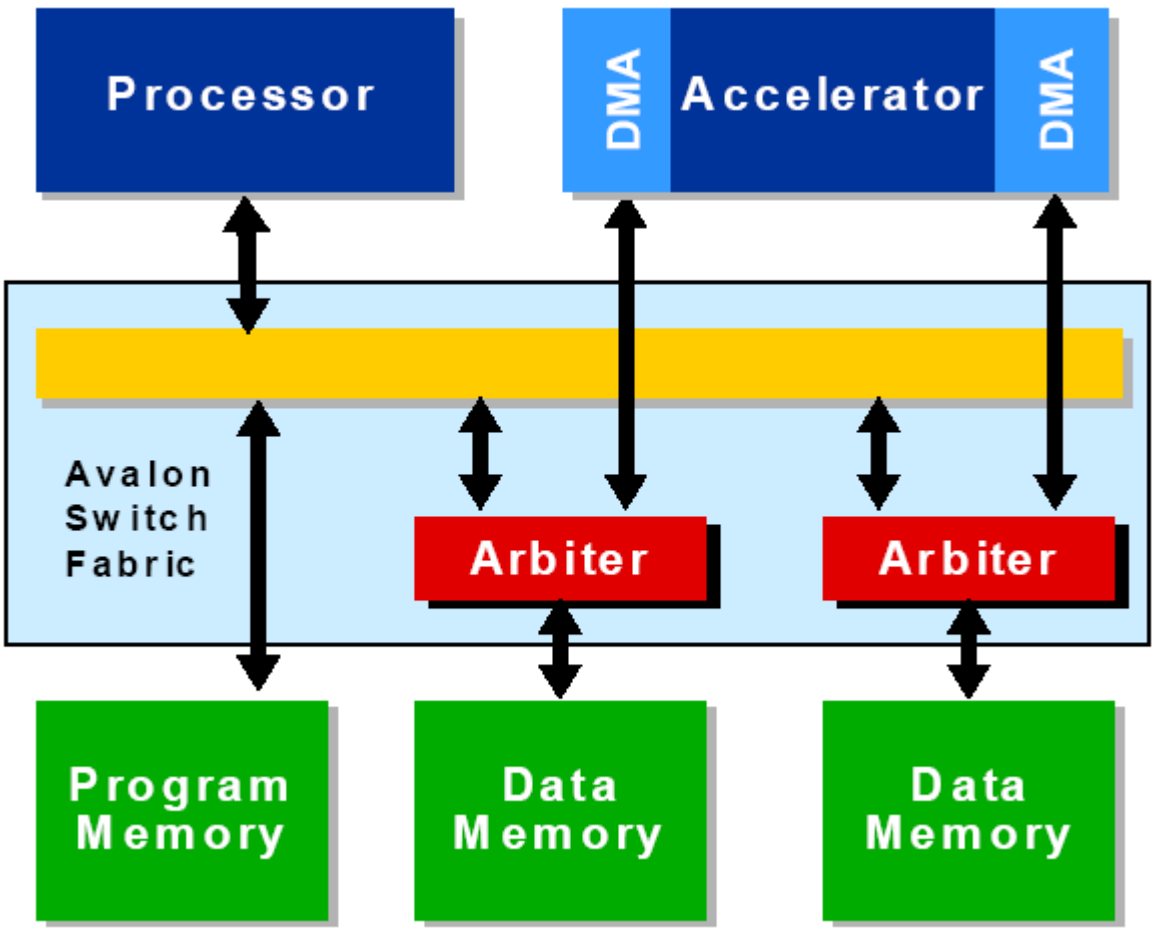




- The instructions can be:
  - Combinatorial, single clock cycle
  - Multi-cycles, synchronized by clk and stall
  - Parameterized
- They can have access to all the FPGA resources
- They can use their own internal registers

- For cycles consuming operations, a **hardware accelerator** can be included/developed
- A **Master unit** which has access to Memory and Programmable Interfaces for accelerated operations or with hard real time constrains

# NIOS II Processor, hardware accelerator

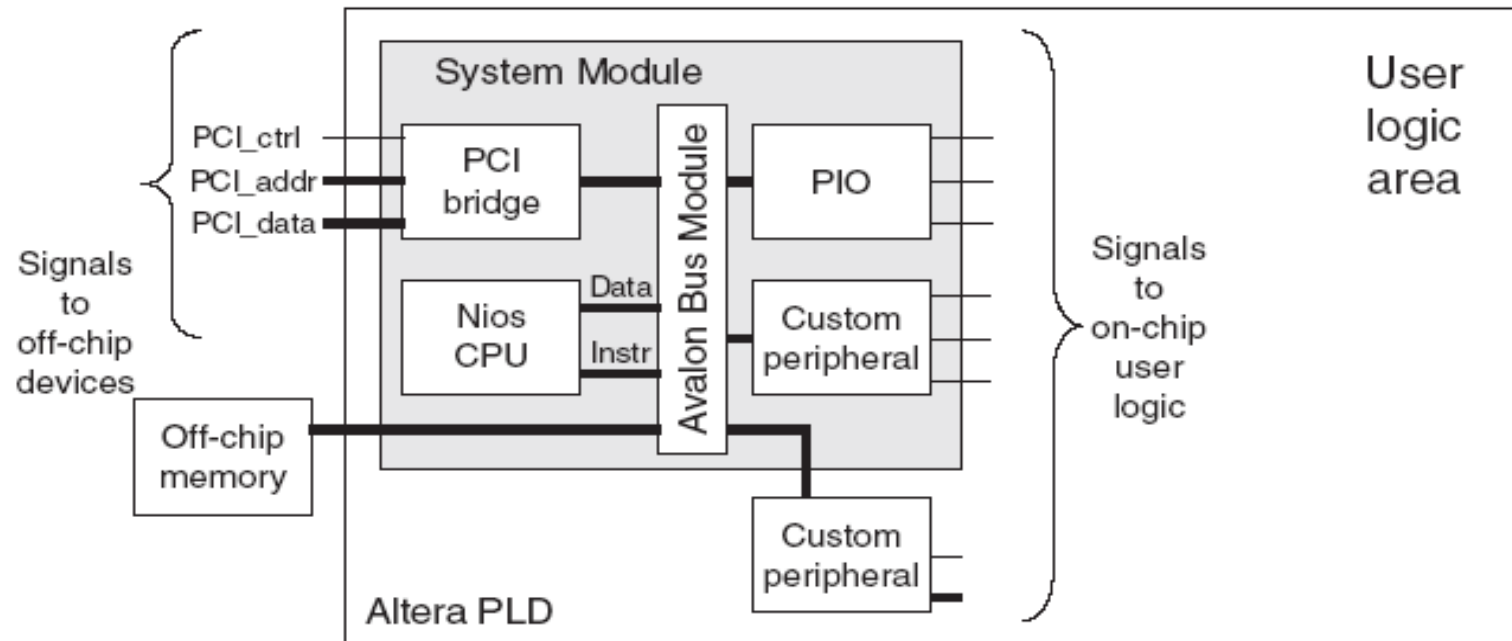


- Classical architecture
  - Processor
  - Memories
  - Input/Output (programmable) interface
  - Address bus
  - Data Bus (tri-state)
  - General decoder

- SOPC architecture (Altera)
  - Processor
  - Memories
  - Input/Output (programmable) interface
  - Address bus
  - **Separated Data Bus In/Out → multiplexers**
  - Local decoder on the Avalon bus
  - Bus transfers size adaptation is done at Avalon bus level

# System on FPGA

## example



# Avalon Bus

To interconnect all the masters and slaves inside the FPGA, an generated internal bus :

- Master/Slave modules
- Synchronous bus on clock rising edge
- **Separate data in and data out**
- Wait state by configuration or dynamic
- Hold / Set up available
- Actual version (>1.0) allows data path until **1024 bits** (8, 16, 32, 64, 128, 256, 512, 1024)

# Avalon

## « slave » main signals

Signal Type	Width	Direction	Required	Description
<b>clk</b>	1	In	(No)	Global clk for system module and Avalon bus modules. <b>All transactions synchronous to clk rising edge</b>
<b>nReset</b>	1	In	No	Global Reset of the system
<b>address</b>	1..32	In	No	Address for Avalon bus modules
<b>ChipSelect</b>	1	In	<i>Old signal</i>	<i>Selection of the Avalon bus module</i>
<b>read/ read_n</b>	1	In	No	Read request to the slave
<b>ReadData</b>	8, 16, 32, .. (1024)	Out	No	Read data from the slave module
<b>write/ write_n</b>	1	In	No	Write request to the slave
<b>WriteData</b>	8, 16, 32, .. (1024)	In	No	Data from Master to Slave module
<b>Irq</b>	1	Out	No <sup>25</sup>	Interrupt request to the master



- The **Address**[n .. 0] is used to access a specific register/memory position in the selected module.
- An address is a **word address** view from the slaves. A word has the width of the slave interface: 8, 16, 32, 64, 128, 256, 512 or 1024 bits
- Only the minimum number of addresses is necessary. *Ex: a module with 6 internal registers needs 3 bits of addresses ( $6 < 2^{**3}$ )*

- The **ChipSelect** is generated by the Avalon bus and selects the module, actually is included in read/write signals. *Thus it is deprecated*
- The **Read** and **Write** signals specifies the direction of the transfers and validate the cycle.  
They are provided by a Master and received by the slave modules
- The direction is the view of the Master unit
- **ReadData(..)** and **WriteData(..)** bus transfers the data from (read)/ to (write) the Slaves

- **BE (Byte Enable)** signals specify the bytes to transfers.
  - The number of BE activated are a power of 2
  - They start at a multiple of the size to transfer
- A **master address** is a **byte** address
- A **slave address** is a **word** address
- The Avalon make the addresses translation and the multiple accesses if necessary

# Avalon

## byte enable (BE)

ByteEnable_n[3..0]	Transfer action
0 0 0 0	Full 32 bits access
1 1 0 0	Lower 2 Bytes access
0 0 1 1	Upper 2 Bytes access
1 1 1 0	Lower Byte (0) access
1 1 0 1	Mid Low Byte (1) access
1 0 1 1	Mid Upper Byte (2) access
0 1 1 1	Upper Byte (3) access

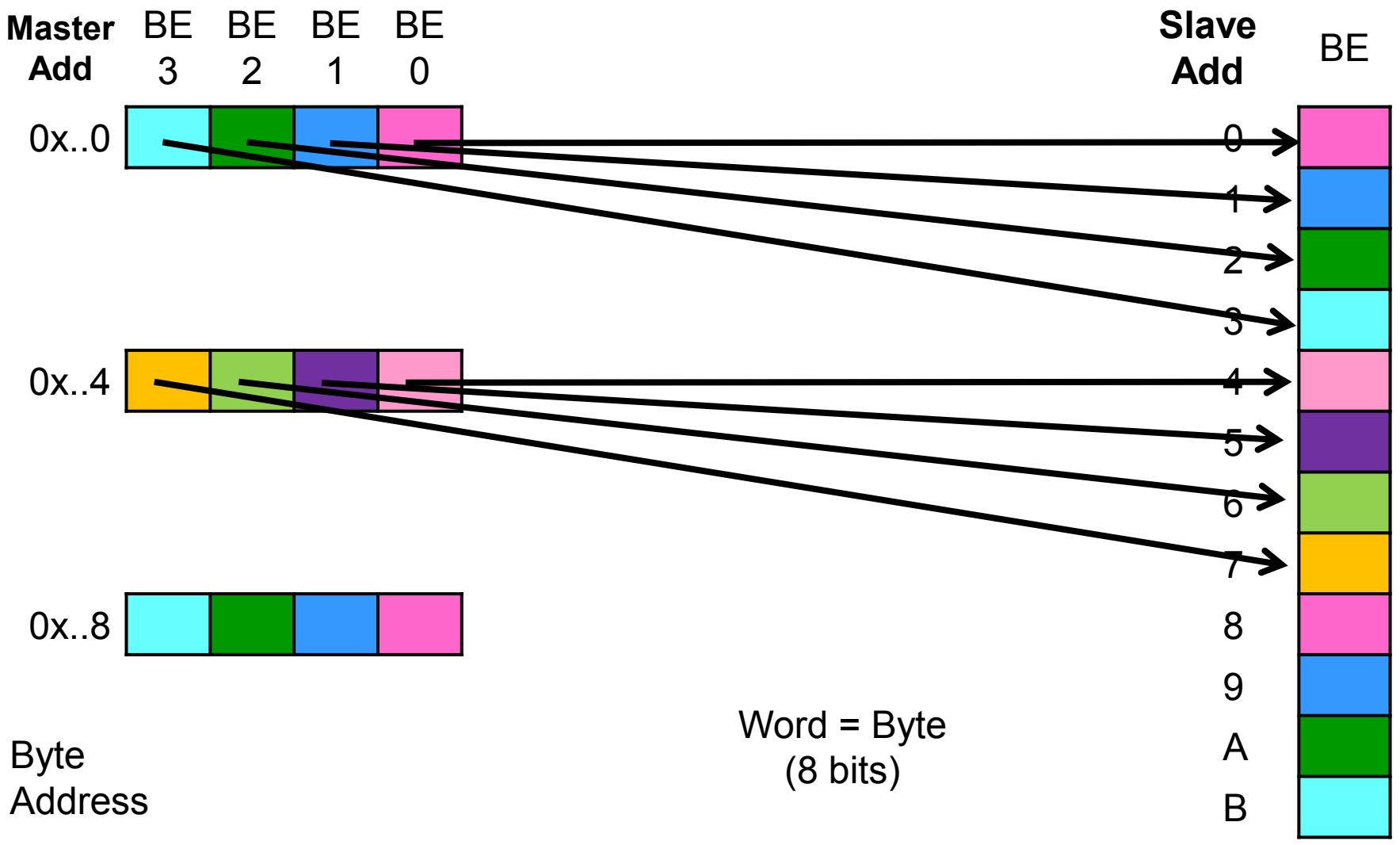
Specify bytes to be transferred

Active low signals in this representation:

- *byteenable\_n*

# Avalon

## Master to slave addresses : Master 32 bits, Slave 8 bits

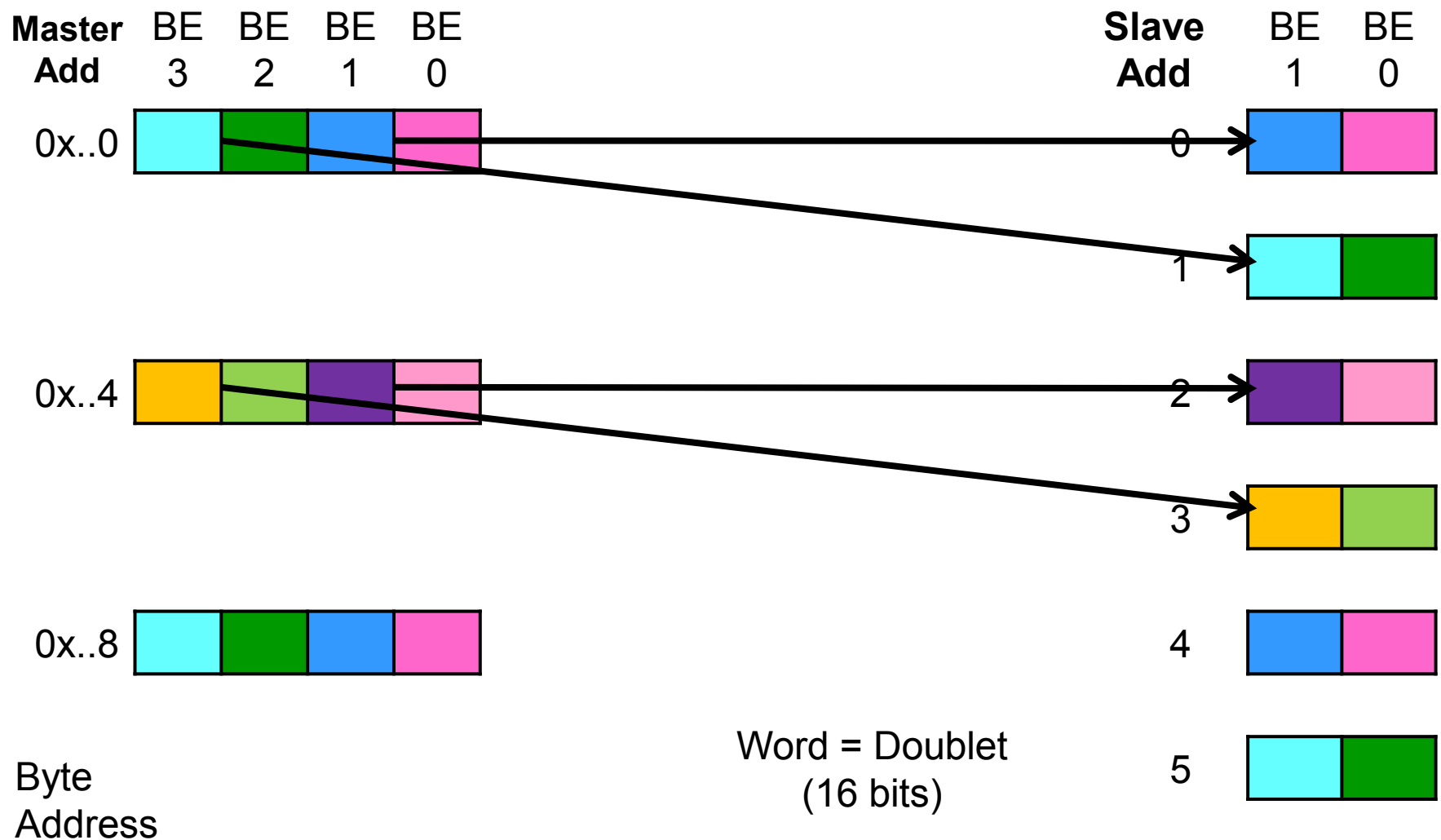


Word = Byte  
(8 bits)

Byte  
Address

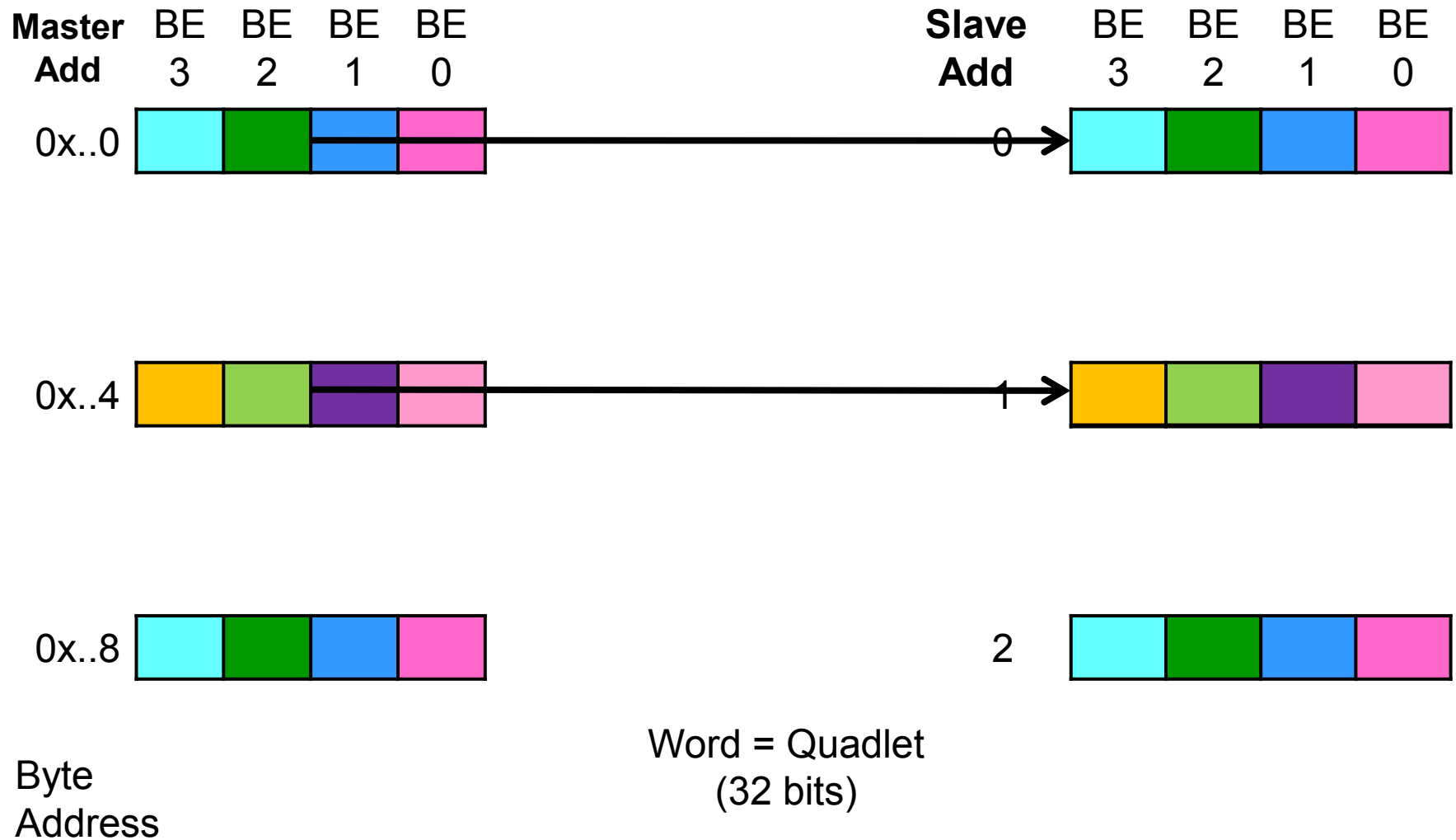
# Avalon

Master to slave addresses : Master 32 bits, Slave 16 bits



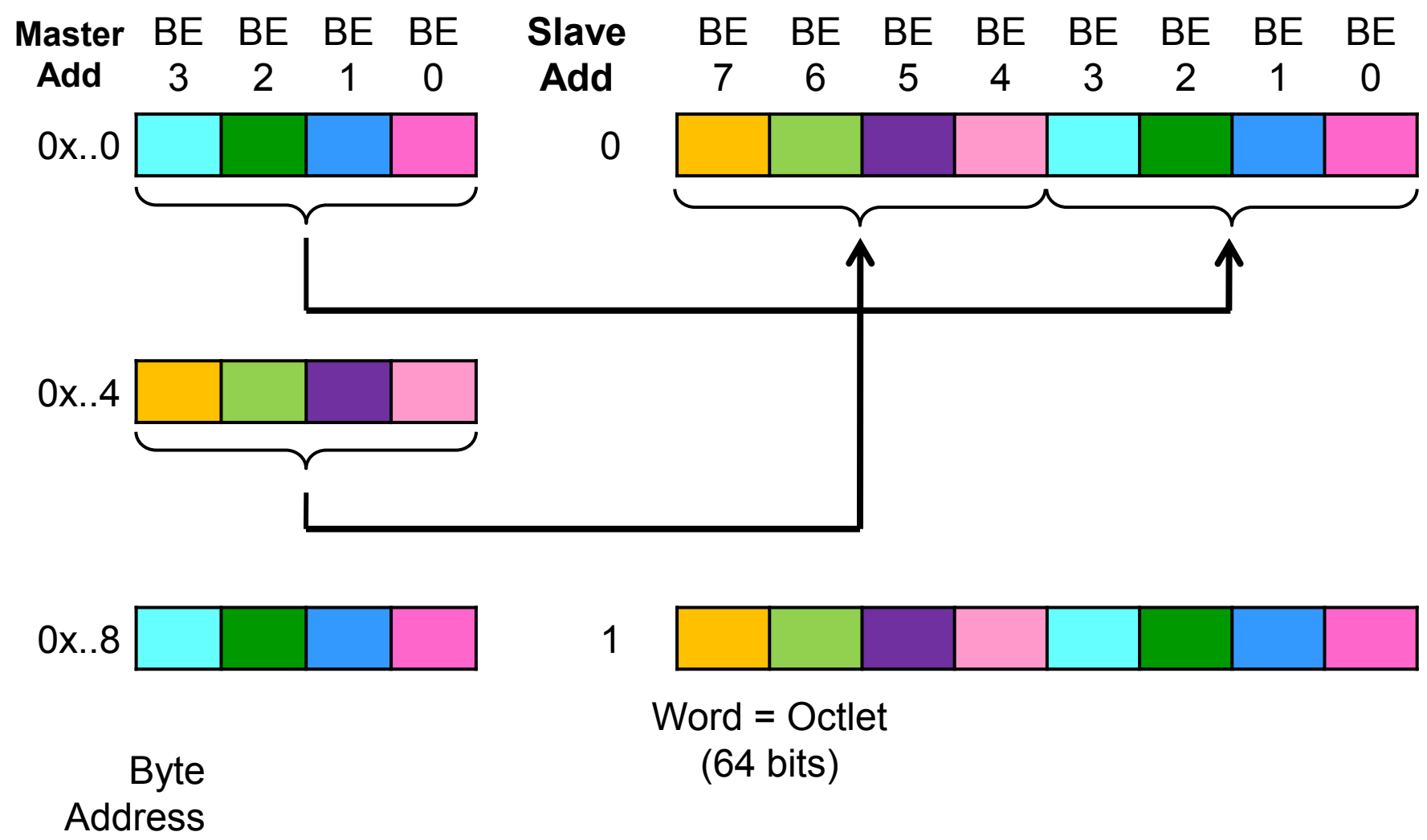
# Avalon

## Master to slave addresses : Master 32 bits, Slave 32 bits



# Avalon

## Master to slave addresses : Master 32 bits, Slave 64 bits





# Avalon

## « slave » signals

Signal Type	Width	Direction	Required	Description
WaitRequest/ WaitRequest_n	1	Out	No	Assert by the slave when it is not able to answer in this clock cycle to read or write access
ByteEnable/ ByteEnable_n	1, 2, 4, 8, ..., 128	In	No	The bytes to transfer
BeginTransfer	1	In	No	Inserted by Avalon fabric at and only at first clock of each transfer
ReadDataValid/ ReadDataValid_n	1	Out	No	For read transfer with <b>variable latency</b> , means data are valid to master
BurstCount	1..11	In	No	Number of burst transfers
BeginBurstTransfer	1	In	No	First cycle of a burst transfer, valid for 1 clock cycle

# Avalon

## « slave » signals

Signal Type	Width	Direction	Required	Description
ReadyForData	1	Out	No	
DataAvailable	1	Out	No	
ResetRequest/ ResetRequest_n	1	Out	No	
ArbiterLock/ ArbiterLock_n	1	In	No	

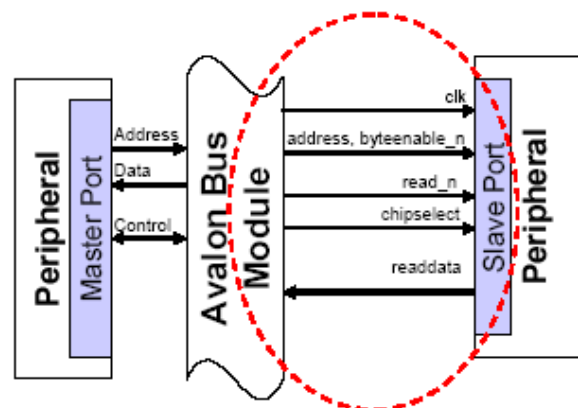
## Slave view of transfers

- Transfers are synchronous on the **rising** edge of the Clk
- Between Clk, the timing relation between signals are NOT relevant

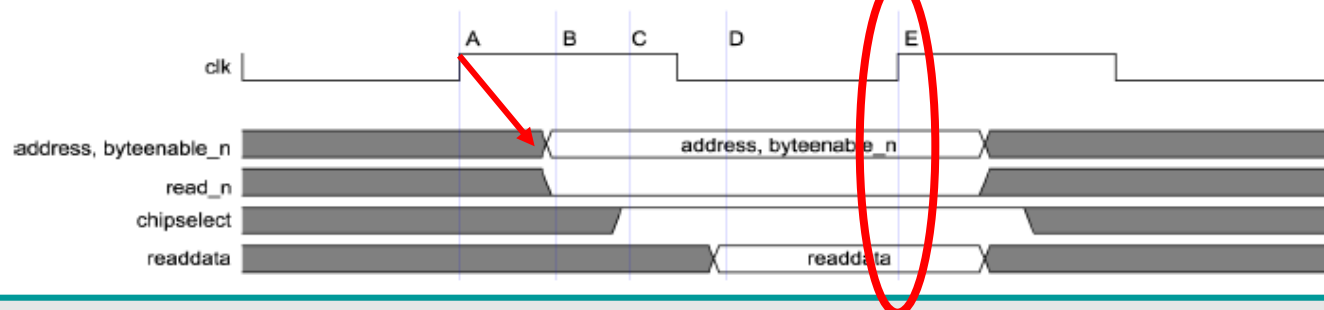
# Avalon (slave view)

## Read transfer, 0 wait, asynchronous peripheral

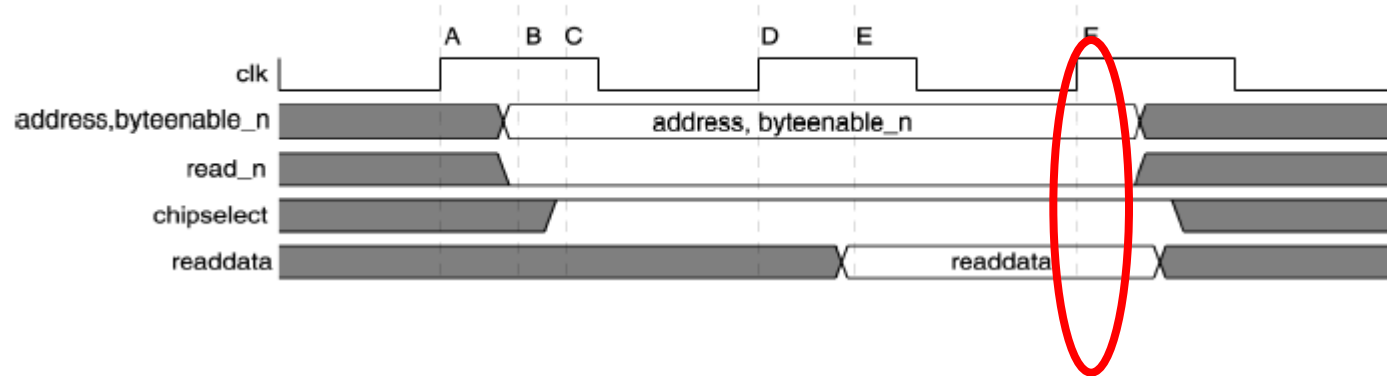
<i>This Example Demonstrates</i>	<i>Relevant PTF Parameters</i>
Read transfer from an asynchronous peripheral	Read_Wait_States = "0"
Zero wait states	Setup_Time = "0"
Zero setup	



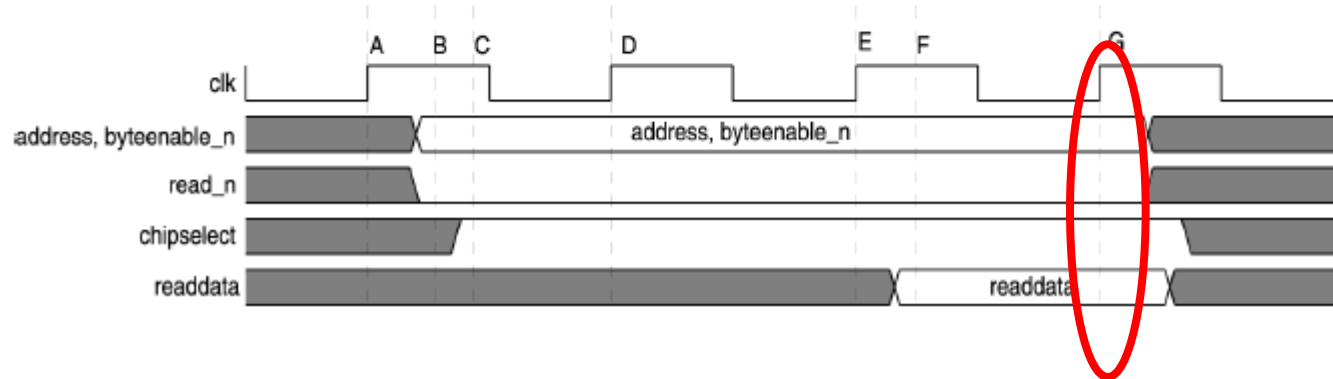
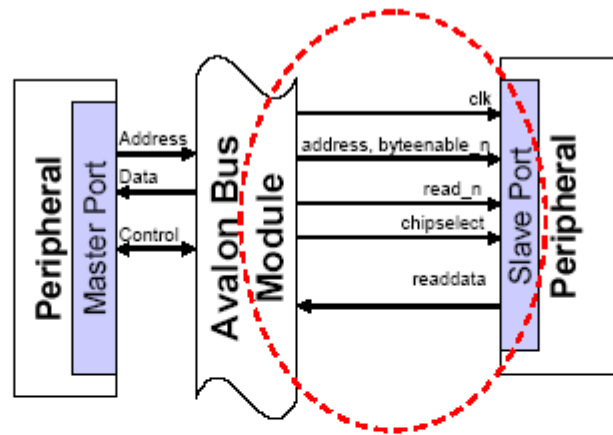
ReadData available at next rising edge of clk (E)



### Wait cycle specified by design

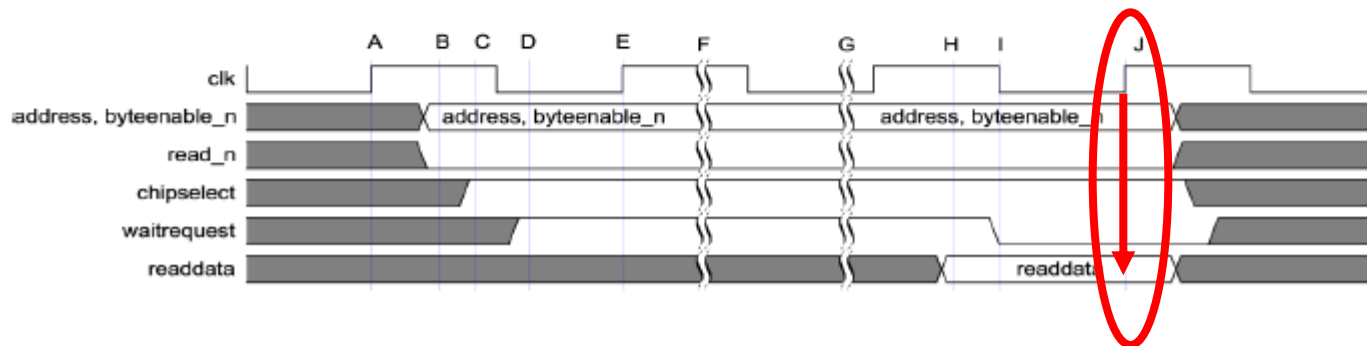
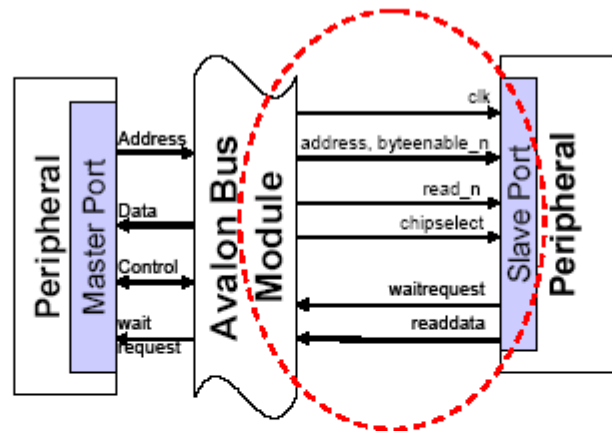


# Avalon (slave view) Read transfer, 2 wait



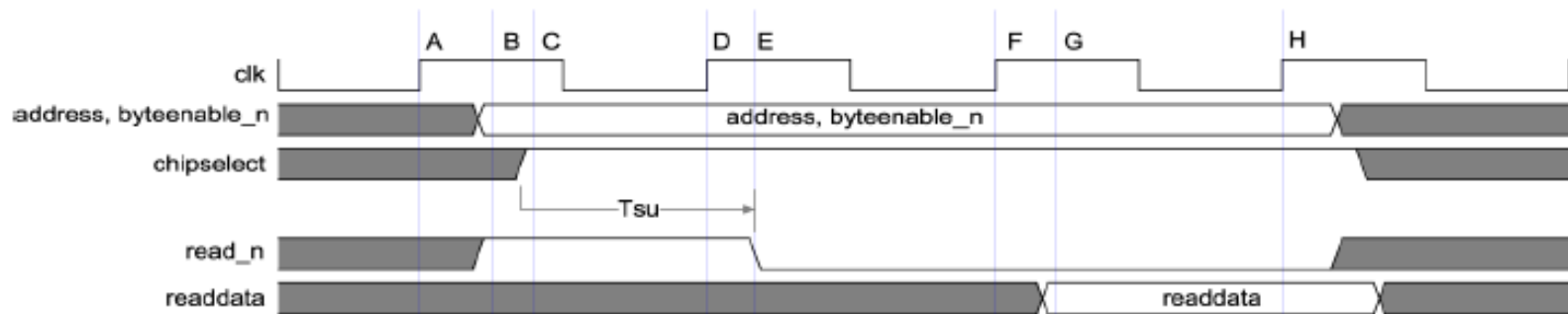
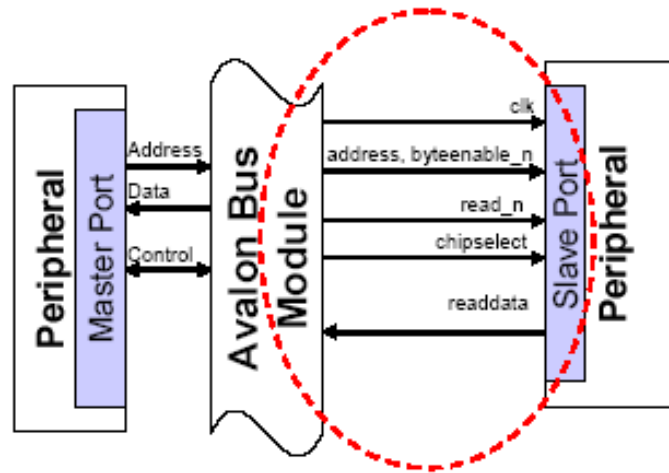
# Avalon (slave view)

## Read transfer, wait request generated by slave device



# Avalon (slave view)

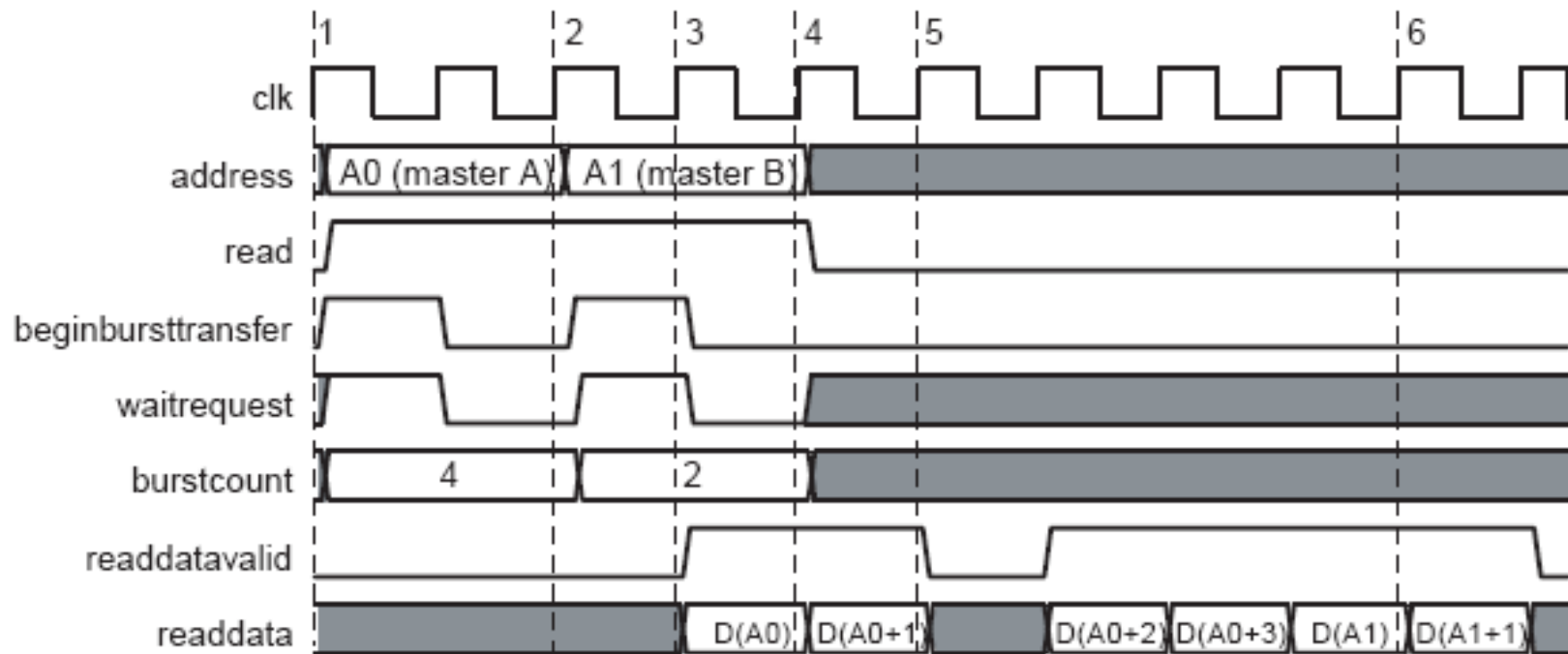
## Read transfer, 1 set up and 1 wait





# Avalon (slave view)

Read transfer, burst of 4 from Master A, 2 from master B

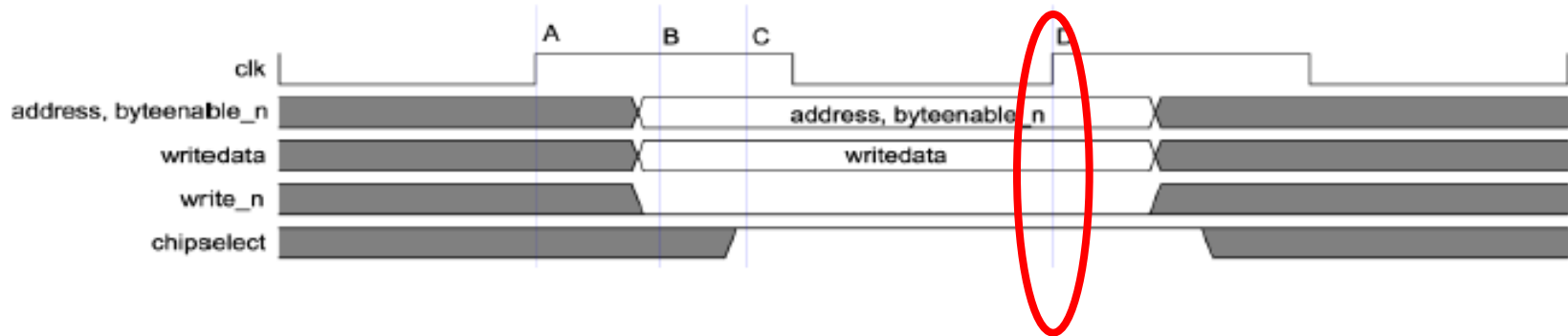
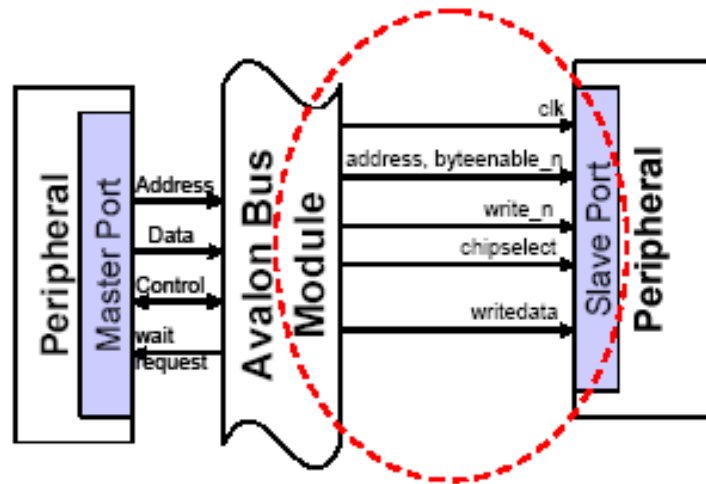


## Pipeline of master access

ReadDataValid activated by slave for each data

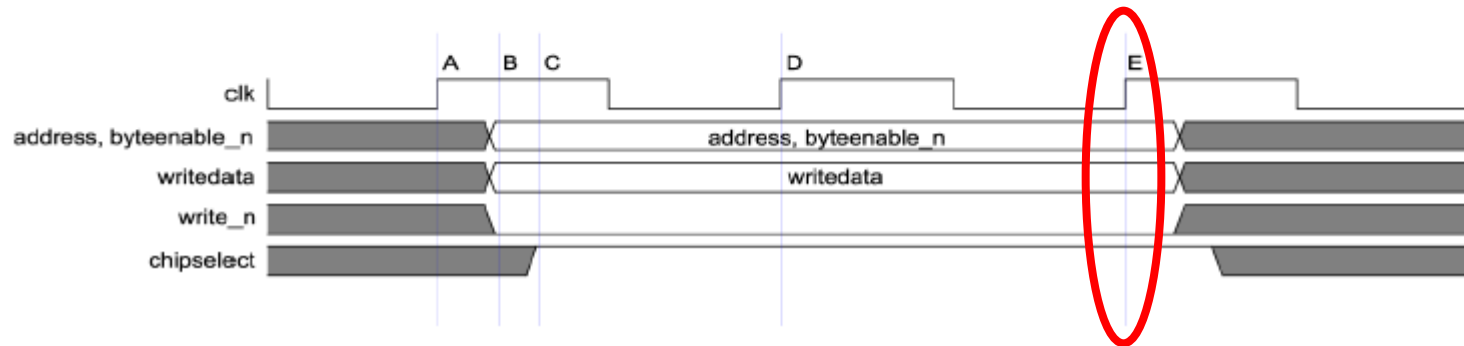
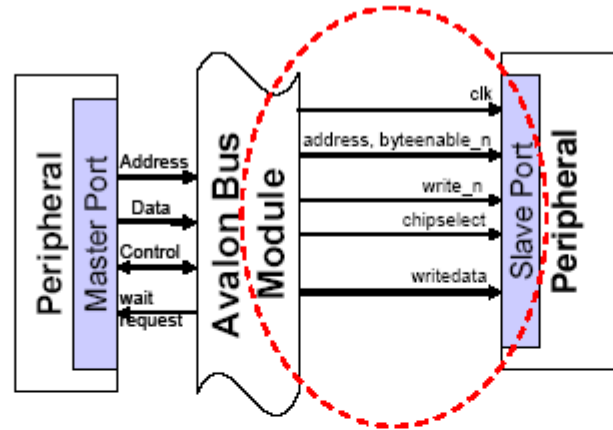
# Avalon (slave view)

## Write transfer, 0 wait



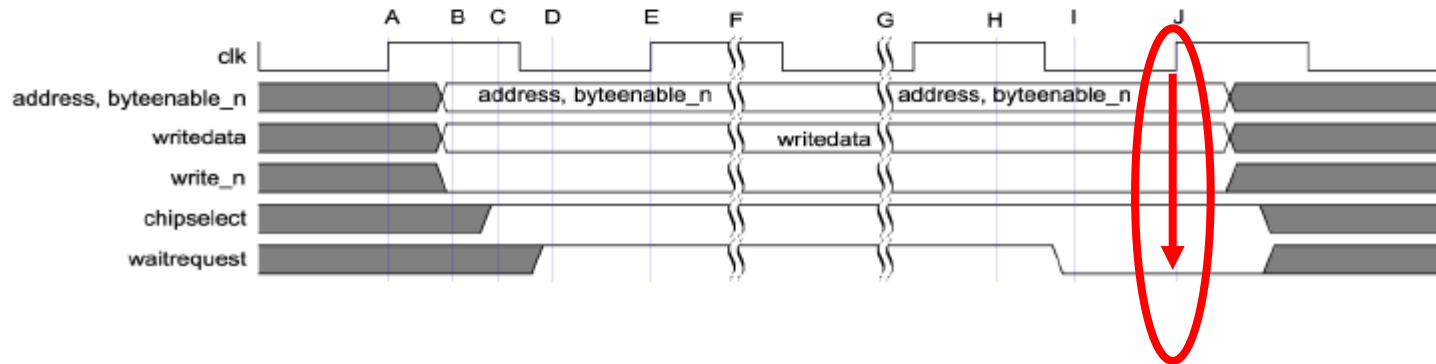
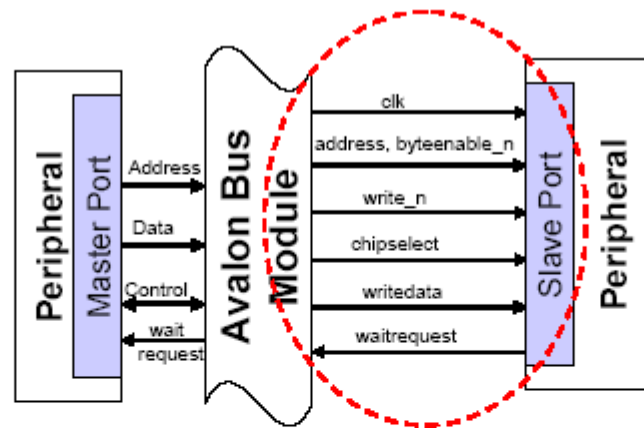
# Avalon (slave view)

## Write transfer, 1 wait



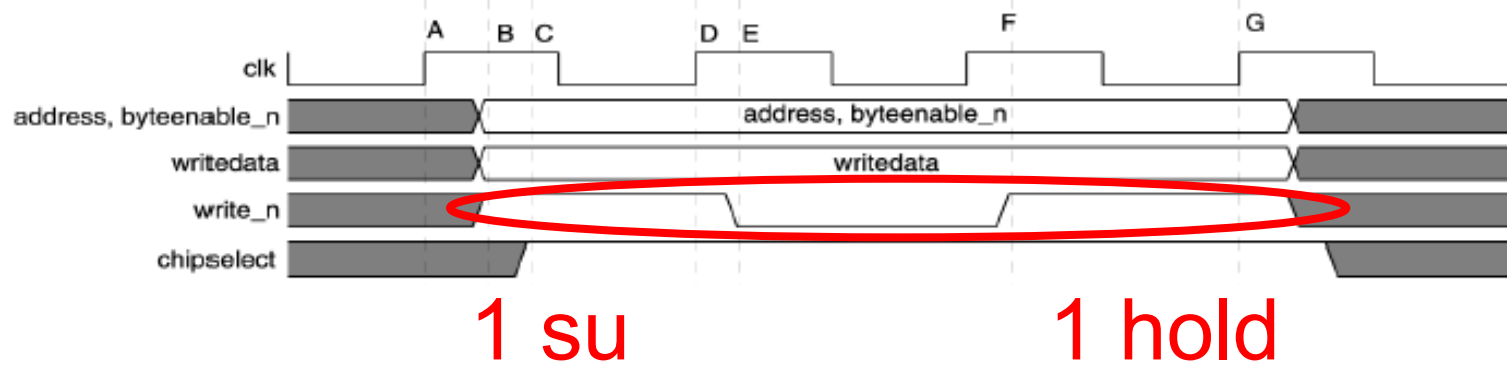
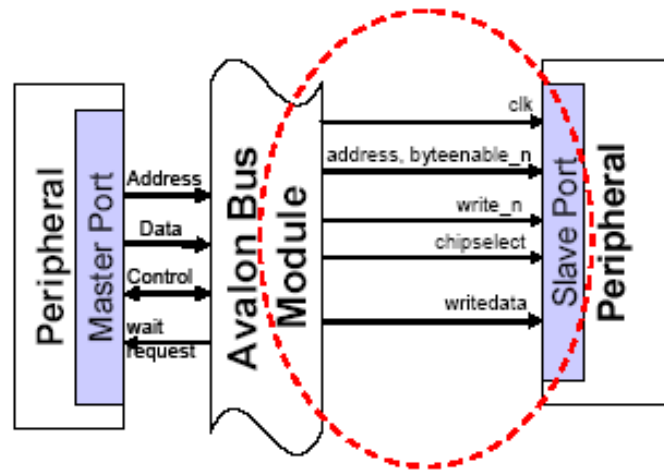
# Avalon (slave view)

## Write transfer, wait request generated by slave



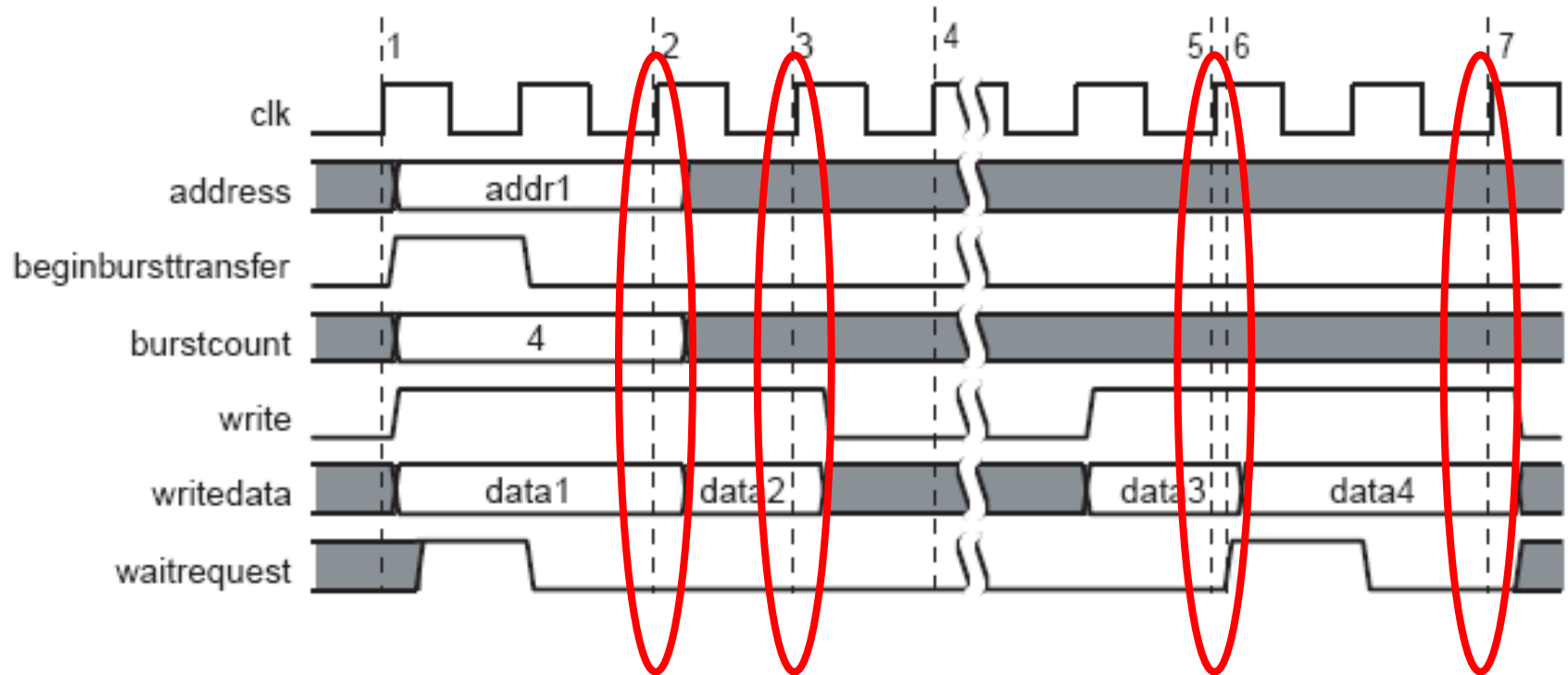
# Avalon (slave view)

## Write transfer, 1 set up, 1 hold, 0 wait



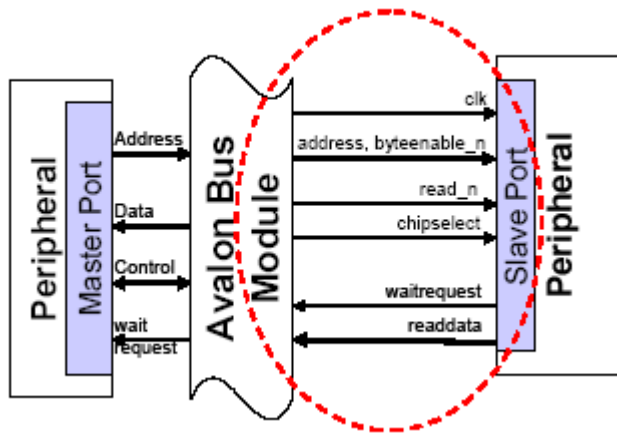
# Avalon (slave view)

## Write transfer, burst transfer of 4, wait request generated by slave



# Avalon (slave view)

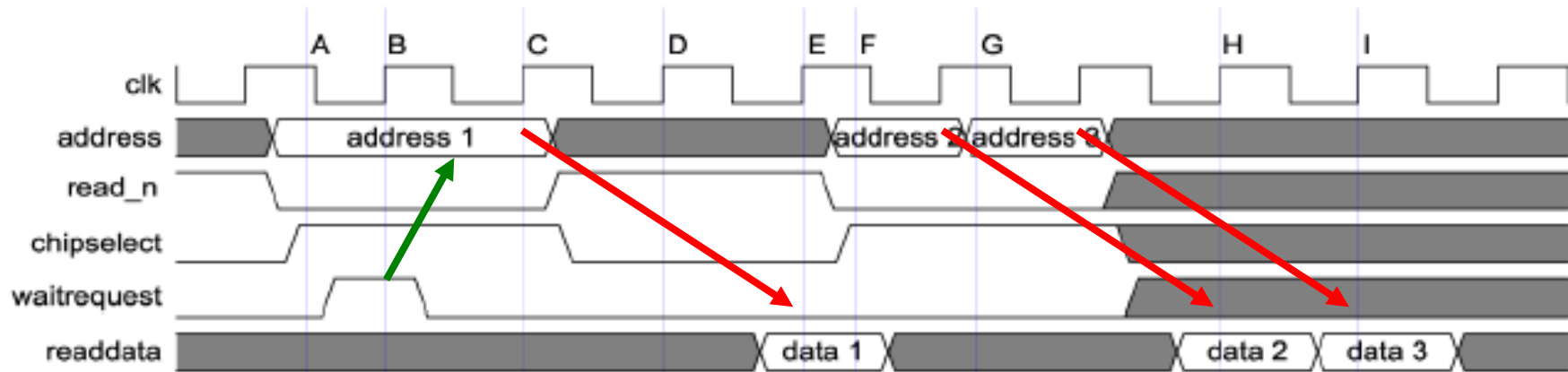
## Read transfers with latency (ex. 2 cycles)



Wait request here means :

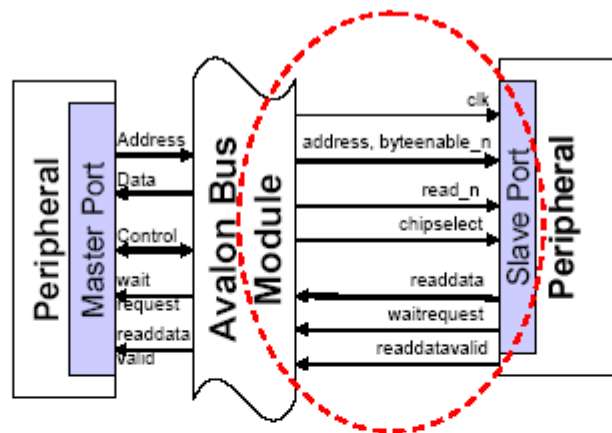
delay address cycle

Fixed latency (here 2)

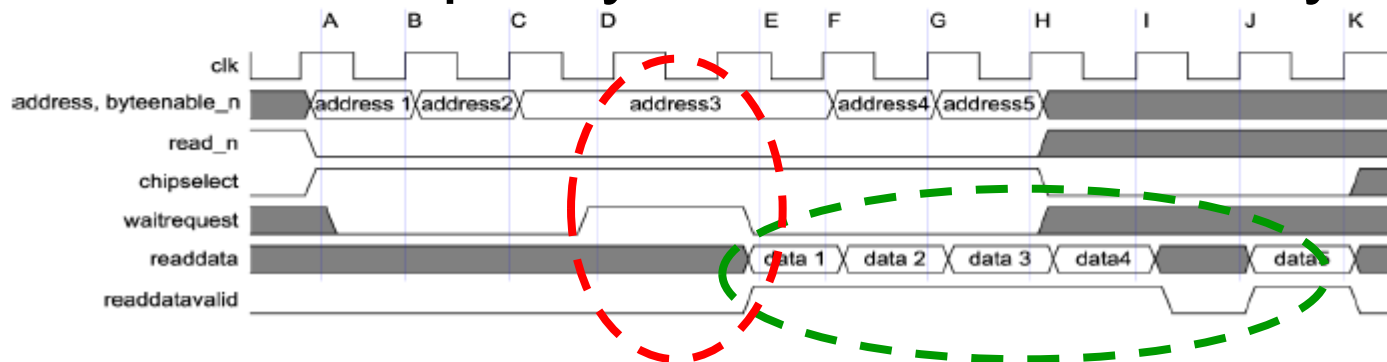


# Avalon (slave view)

Read transfers with latency, and *readdatavalid* generated by slave



*Readdatavalid* specify when data are ready





# Bus avalon

## Master view

- The master start a transfer (read or write)
- It provide the Addresses (32 bits on NIOSII)
- It waits on **WaitRequest** signal to resume the transfer

# Avalon master signals (1)

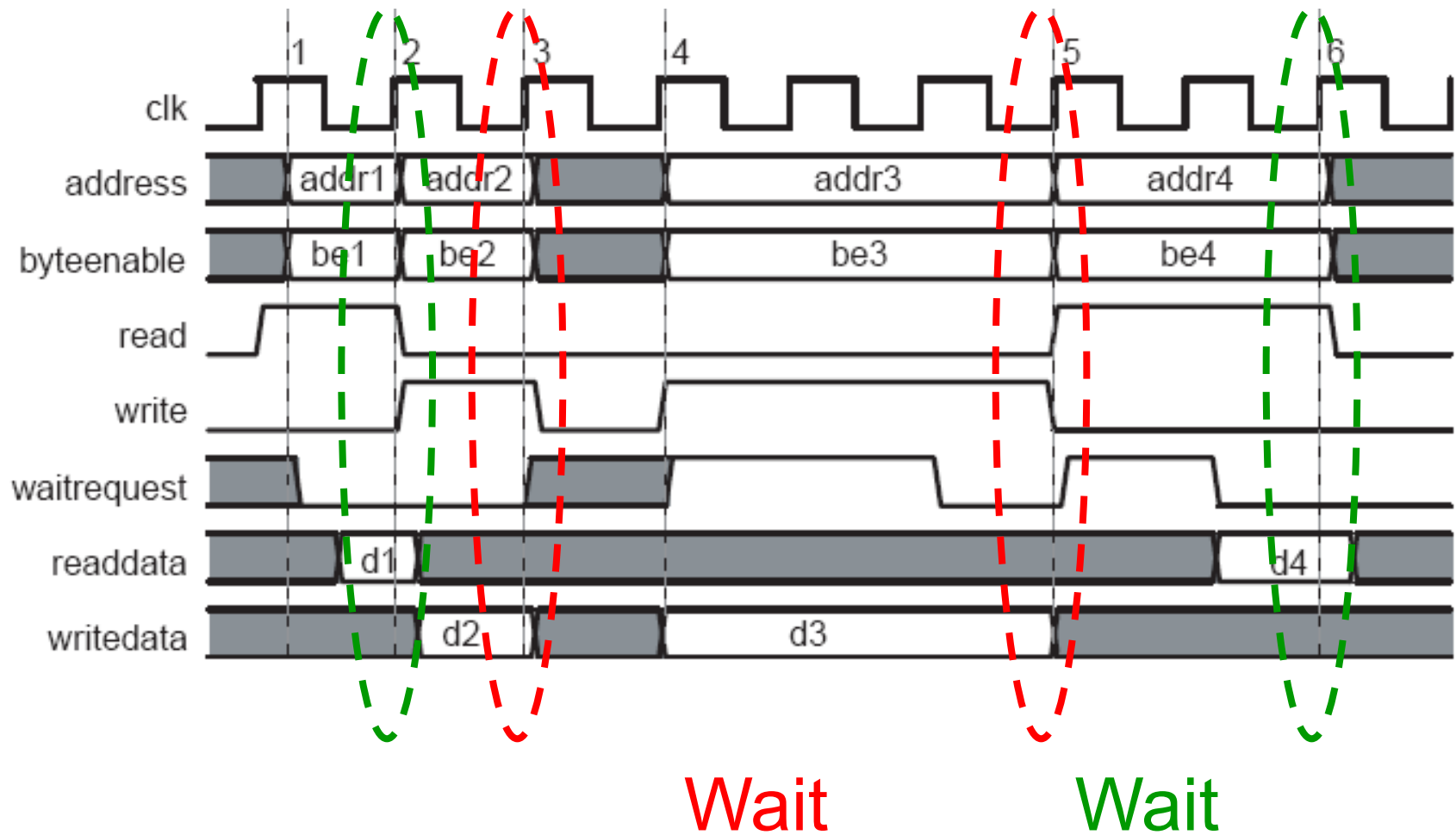
Signal Type	Width	Direction	Required	Description
clk	1	in	yes	Global clock signal for the system module and Avalon bus module. All bus transactions are synchronous to clk.
reset	1	in	no	Global reset signal. Implementation is peripheral-specific.
address	1 - 32	out	yes	Address lines from the Avalon bus module. All Avalon masters are required to drive a byte address on their address output port.
byteenable	0, 2, 4	out	no	Byte-enable signals to enable specific byte lane(s) during transfers to memories of width greater than 8 bits. Implementation is peripheral-specific.
read	1	out	no	Read request signal from master port. Not required if master never performs read transfers. If used, readdata must also be used.
readdata	8, 16, 32	in	no	Data lines from the Avalon bus module for read transfers. Not required if the master never performs read transfers. If used, read must also be used.
write	1	out	no	Write request signal from master port. Not required if the master never performs write transfers. If used, writedata must also be used.

# Avalon master signals (2)

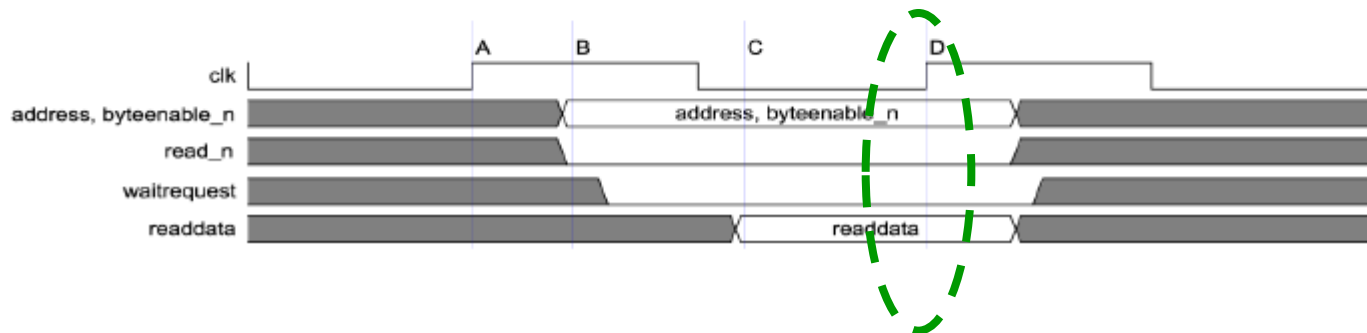
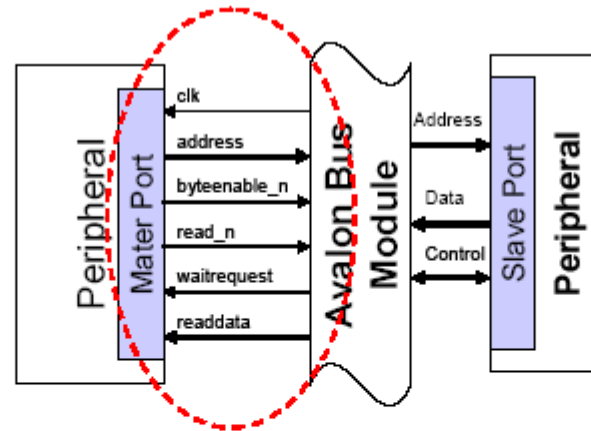
Signal Type	Width	Direction	Required	Description
writedata	8, 16, 32	out	no	Data lines to the Avalon bus module for write transfers. Not required if the master never performs write transfers. If used, <code>write</code> must also be used.
waitrequest	1	in	yes	Forces the master port to wait until the Avalon bus module is ready to proceed with the transfer.
irq	1	in	no	Interrupt request has been flagged by one or more slave ports.
irqnumber	6	in	no	The interrupt priority of the interrupting slave port. Lower value has higher priority.
endofpacket	1	in	no	Signal for streaming transfers. May be used to indicate an end of packet condition from the slave to the master port. Implementation is peripheral-specific.
readdatavalid	1	in	no	Signal for read transfers with latency and is for a master only. Indicates that valid data from a slave port is present on the <code>readdata</code> lines. Required if the master is latency-aware.
flush	1	out	no	Signal for read transfers with latency. Master can clear any pending latent read transfers by asserting <code>flush</code> .

# Avalon (Master view)

## Basic fundamental transfers

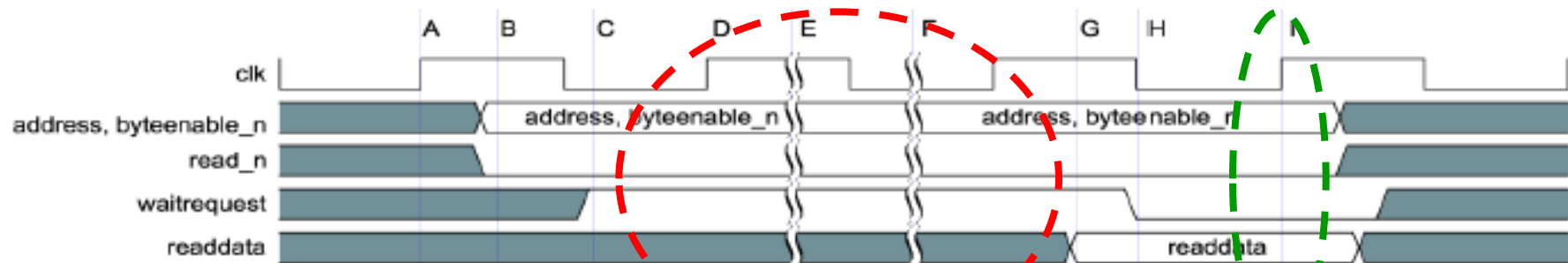


# Avalon (Master view) Read transfer, 0 wait



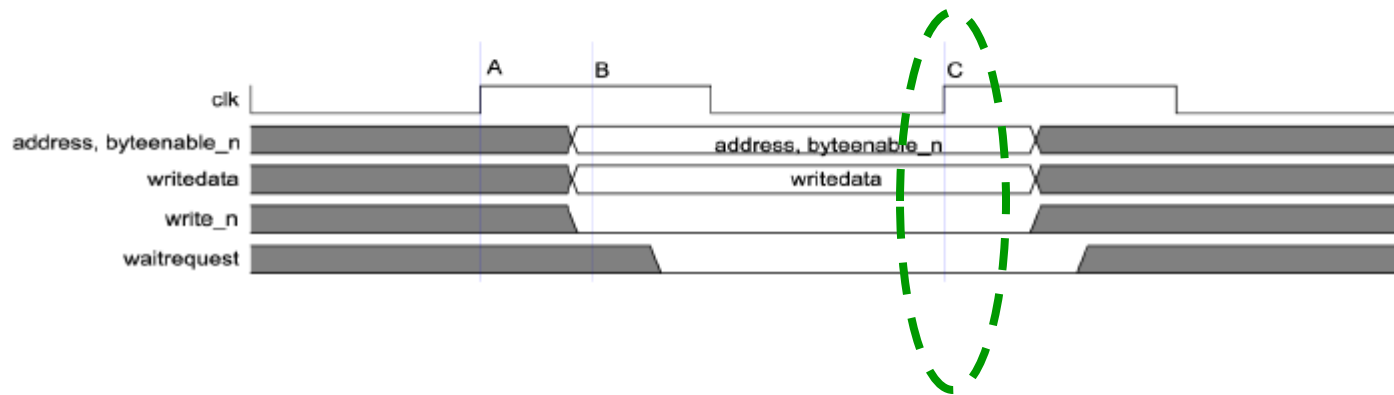
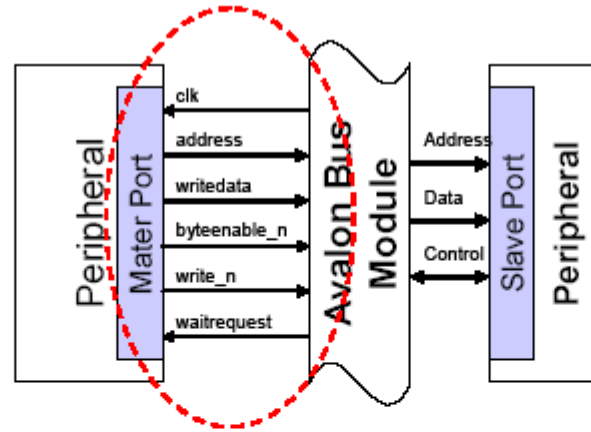
# Avalon (Master view)

## Read transfer, wait generated by slave/Avalon bus



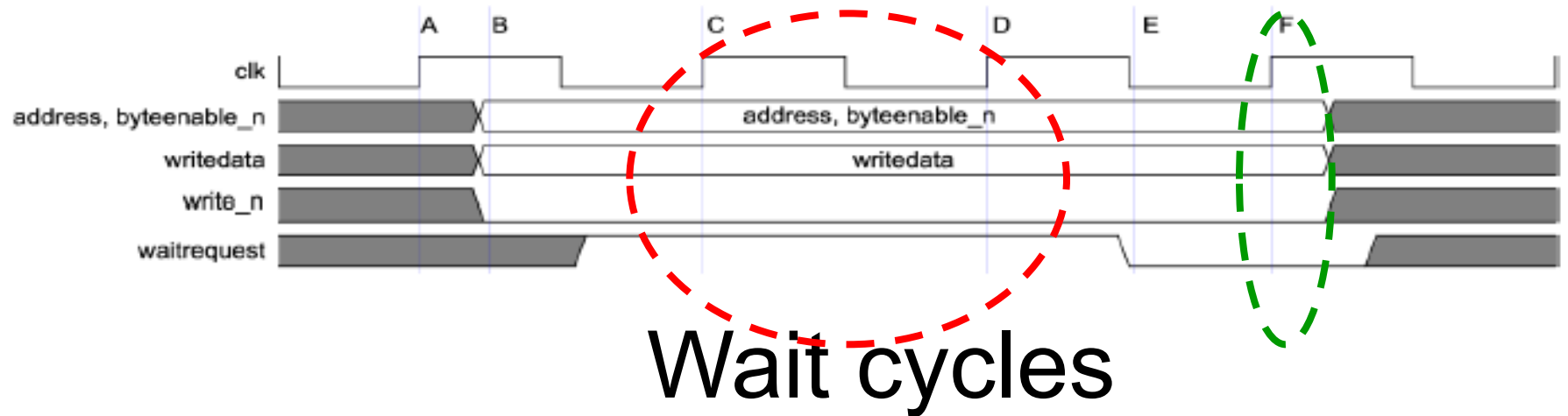
Wait cycles

# Avalon (Master view) Write transfer, 0 wait



# Avalon (Master view)

## Write transfer, wait generated by slave

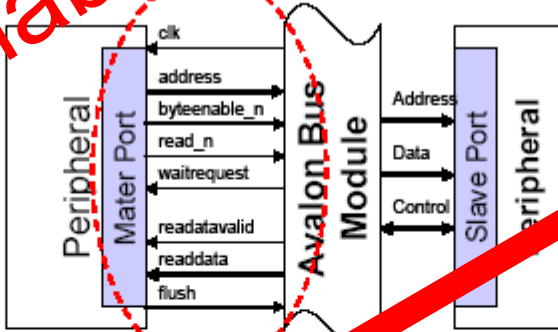




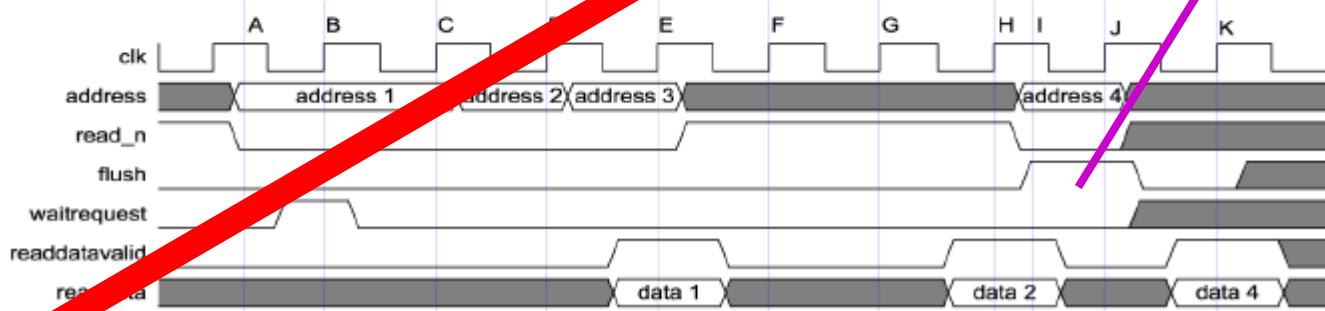
# Avalon (Master view)

Read transfers with latency, and *readdatavalid* generated by slave

**NO MORE AVAILABLE**

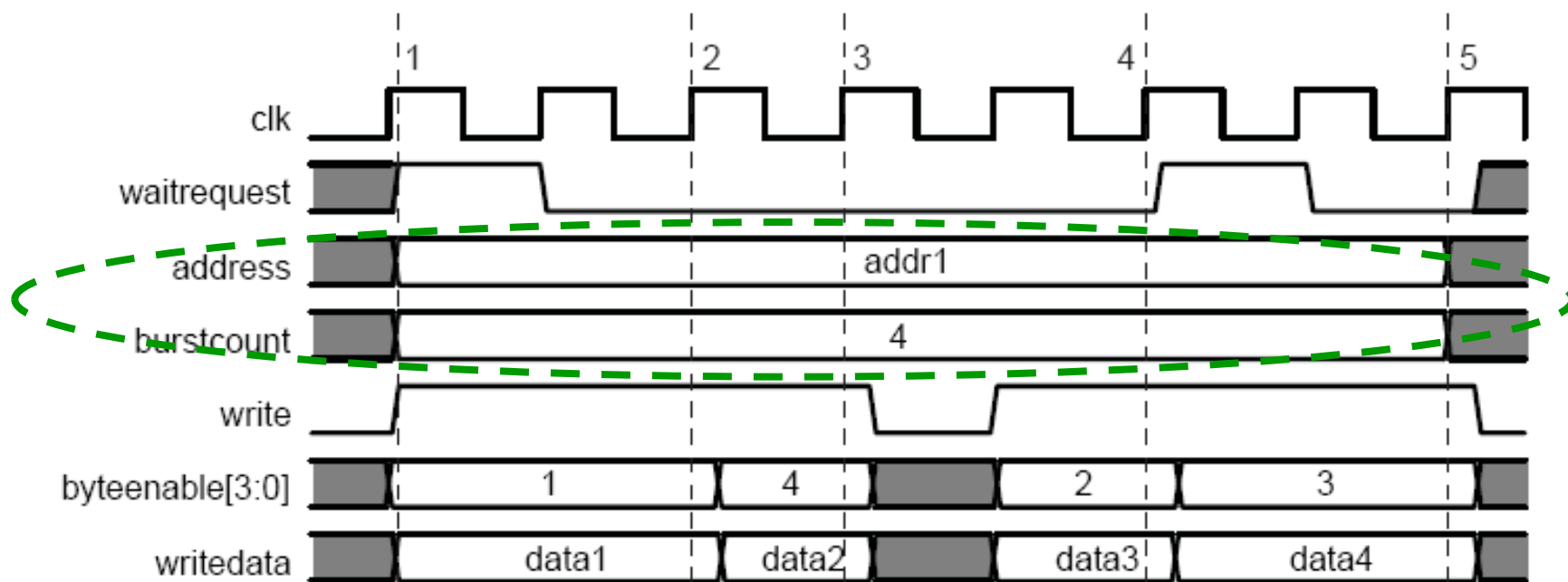


Flush : Kill previous Read data



# Avalon (Master view)

## Burst Write transfers



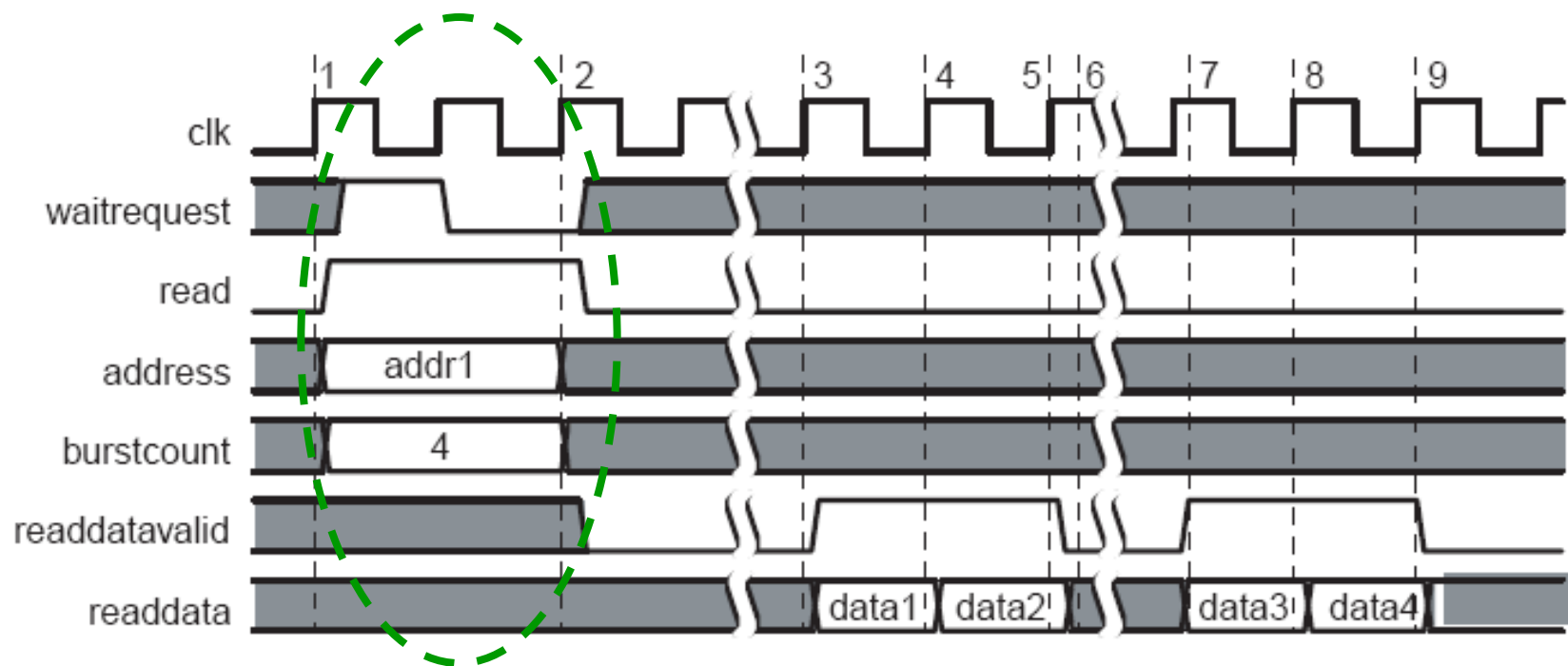
*Address and BurstCount available for the whole transfer*

*Write can be deactivated by the master*

*The number of burstcount needs to be generated*

# Avalon (Master view)

## Burst Read transfer



*Address and BurstCount available for the first cycle only*

*Read signal only for the first cycle*

*The number of burstcount ReadDataValid needs to be generated*

*The master could start a new transfer in 2*

# Bus avalon transfers resume

- Separate :
  - address, data in, data out
- Synchronous on clock's rising edge
- Bus Internal or external wait request
- Transfers with latency available
- Multi-masters
- Arbitration at slave side

# Avalon Address view

- 2 different views of addresses from master and slave, mode of decoding :

- **Memory** (dynamic bus sizing)
- **Register** (native transfers)

← *deprecated*

- Example :

- Master 32 bits data
- Slave 8 bits data

Data bus seen on the Avalon Master side

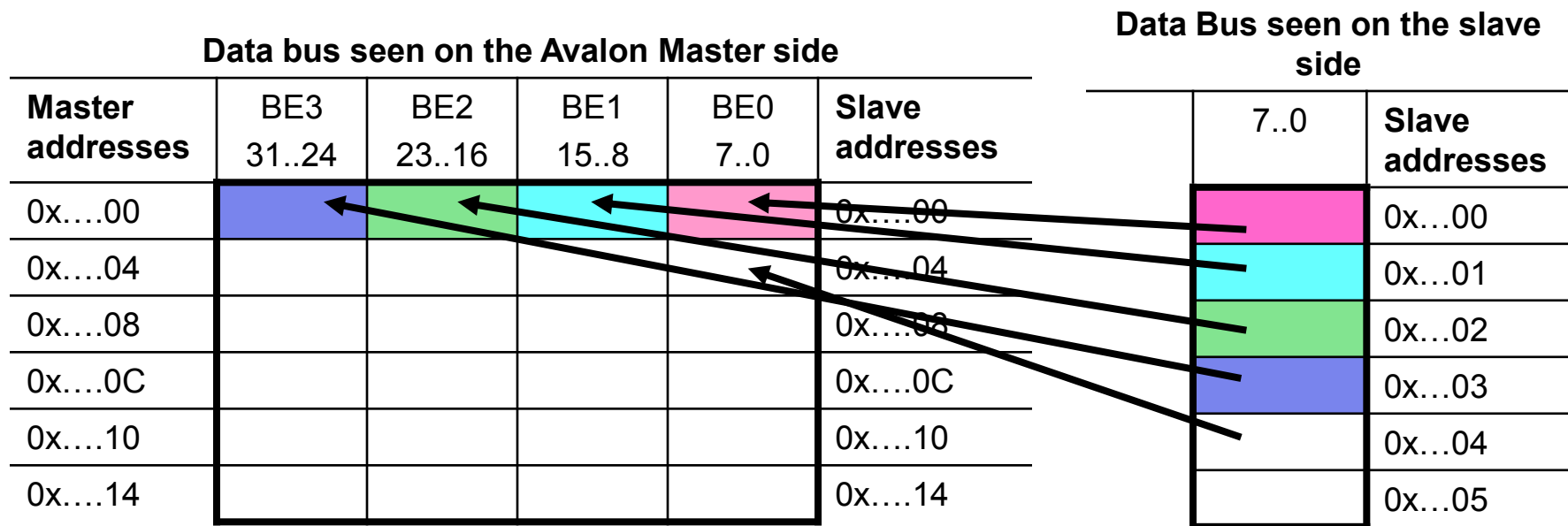
Master addresses	31..24	23..16	15..8	7..0	Slave addresses
0x...00					
0x...04					
0x...08					
0x...0C					
0x...10					
0x...14					

Data Bus seen on the slave side

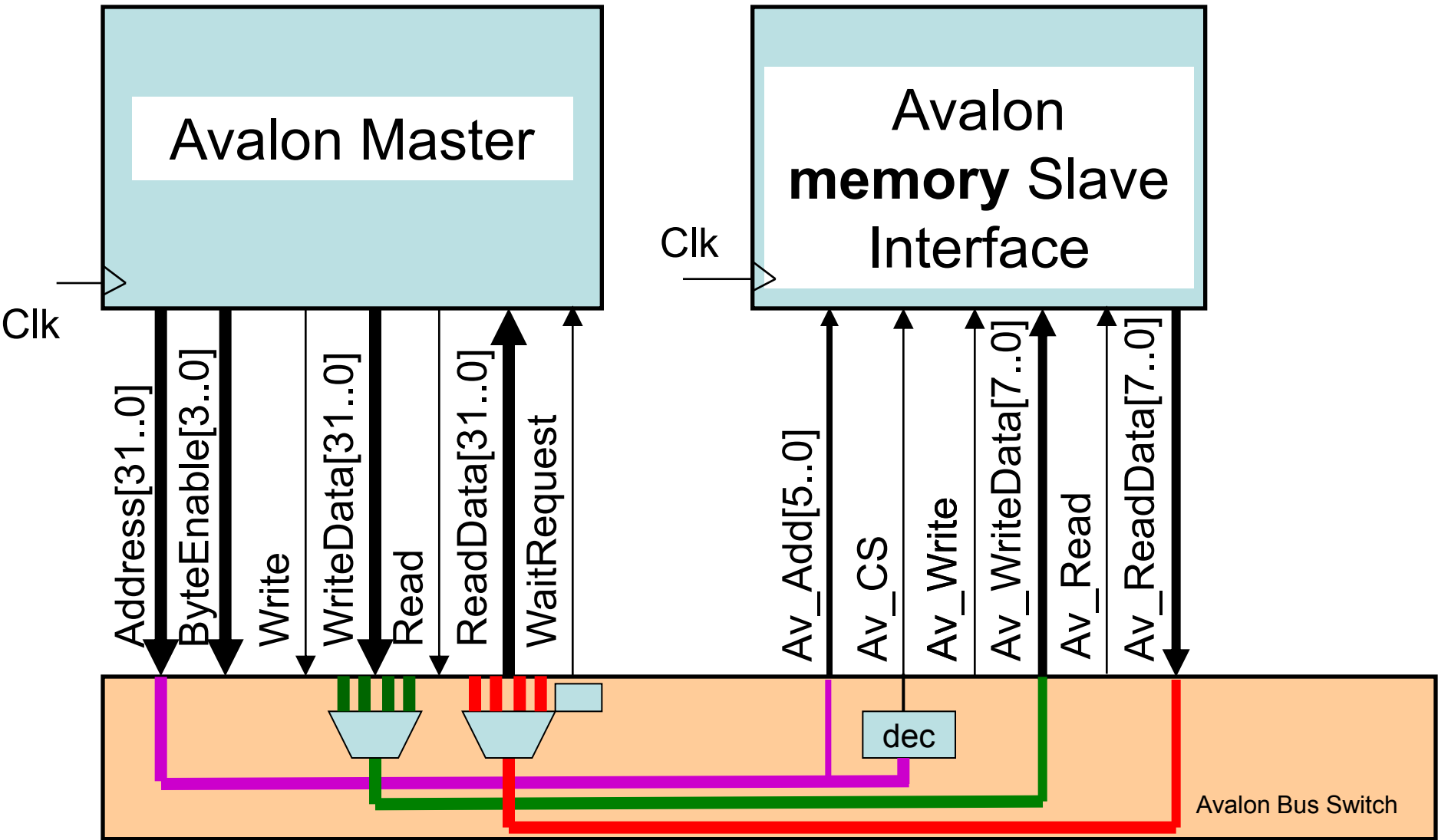
7..0	Slave addresses
	0x...00
	0x...01
	0x...02
	0x...03
	0x...04
	0x...05

# Address view, Memory model

- **Memory model, dynamic bus sizing :**
  - No hole in the master address space
  - Need multiplexers on the data path
  - Master byte address = Slave byte address
  - 1 x 32 bits master transfer → 4 x 8 bits slave access by Avalon switch
- BEx : ByteEnable x

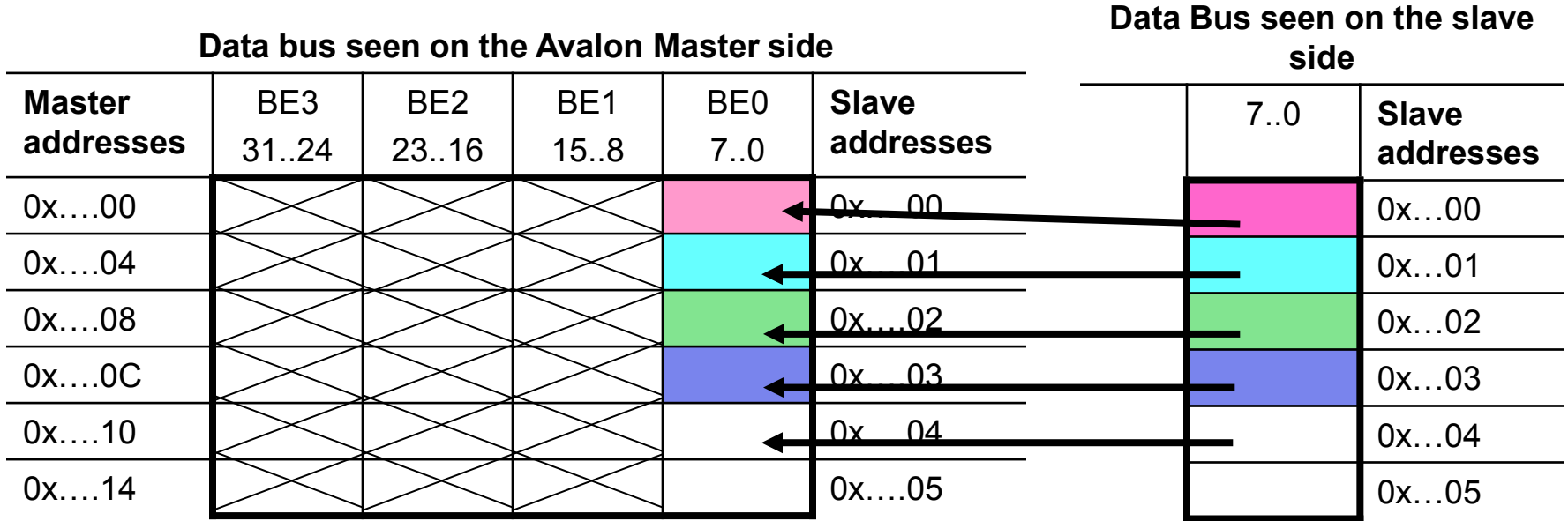


# Memory model for Avalon memory slave



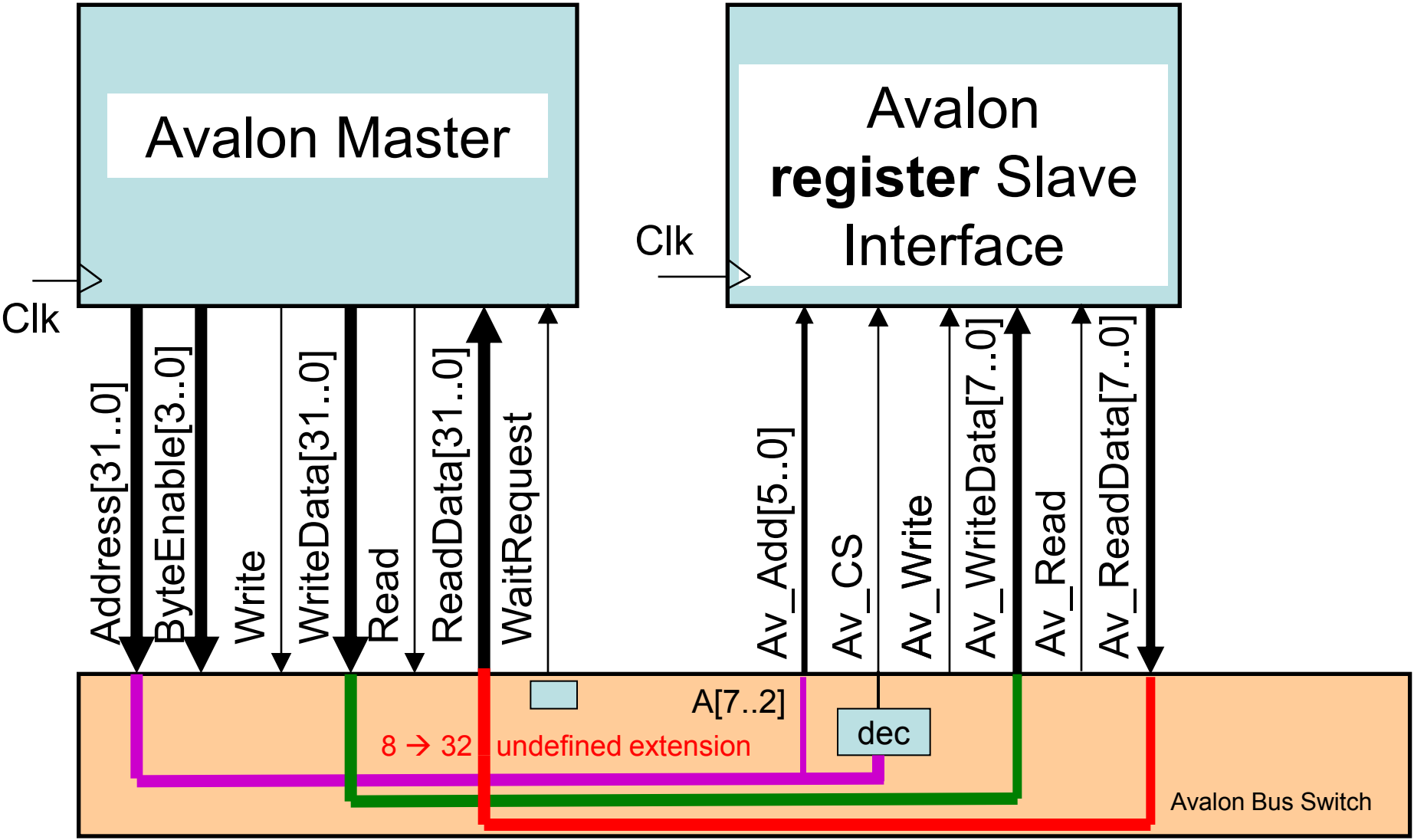
# Address view, Register model (deprecated)

- **Register model, native transfer :**
  - Holes the master address space
  - NO multiplexers needed on the data path to align data
  - Master byte address ≠ Slave byte address
- Access by size of master bus (i.e. 32 bits), 8 bits available, highest bits undefined
- 1 master transfer = 1 slave transfer

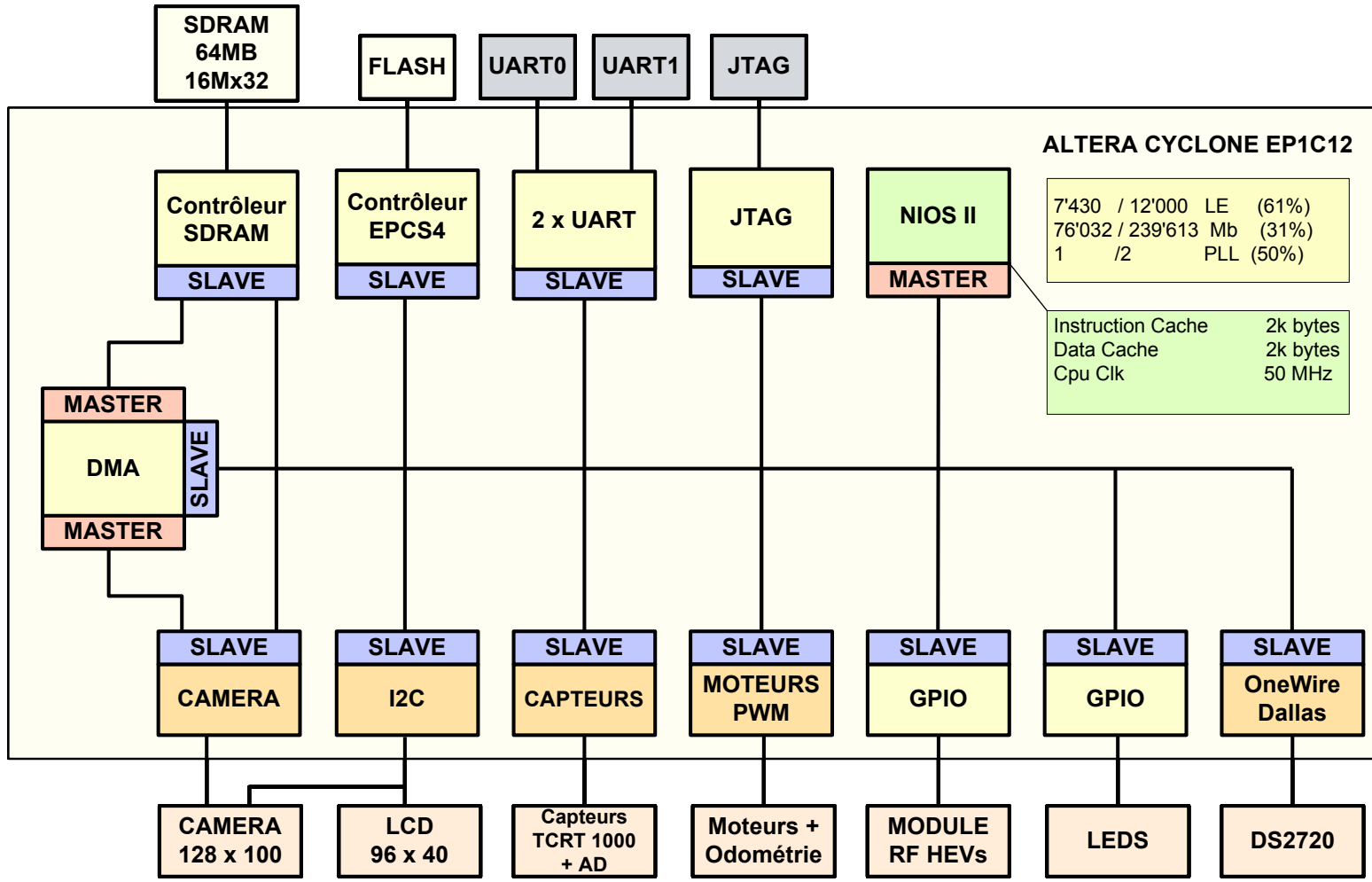




# Memory model for Avalon register slave



# Embedded System on FPGA (example)

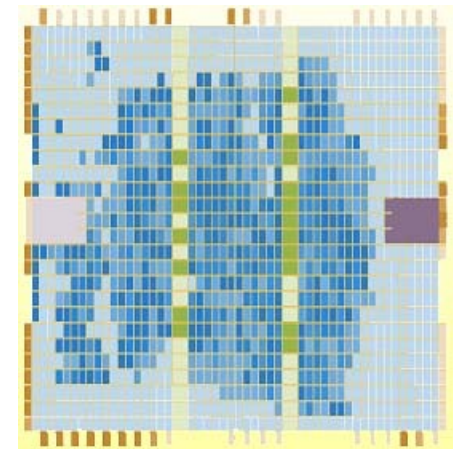
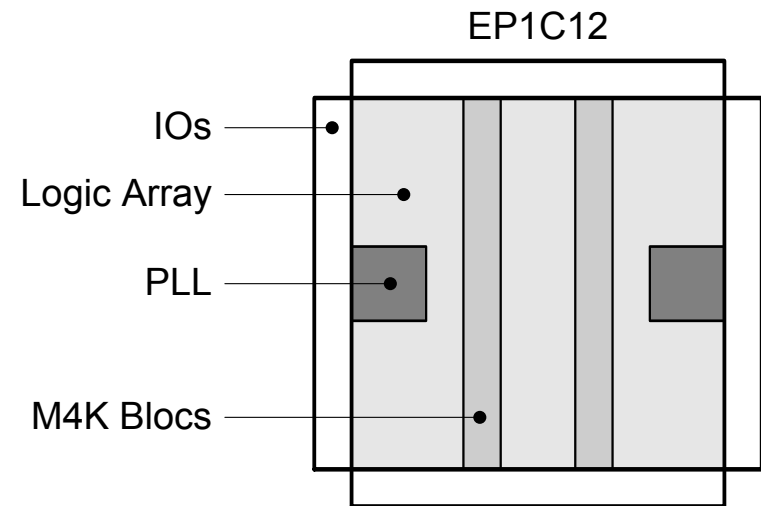


## Architecture of EP1C12

- 12'000 logic Elements (LE)
- 52 x 4 Kbits RAM
- 2 x PLLs
- 180 IOs on 4 bancs
- Proprietary Configuration Bus
- JTAG Port

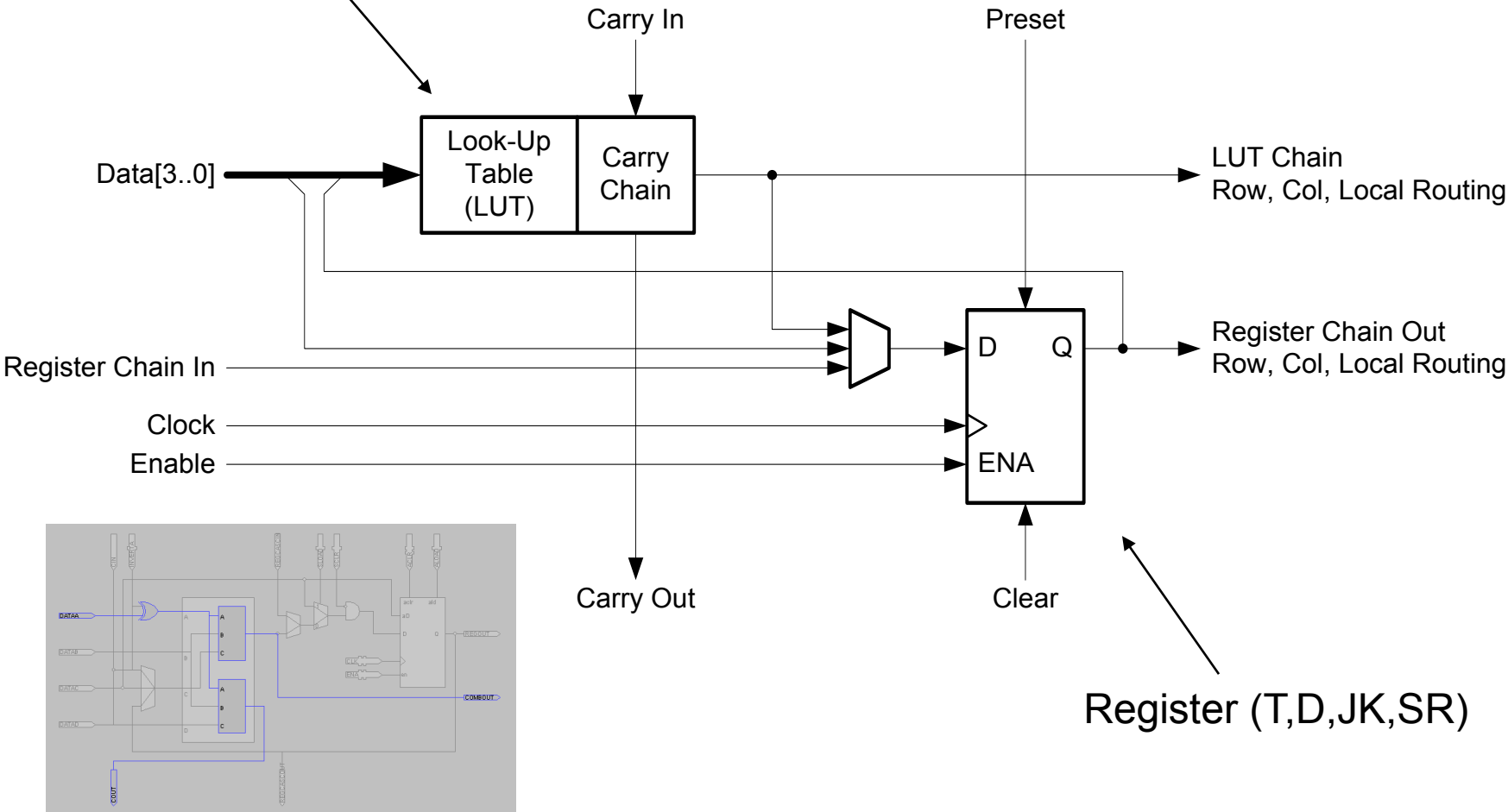
## Quelques limites de fonctionnement

- multiplexor 16→1 :  $f_{\max} \text{ LE} = 275 \text{ MHz}$
- counter 64 bits :  $f_{\max} \text{ LE} = 160 \text{ MHz}$
- memory :  $f_{\max} \text{ M4K} = 220 \text{ MHz}$
- PLL :  $f_{\max} \text{ PLL} = 275 \text{ MHz}$



# Logics Elements (LE)

## Function Generator



Register (T,D,JK,SR)

# Developments Tools from ALTERA

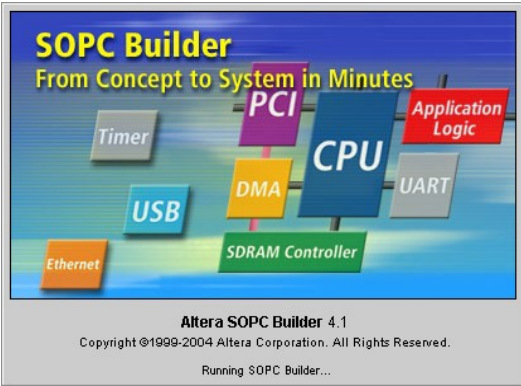
## Quartus II → Hardware Description



- Schematic Editor, VHDL, ...
- Synthesis + placement routing
- Simulation (graphical éditor )
- Signal TAP

## SOPC Builder → SOC NIOS II

2011 → QSys



- Configuration + SOC generation
- Programmable Interface library
- Own Programmable Interfaces.
- Generation SDK

## NIOS II IDE → NIOS II Code

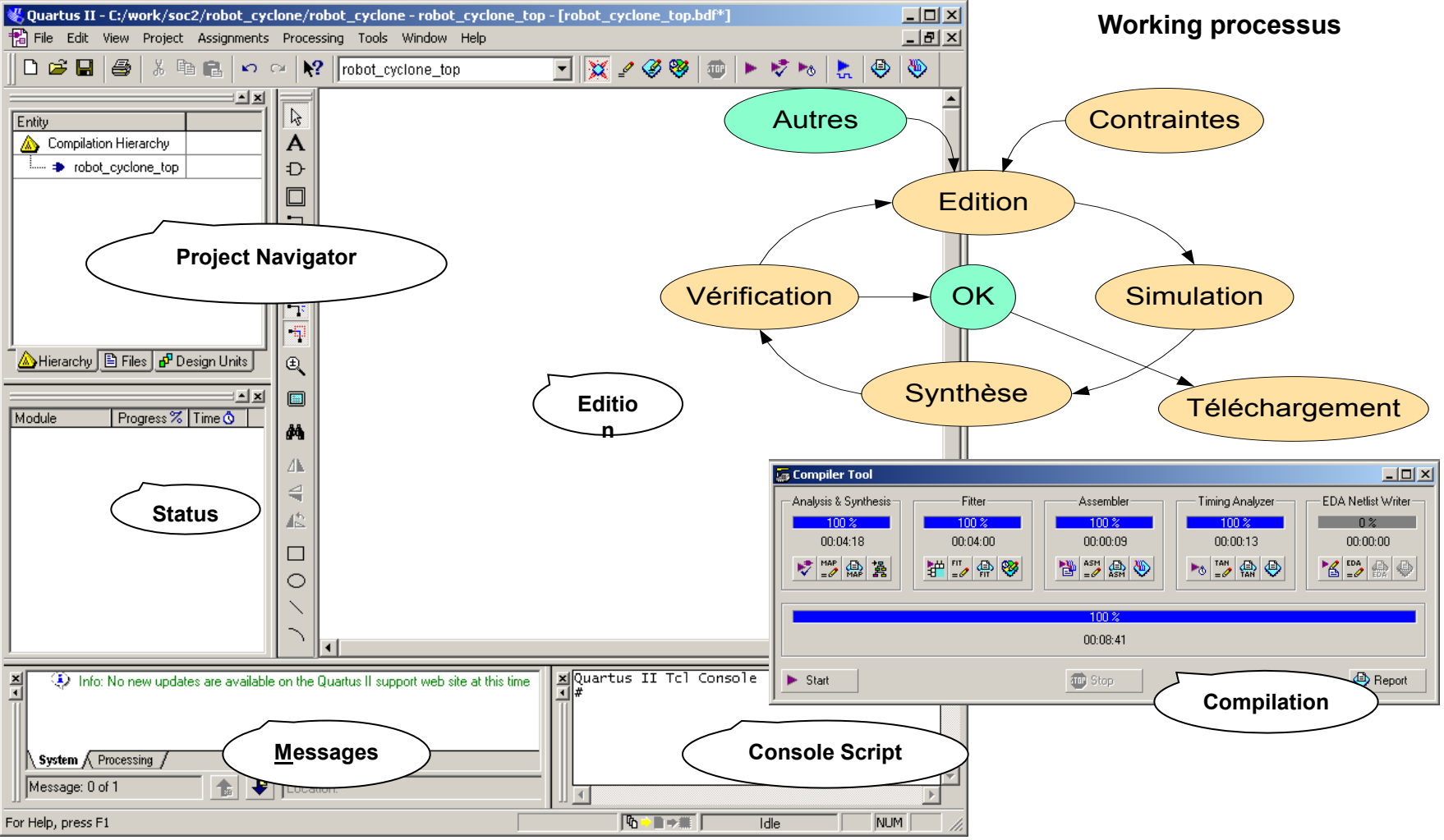
2010 → SBP



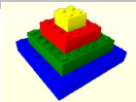
- Project management
- Compiler + Link Editor
- Debugger
- SOC Programmer

# Developments Tools from ALTERA

## Quartus II



# Developments Tools from ALTERA



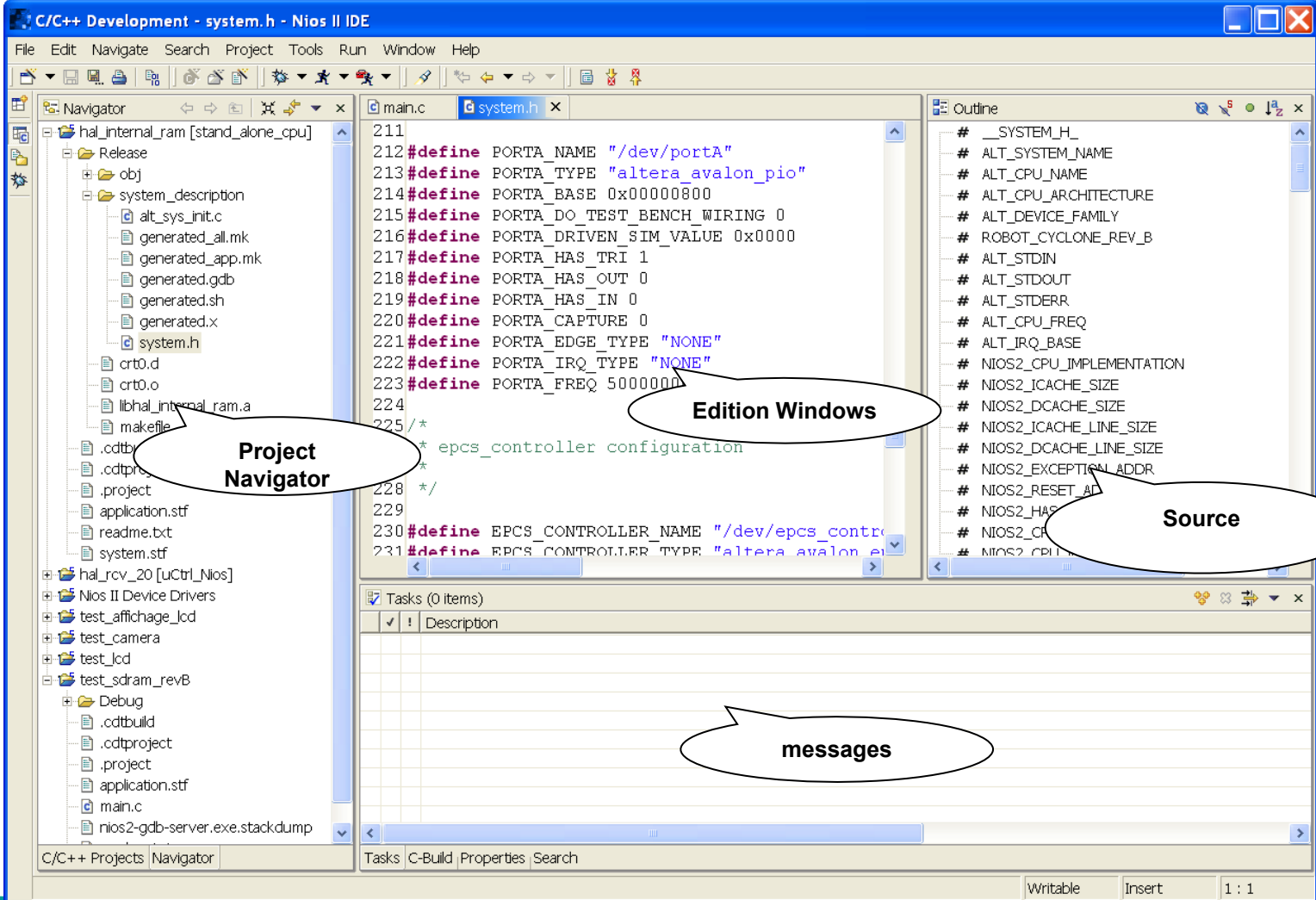
## SOPC Builder (old) → Qsys

The screenshot shows the Altera SOPC Builder interface for a Nios II system. The main window displays a list of modules and their properties. Callouts highlight specific features:

- Processor Nios II:** Points to the 'Nios II Processor - Altera Corporation' module.
- Bus Arbitration:** Points to the 'cpu\_NIOS2F' module.
- Components Library:** Points to the left-hand pane showing various component categories like Bridges, Communication, and Memory.
- SOC:** Points to the central table of modules.
- Memory Map:** Points to the 'Base' and 'End' columns in the table.
- Interrupts:** Points to the 'IRQ' column in the table.

Module Name	Description	Clock	Bus Type	Base	End	IRQ
cpu_NIOS2F	Nios II Processor - Altera Corpora...	clk				
instruction_master	Master port		avalon			
data_master	Master port		avalon			
jtag_debug_module	Slave port		avalon	0x00000000	0x000007FF	IRQ 0 - IRQ 31
dma_0	DMA	clk				
read_master	Master port		avalon			
write_master	Master port		avalon			
control_port_slave	Slave port		avalon	0x00000000	0x000001F	0
epcs_controller	EPCS Serial Flash Controller	clk	avalon	0x00001000	0x000017FF	1
sdram	SDRAM Controller	clk	avalon	0x04000000	0x07FFFFFF	1
jtag_uart	JTAG UART	clk	avalon	0x00002000	0x00002007	2
uart_0	UART (RS-232 serial port)	clk	avalon	0x00002200	0x0000221F	3
portA	PIO (Parallel I/O)	clk	avalon	0x00002400	0x0000240F	1
Isn_i2c_0	Isn_i2c	clk	avalon	0x00003000	0x0000300F	4
cam_port_0	cam_port	clk	avalon			

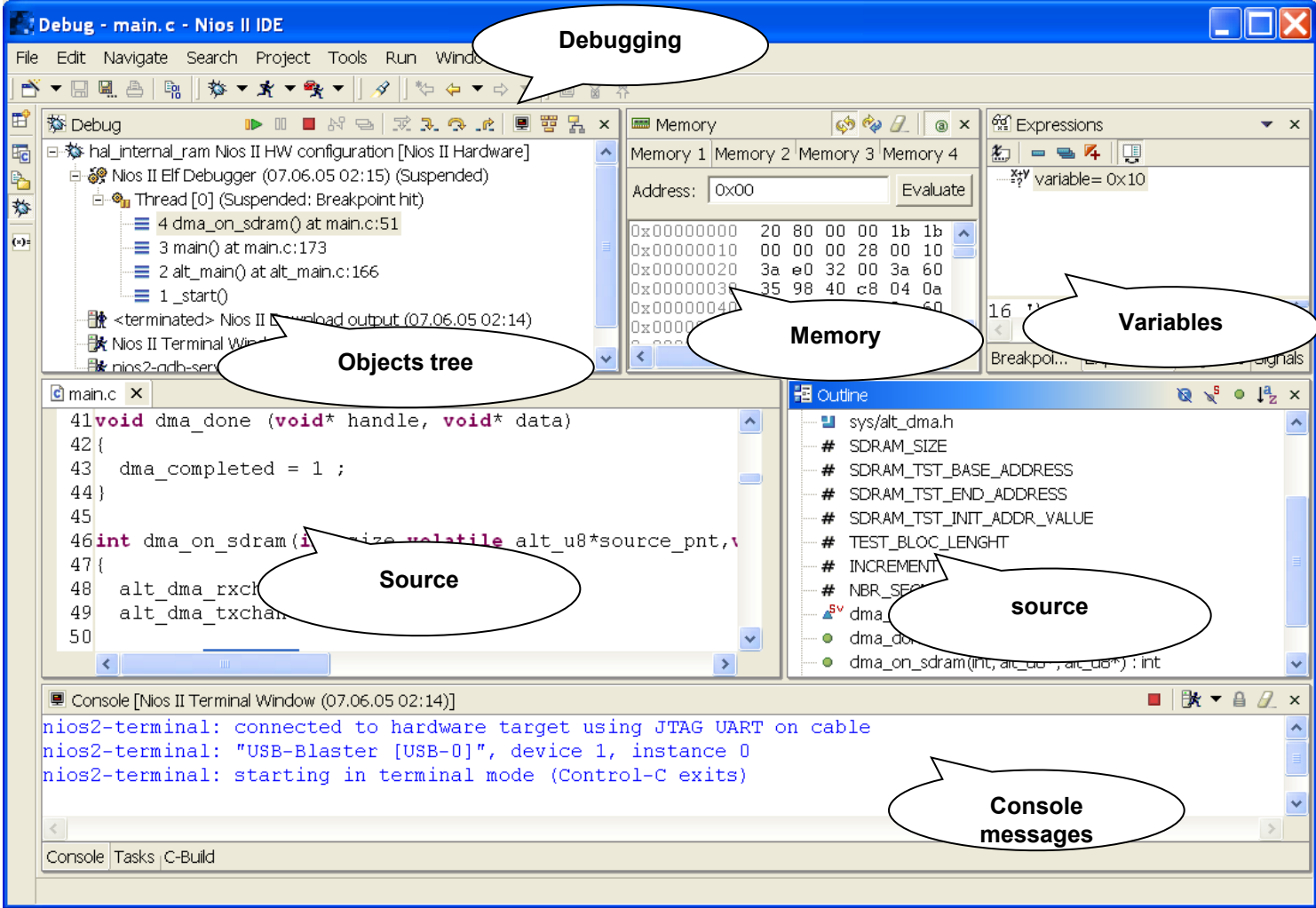
## NIOS II IDE (development)





# Developments Tools from ALTERA

## NIOS II IDE (debugger)



# Conclusion

## Some positives points of a softcore architecture

- Fast implementation**
- Modular Architecture**
- Simplicity**
- Good documentation**
- Nice for teaching complex integrated embedded systems**
- Ease of development of our own programmable interface on internal bus (i.e. Avalon in VHDL, Verilog)**
- Full system on FPGA, easily adaptable**
- Operating System included (uC/OS II)**

## Some negate points

- Quite big tools to develop a system**
- Thus tools to learn**