Design of Embedded Hardware and Firmware

Introduction on
"System On Programmable Chip"
NIOS II – Avalon Bus

Andres Upegui, René Beuchat
Laboratoire de Systèmes Numériques
hepia/HES-SO

Embedded system on Altera FPGA

Goals :

- To understand the architecture of an embedded system on FPGA
- To be able to identify the components required for building a specific interface
- To have an overview of the construction of a full system based on a standard softcore bus in a FPGA with block modules

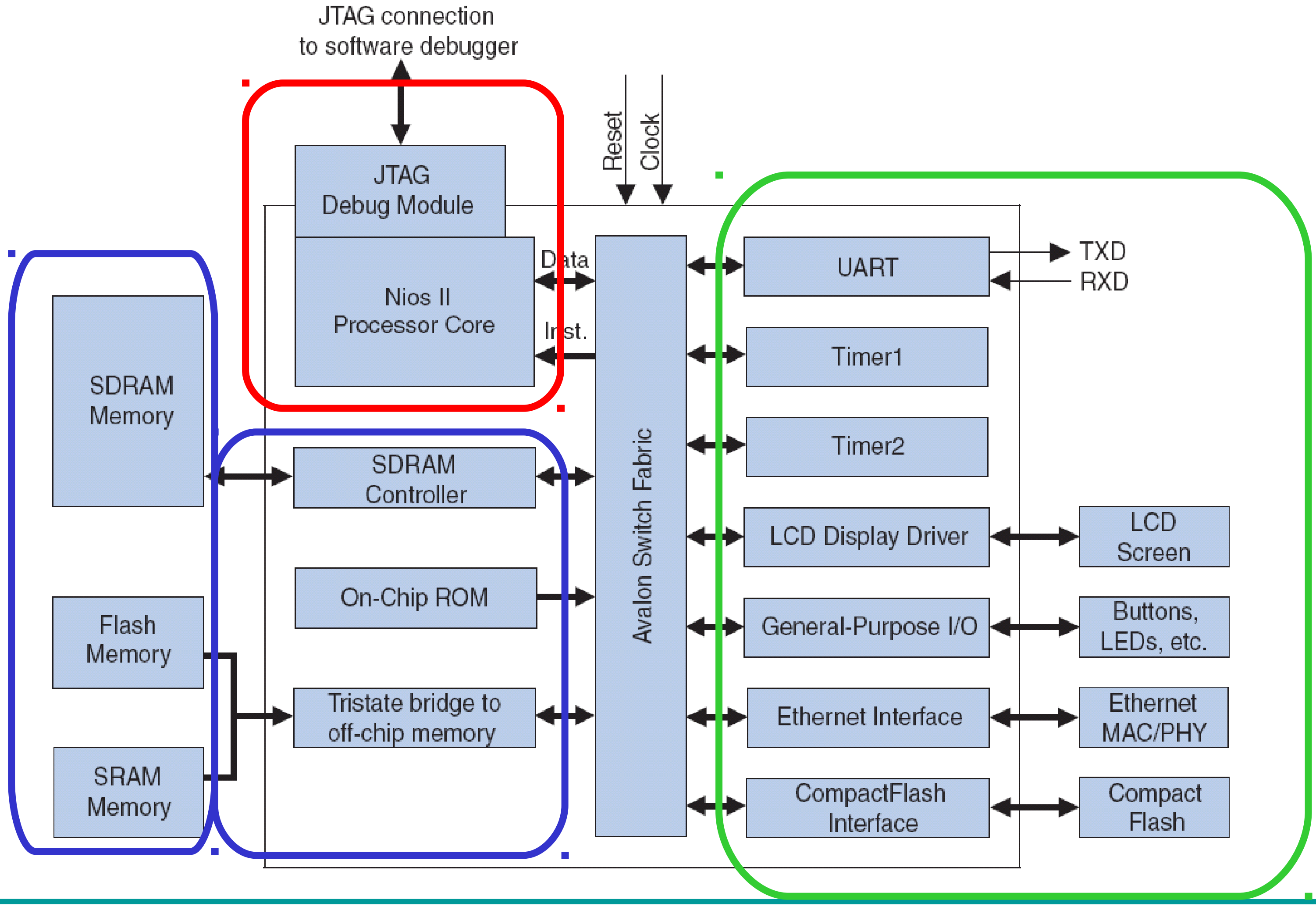
Contents

- System On FPGA
- NIOS II a softcore processor
- Avalon Bus
- Design of a specific slave programmable interface on Avalon

NIOS II –

Embedded system NIOSII/Avalon Architecture

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


JTAG connection
to software debugger
JTAG
Debug Module
Nios II
Processor Core
Data
Inst.
Reset
Clock
SDRAM
Memory
SDRAM
Controller
On-Chip ROM
Flash
Memory
Tristate bridge to
off-chip memory
SRAM
Memory
Avalon Switch Fabric
UART
TXD
RXD
Timer1
Timer2
LCD Display Driver
LCD
Screen
General-Purpose I/O
Buttons,
LEDs, etc.
Ethernet Interface
Ethernet
MAC/PHY
CompactFlash
Interface
Compact
Flash

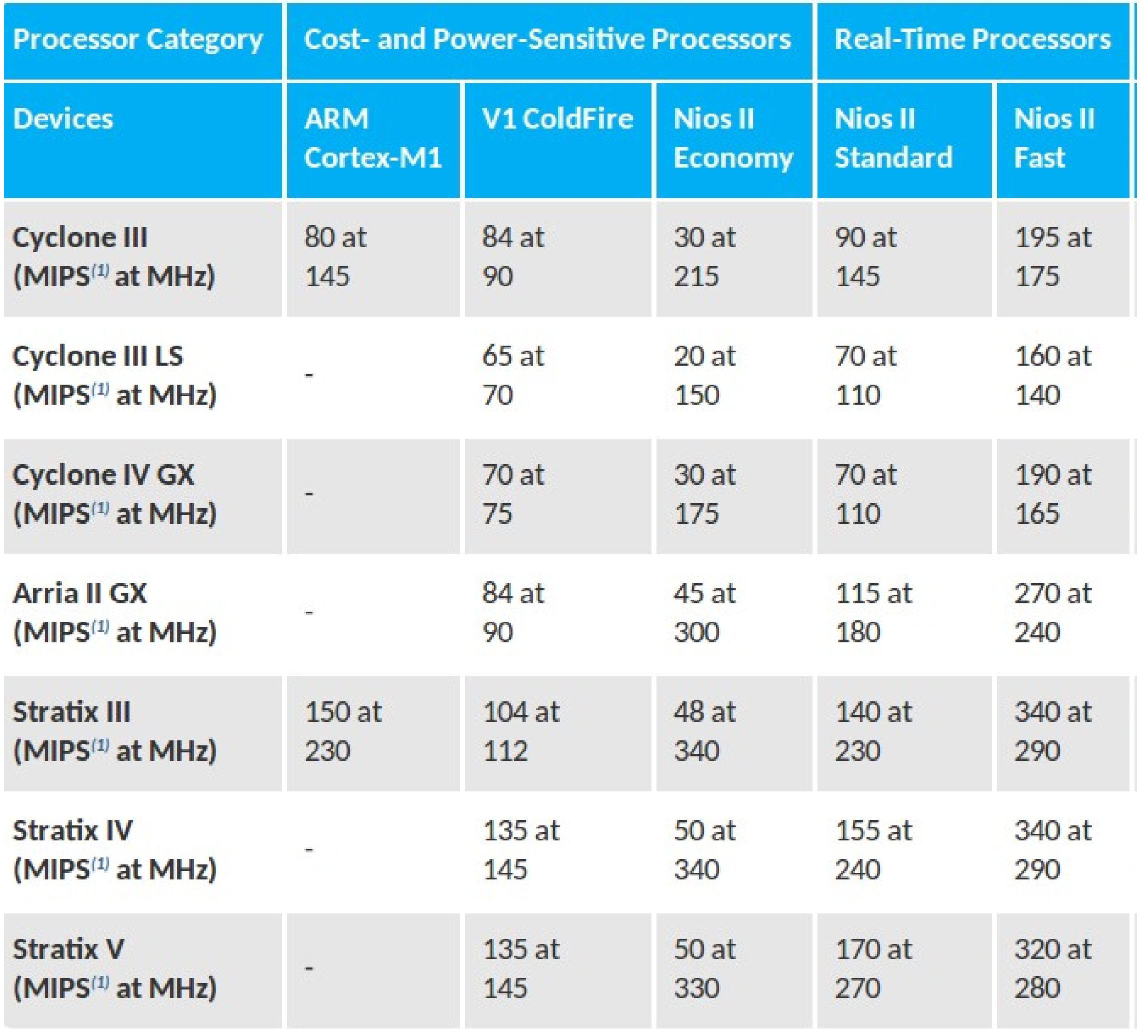
NIOS II

- 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 langage(VHDL/Verilog/…)
- 32 bits Architecture
- 256 instructions available for user implementation
- 3 versions
- Operating systems supported: linux, eCos, embOS, Euros RTOS, oSCAN, uCLinux, ThreadX, VxWorks

NIOS II Processor :
3 basic configurable architectures

	Nios II / f Fast	Nios II / s Standard	Nios II / e Economy
Pipeline	6 stages	5 stages	None
Branch prediction	Dynamic	Static	None
Instruction Cache	Configurable	Configurable	None
Data Cache	Configurable	None	None
MMU	Yes	No	No
Hard Multiplier	1 cycle	3 cycles	None
Hard divider	Yes	Yes	No
Resources	Env 2500 LEs	< 1400 LEs	< 700 LEs
Max freq	290 MHz	270 MHz	340 MHz

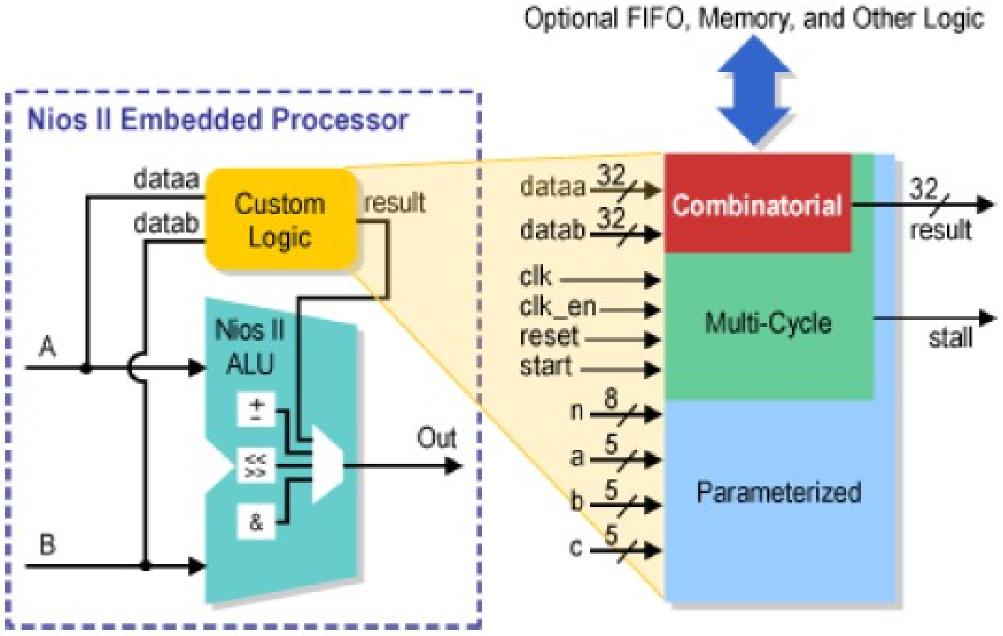
NIOS II Processor performance



Processor Category	Cost- and Power-Sensitive Processors			Real-Time Processors	
Devices	**ARM Cortex-M1**	**V1 ColdFire**	**Nios II Economy**	**Nios II Standard**	**Nios II Fast**
Cyclone III (MIPS(1) at MHz)	80 at 145	84 at 90	30 at 215	90 at 145	195 at 175
Cyclone III LS (MIPS(1) at MHz)	-	65 at 70	20 at 150	70 at 110	160 at 140
Cyclone IV GX (MIPS(1) at MHz)	-	70 at 75	30 at 175	70 at 110	190 at 165
Arria II GX (MIPS(1) at MHz)	-	84 at 90	45 at 300	115 at 180	270 at 240
Stratix III (MIPS(1) at MHz)	150 at 230	104 at 112	48 at 340	140 at 230	340 at 290
Stratix IV (MIPS(1) at MHz)	-	135 at 145	50 at 340	155 at 240	340 at 290
Stratix V (MIPS(1) at MHz)	-	135 at 145	50 at 330	170 at 270	320 at 280

Accelerating computation : Custom user instructions

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


Optional FIFO, Memory, and Other Logic
Nios II Embedded Processor
dataa
datab
Custom Logic
result
A
B
Nios II ALU
+ -
<<
>>
&
Out
dataa 32
datab 32
clk
clk_en
reset
start
n 8
a 5
b 5
c 5
Combinatorial
Multi-Cycle
Parameterized
32
result
stall

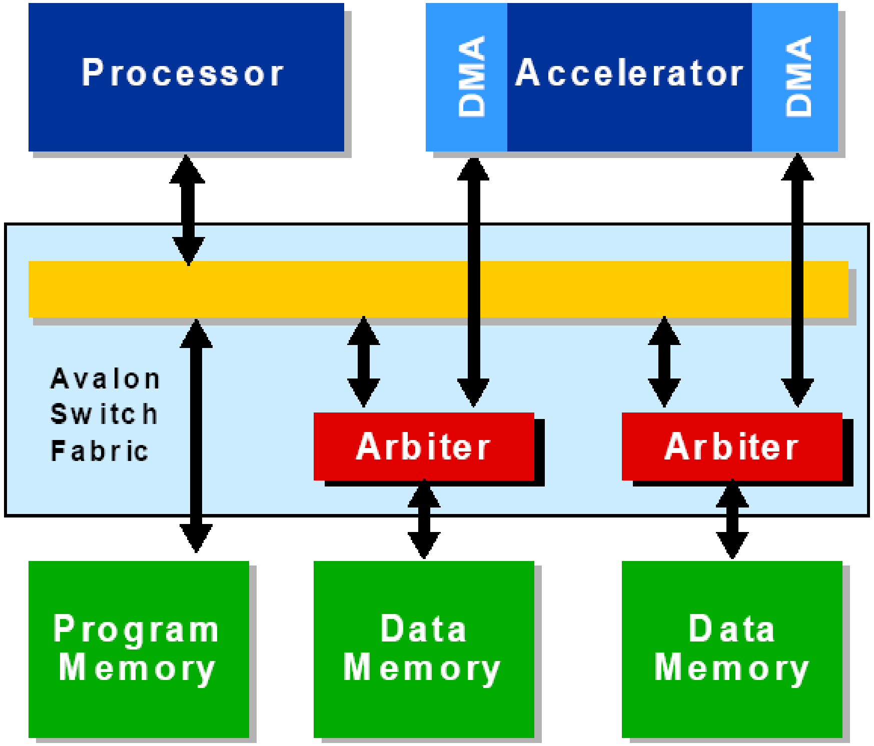
Instructions can be:

- Combinatorial
- Multi-cycle, synchronized by *clk* and *stall*
- Parameterized

They can have access to all the FPGA resources !!!

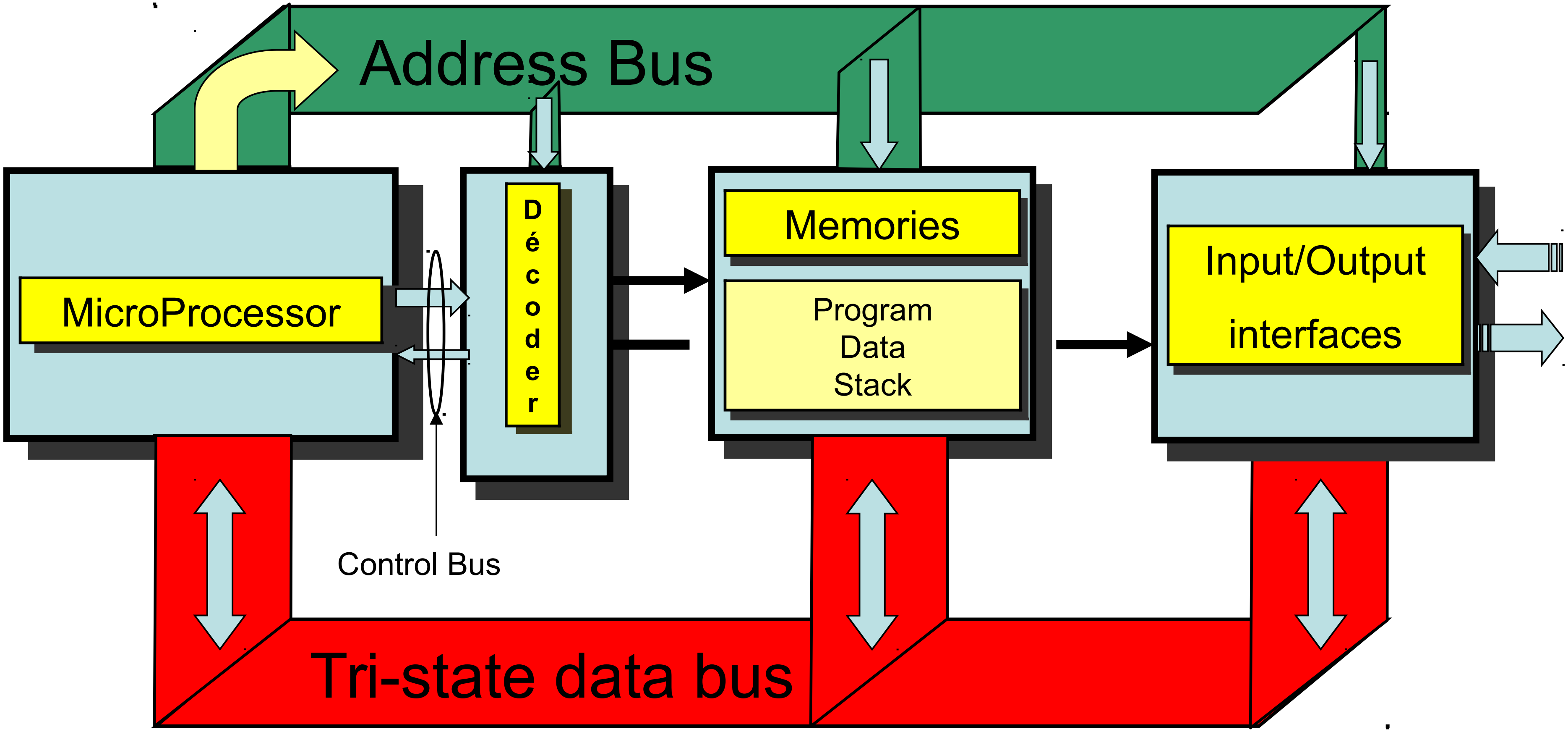
Improving performance : hardware accelerator

- 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


Processor
DMA
Accelerator
DMA
Avalon Switch Fabric
Arbiter
Arbiter
Program Memory
Data Memory
Data Memory

Classic Computer architecture (general and simplified)

In single master, address bus is totem-pole, data bus is tri-state


Address Bus
MicroProcessor
Décoder
Memories
Program Data Stack
Input/Output interfaces
Control Bus
Tri-state data bus

Classic Computer architecture (general and simplified)

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

SoPC architecture : Avalon bus

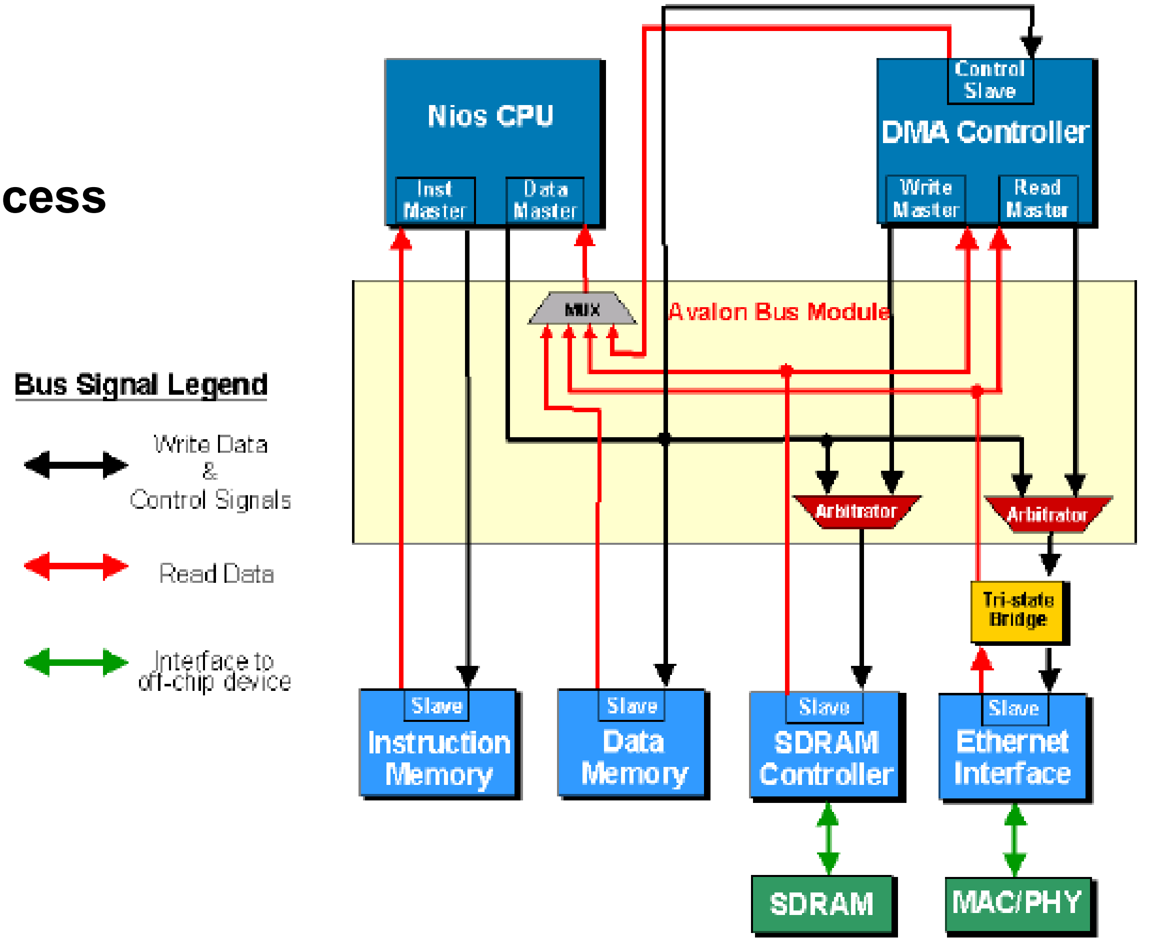
Multi-Master

Arbitrage « slave-side »

Concurrent Master-Slave Access

Synchronous transfers

. *Avalon Bus Module Block Diagram - An Example System*


Nios CPU
Inst Master
Data Master
Control Slave
DMA Controller
Write Master
Read Master
MUX
Avalon Bus Module
Arbitrator
Arbitrator
Tri-state Bridge
Slave
Instruction Memory
Slave
Data Memory
Slave
SDRAM Controller
Slave
Ethernet Interface
SDRAM
MAC/PHY
Bus Signal Legend
Write Data & Control Signals
Read Data
Interface to off-chip device

SoPC architecture

- 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
- Master/Slave
- Synchronous bus
- Wait state by configuration or dynamic
- Hold / Set up available

Avalon
« slave » signals

Signal Type	**Width**	**Direction**	**Required**	**Description**
clk	1	In	(No)	Global clk for system module and Avalon bus modules. **All transactions synchronous to clk rising edge**
nReset	1	In	No	Global Reset of the system
address	1..32	In	No	Address for Avalon bus modules
chipselect	1	In	**Yes**	Selection of the Avalon bus module
read	1	In	No	Read request to the slave
readdata	1..32	Out	No	Read data from the slave module
write	1	In	No	Write request to the slave
writedata	1..32	In	No	Data from Master to Slave module
irq	1	Out	No	Interrupt request to the master

Avalon « slave » signals

- The **ChipSelect** is generated by the Avalon bus and selects the module
- The **Address**[n .. 0] is used to access a specific register/memory position in the selected module. Only the minimum number of address is necessary.
- *The **Read** and **Write** signals specified the direction of the transfers.* They are provided by a Master and received by the slave modules
- **ReadData**(..) and **WriteData**(..) bus transfers the Datas from (read)/ to (write) the Slaves

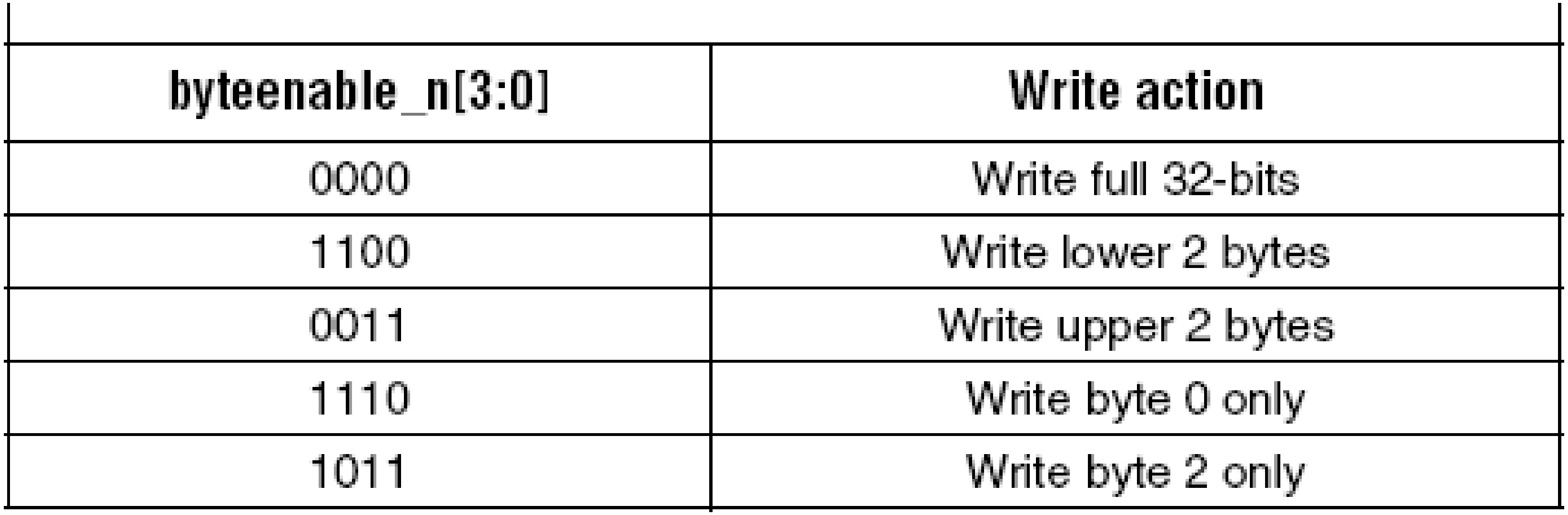
Avalon
« slave » signals

- The **Read** and **Write** signals specified the direction of the transfers.
- 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 Datas from (read)/ to (write) the Slaves

Avalon « slave » signals

- BE (**Byte Enable**) signals indicate the bytes to transfer.
  - The number of BE activated are always 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 bus make the translation and performs multiple accesses if required.

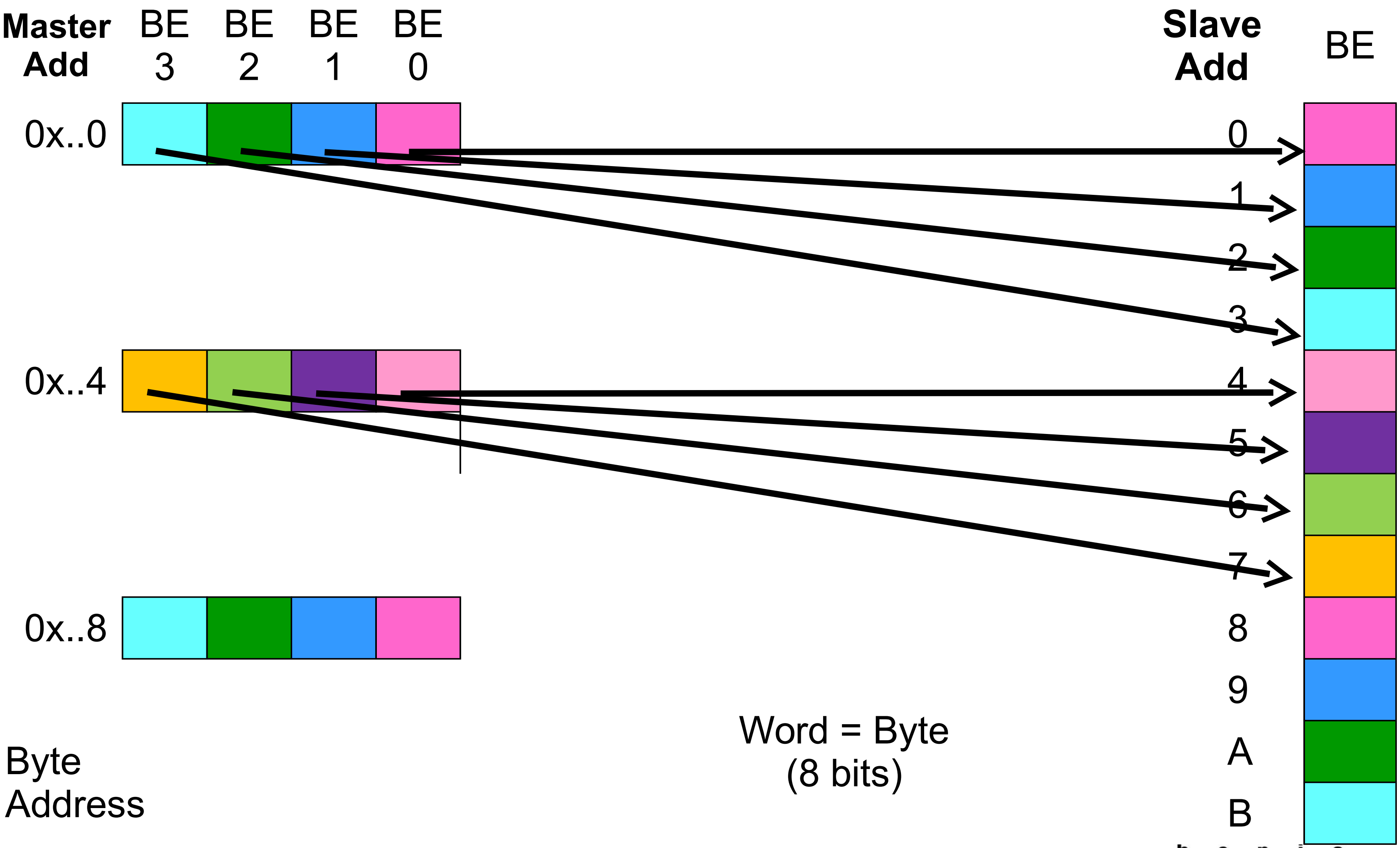
Avalon
byte enable



byteenable_n[3:0]	Write action
0000	Write full 32-bits
1100	Write lower 2 bytes
0011	Write upper 2 bytes
1110	Write byte 0 only
1011	Write byte 2 only

Specify bytes to be transferred
Active low signals in this representation

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



Master Add

BE 3 | BE 2 | BE 1 | BE 0

0x..0

0x..4

0x..8

Slave Add

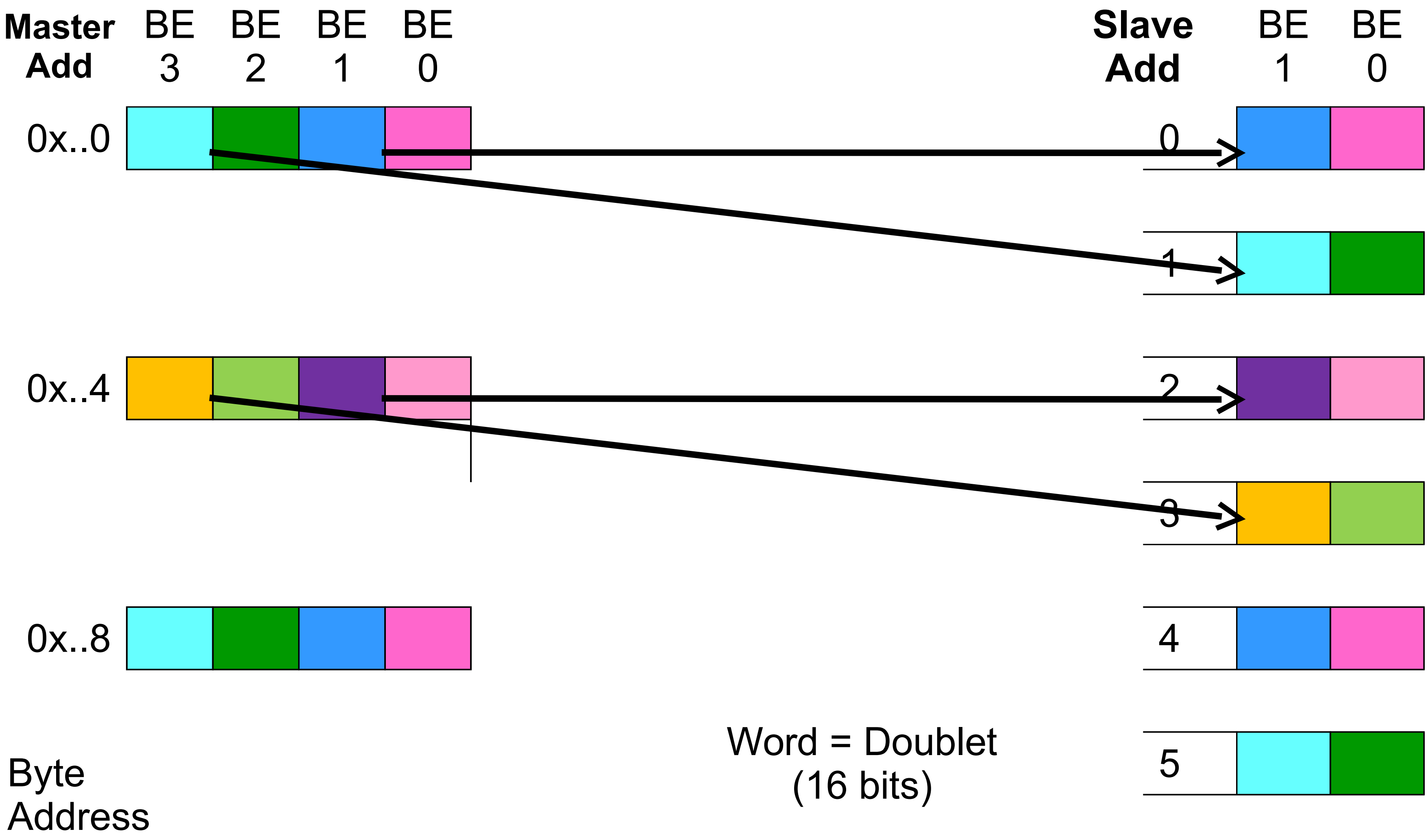
BE

0
1
2
3
4
5
6
7
8
9
A
B

Word = Byte
(8 bits)

Byte
Address

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



Master Add BE 3 BE 2 BE 1 BE 0

Slave Add BE 1 BE 0

0x..0

0x..4

0x..8

0

1

2

3

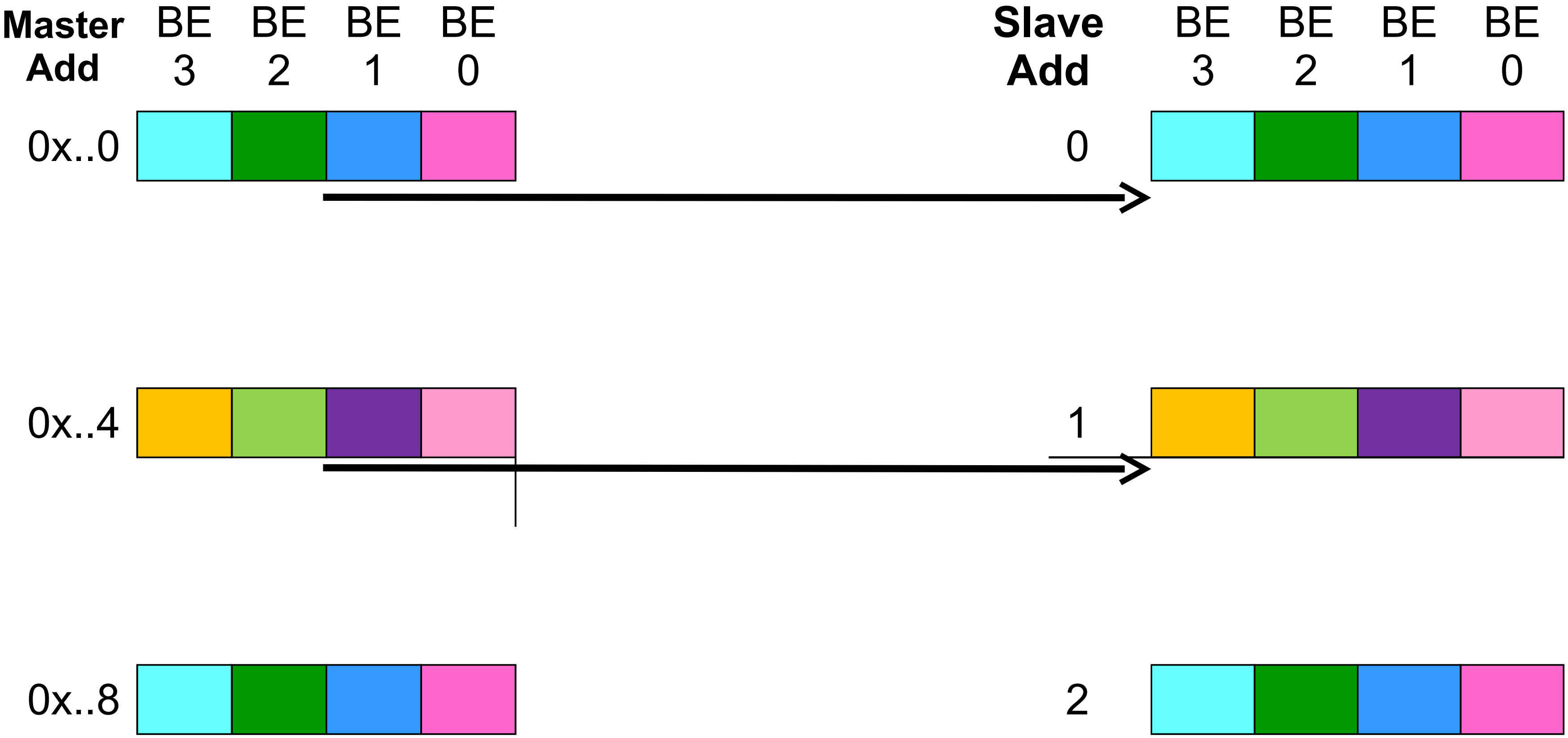
4

5

Byte Address

Word = Doublet (16 bits)

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



Master Add BE 3 BE 2 BE 1 BE 0

Slave Add BE 3 BE 2 BE 1 BE 0

0x..0 → 0

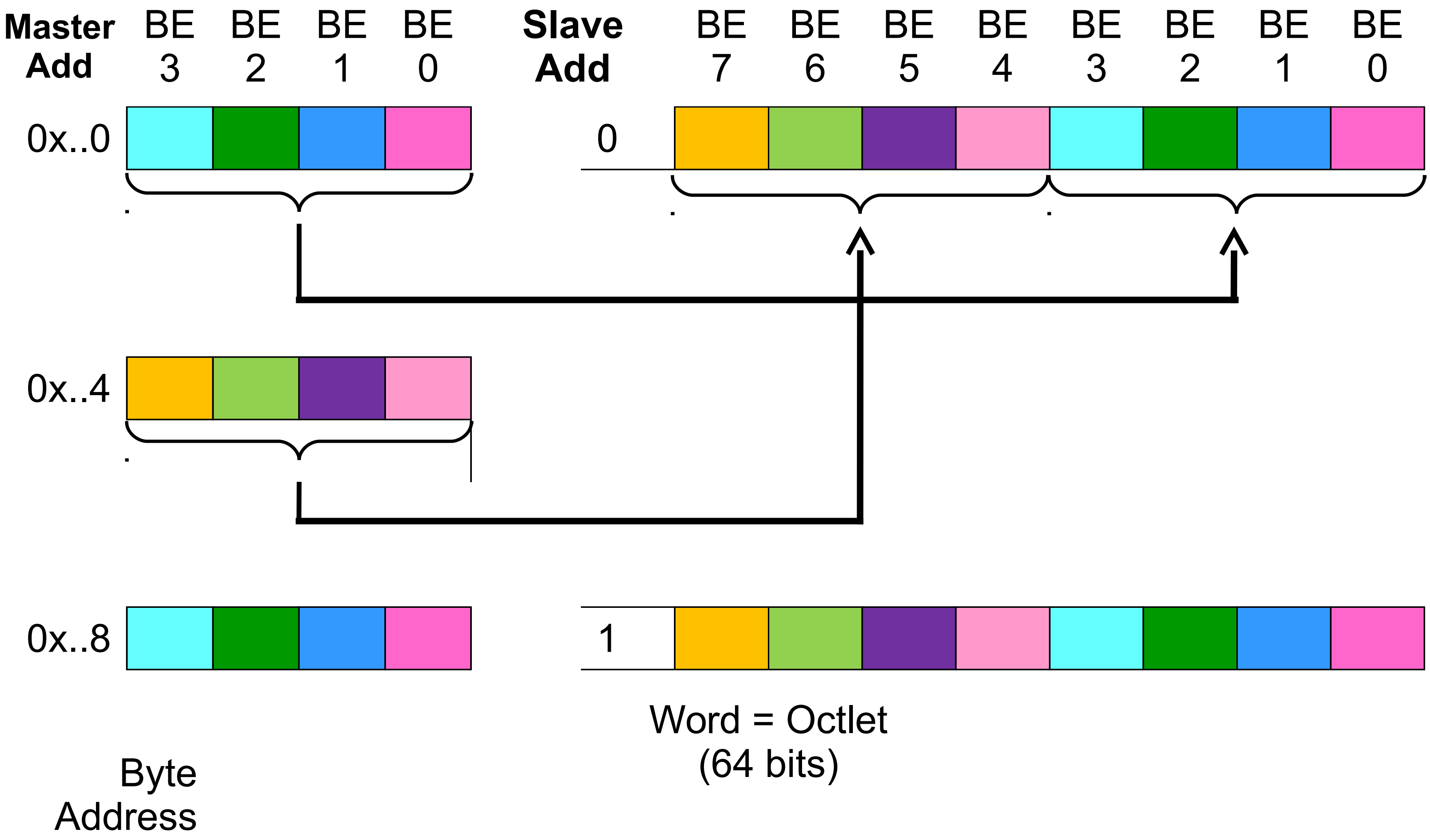
0x..4 → 1

0x..8 2

Byte Address

Word = Quadlet (32 bits)

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


Master Add
BE 3
BE 2
BE 1
BE 0
Slave Add
BE 7
BE 6
BE 5
BE 4
BE 3
BE 2
BE 1
BE 0
0x..0
0
0x..4
0x..8
1
Word = Octlet (64 bits)
Byte Address

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 **variable latency**, 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 Bus

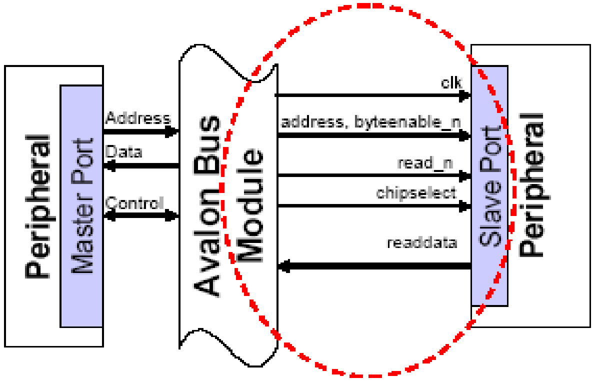
Slave view of transfers

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

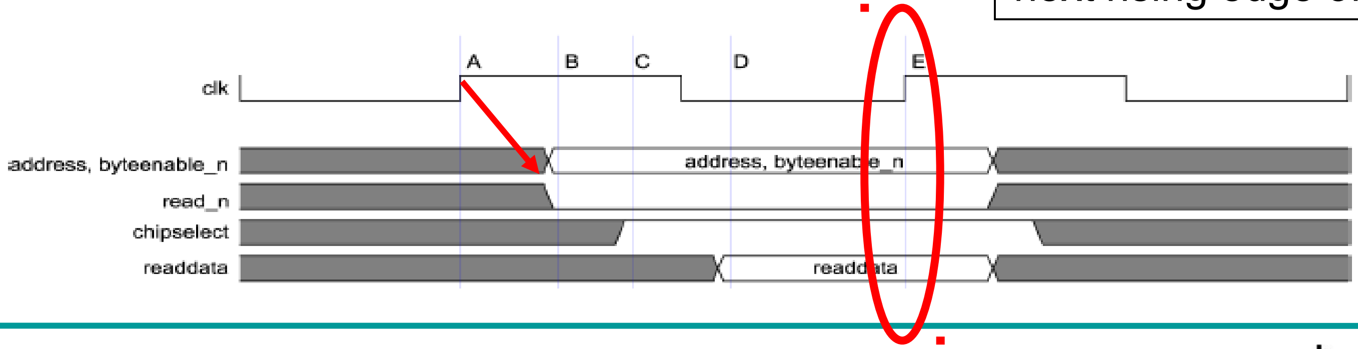
Avalon slave

read, 0 wait, asynchronous peripheral

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

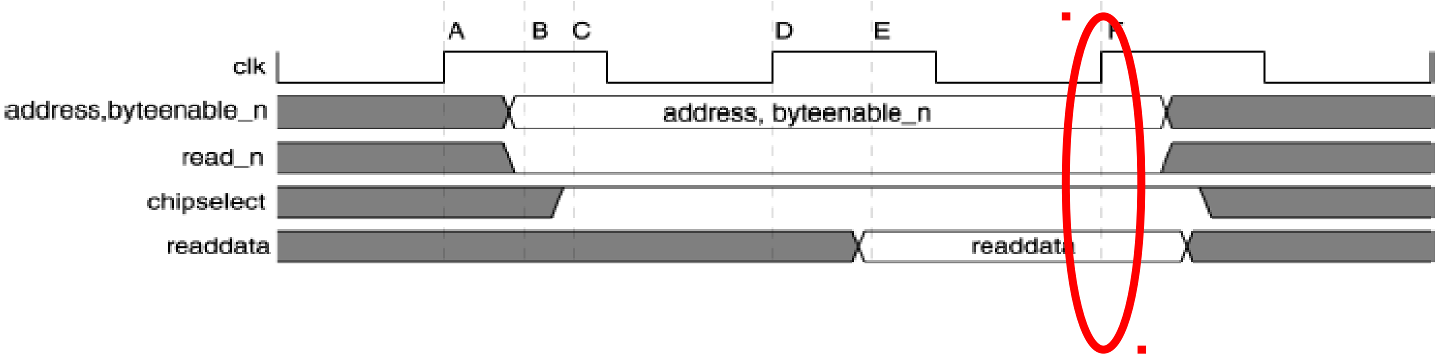

Peripheral
Master Port
Address
Data
Control
Avalon Bus Module
clk
address, byteenable_n
read_n
chipselect
readdata
Slave Port
Peripheral

ReadData available at next rising edge of clk (E)

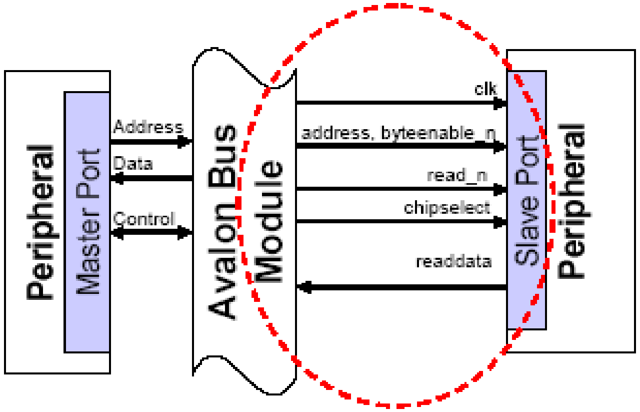

A
B
C
D
E
clk
address, byteenable_n
read_n
chipselect
readdata
address, byteenable_n
readdata

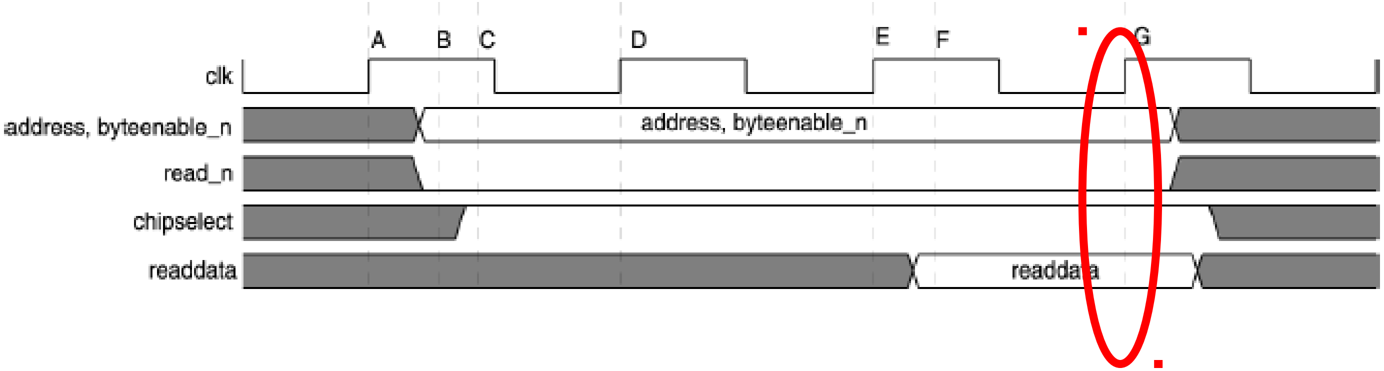
Avalon slave read, 1 wait

Wait cycle specified by design

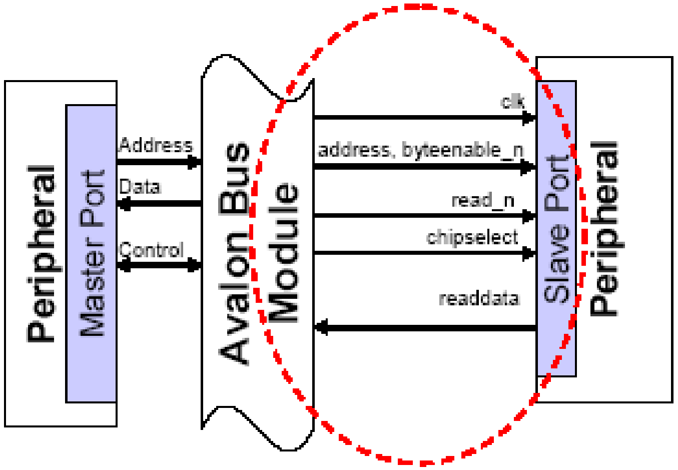

A
B
C
D
E
F
clk
address,byteenable_n
read_n
chipselect
readdata
address, byteenable_n
readdata

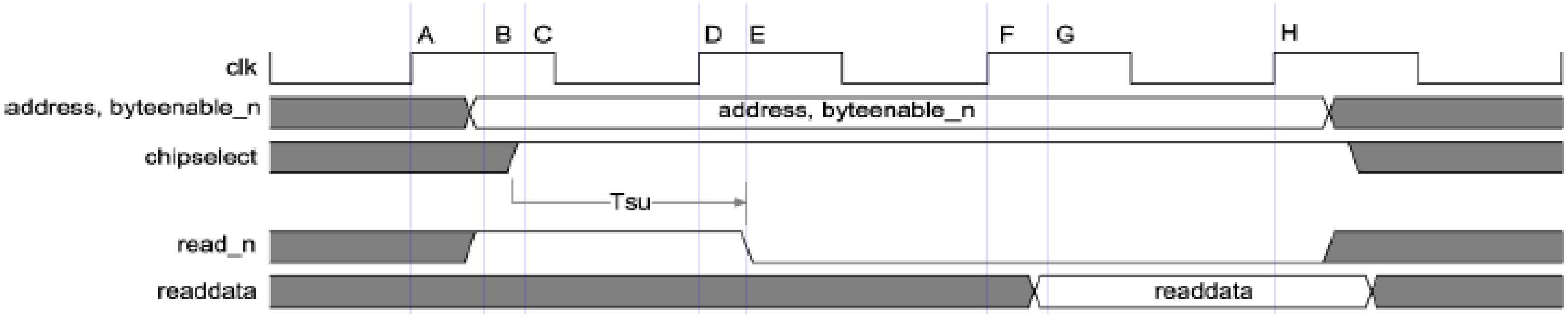
Avalon read slave, 2 wait


Peripheral
Master Port
Address
Data
Control
Avalon Bus Module
clk
address, byteenable_n
read_n
chipselect
readdata
Slave Port
Peripheral


A
B
C
D
E
F
G
clk
address, byteenable_n
read_n
chipselect
readdata
address, byteenable_n
readdata

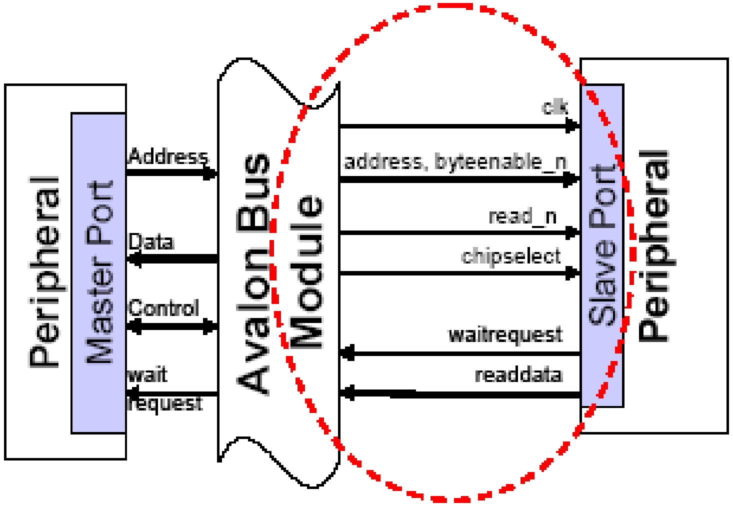
Avalon read
slave, 1 set up and 1 wait

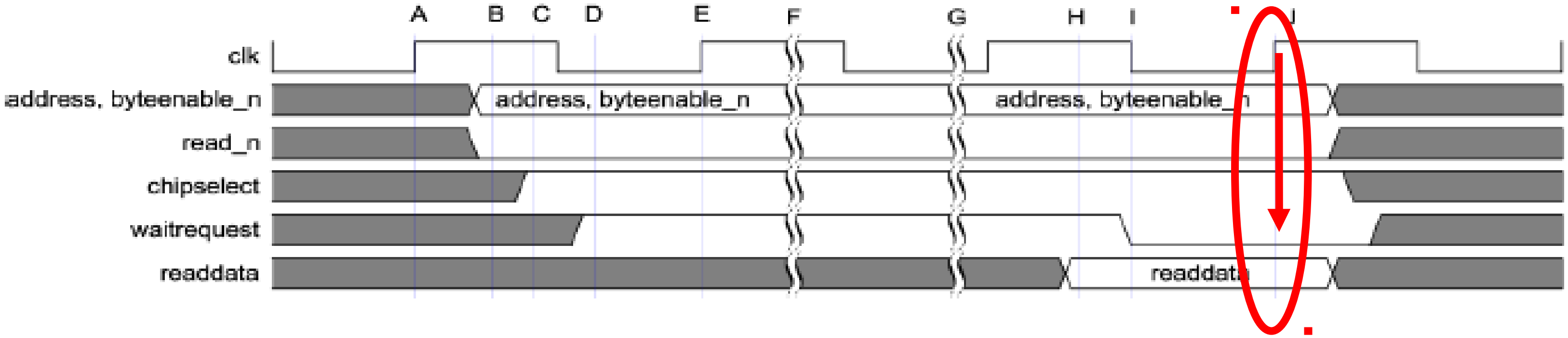

Peripheral
Master Port
Address
Data
Control
Avalon Bus Module
clk
address, byteenable_n
read_n
chipselect
readdata
Slave Port
Peripheral


A
B
C
D
E
F
G
H
clk
address, byteenable_n
address, byteenable_n
chipselect
Tsu
read_n
readdata
readdata

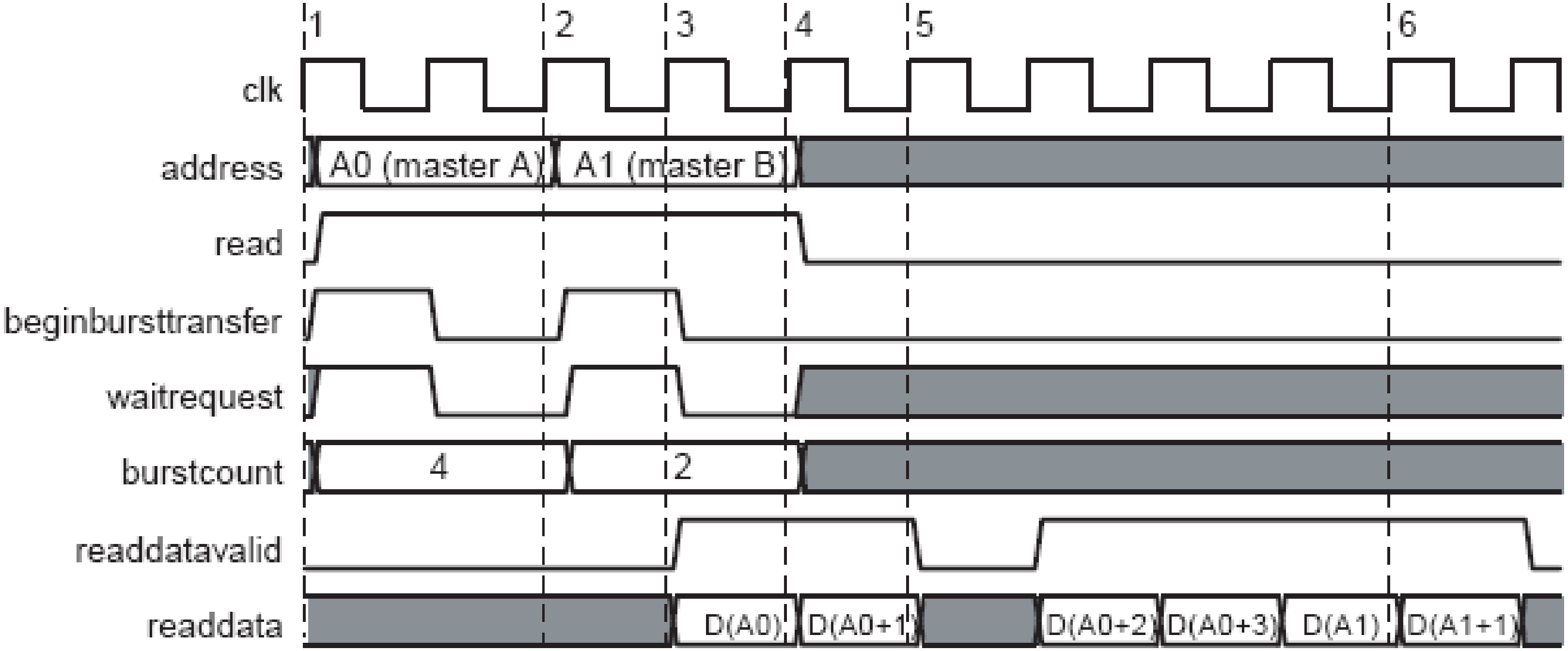
Avalon read

slave, wait request generated by slave device


Peripheral
Master Port
Address
Data
Control
wait request
Avalon Bus Module
clk
address, byteenable_n
read_n
chipselect
waitrequest
readdata
Slave Port
Peripheral


A
B
C
D
E
F
G
H
I
J
clk
address, byteenable_n
read_n
chipselect
waitrequest
readdata
address, byteenable_n
address, byteenable_n
readdata

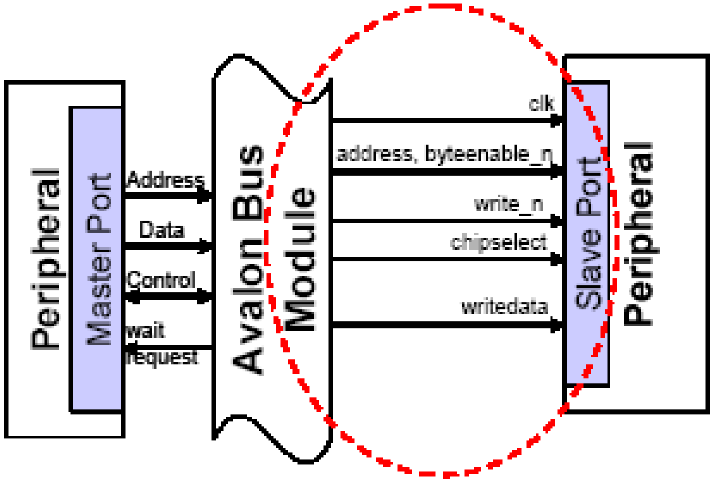
Avalon read
slave, burst of 4 from Master A, 2 from master B

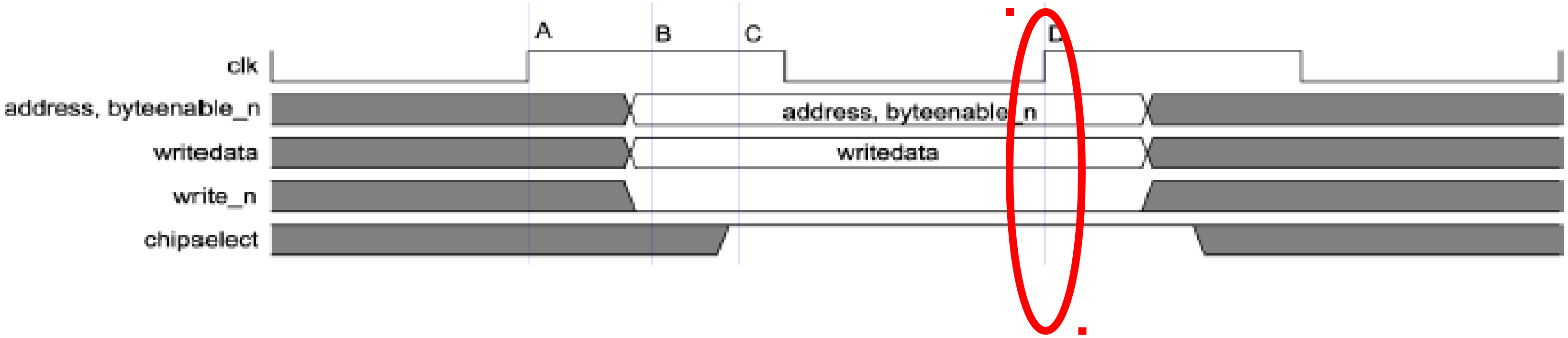

1
2
3
4
5
6
clk
address
A0 (master A)
A1 (master B)
read
beginbursttransfer
waitrequest
burstcount
4
2
readdatavalid
readdata
D(A0)
D(A0+1)
D(A0+2)
D(A0+3)
D(A1)
D(A1+1)

Pipeline of master access

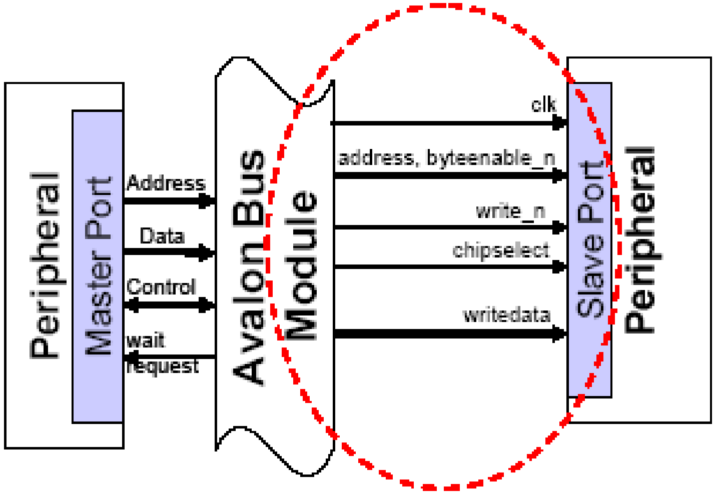
ReadDataValid activated by slave for each data

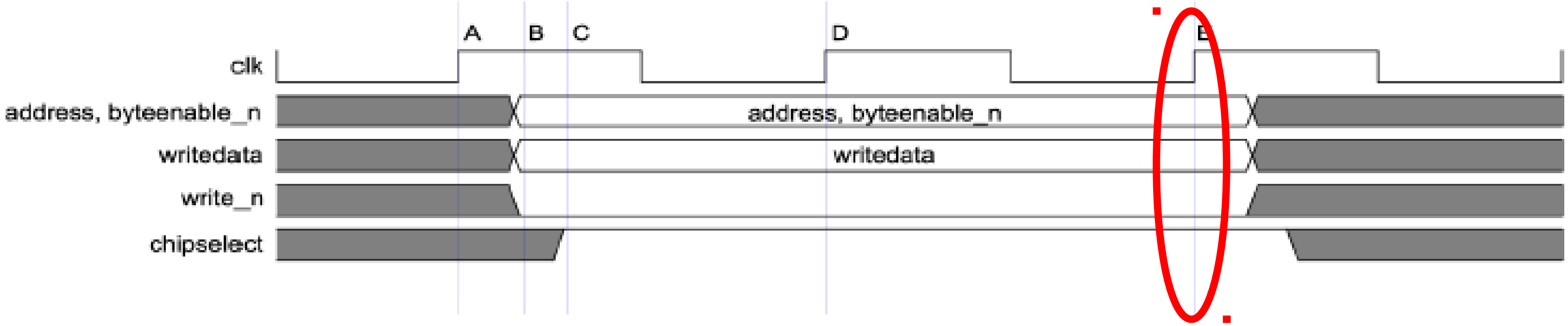
Avalon write slave, 0 wait


Peripheral
Master Port
Address
Data
Control
wait request
Avalon Bus Module
clk
address, byteenable_n
write_n
chipselect
writedata
Slave Port
Peripheral

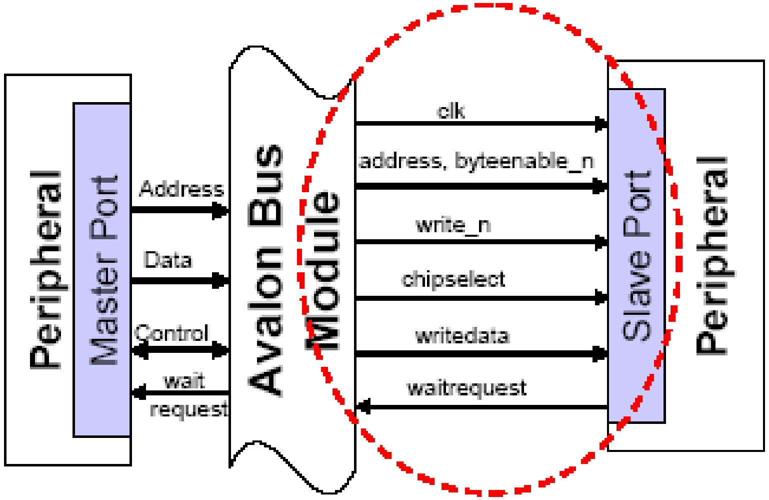

A
B
C
D
clk
address, byteenable_n
writedata
write_n
chipselect
address, byteenable_n
writedata

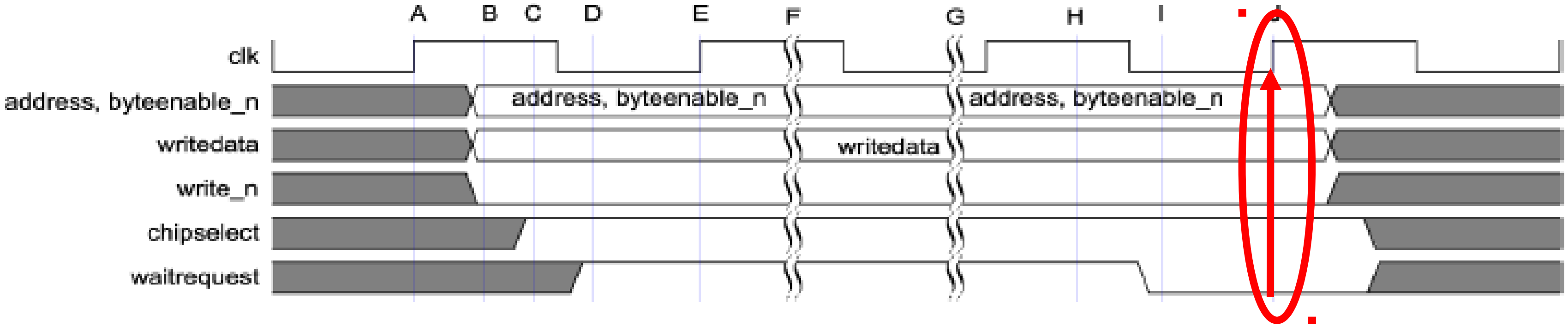
Avalon write slave, 1 wait


Peripheral
Master Port
Address
Data
Control
wait request
Avalon Bus Module
clk
address, byteenable_n
write_n
chipselect
writedata
Slave Port
Peripheral

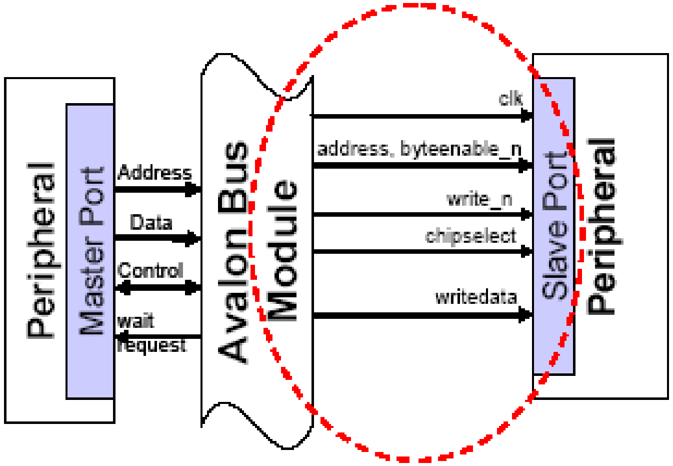

A
B
C
D
E
clk
address, byteenable_n
writedata
write_n
chipselect
address, byteenable_n
writedata

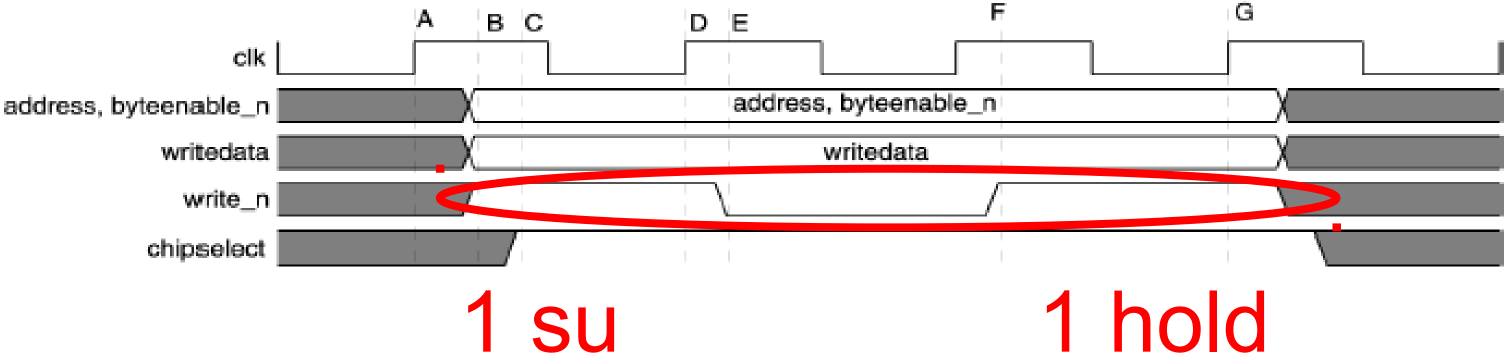
Avalon write
slave, wait request generated by slave


Peripheral
Master Port
Address
Data
Control
wait
request
Avalon Bus
Module
clk
address, byteenable_n
write_n
chipselect
writedata
waitrequest
Slave Port
Peripheral

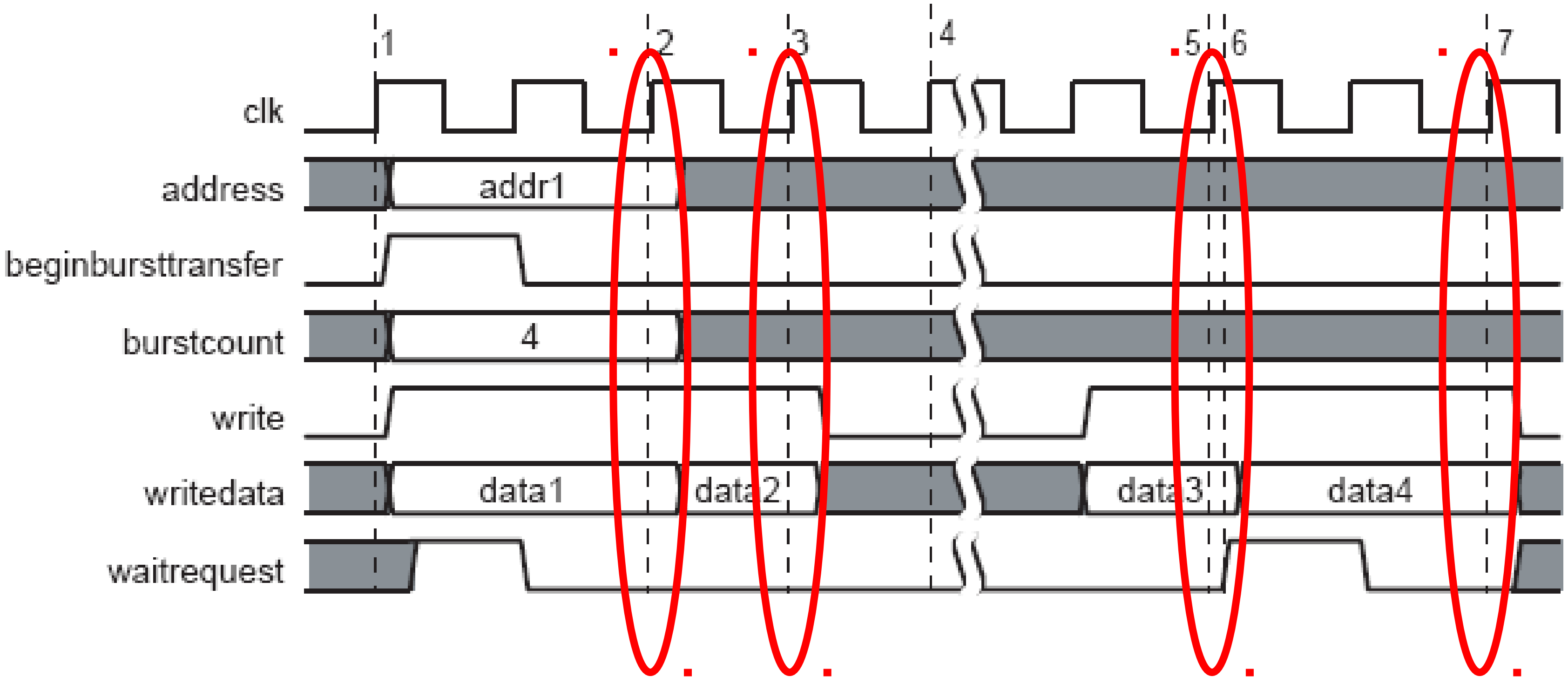

A
B
C
D
E
F
G
H
I
J
clk
address, byteenable_n
writedata
write_n
chipselect
waitrequest
address, byteenable_n
writedata
address, byteenable_n

Avalon write
slave, 1 set up, 1 hold, 0 wait


Peripheral
Master Port
Address
Data
Control
wait
request
Avalon Bus
Module
clk
address, byteenable_n
write_n
chipselect
writedata
Slave Port
Peripheral


A
B
C
D
E
F
G
clk
address, byteenable_n
writedata
write_n
chipselect
address, byteenable_n
writedata
1 su
1 hold

Avalon write
burst transfer of 4, wait request generated by slave

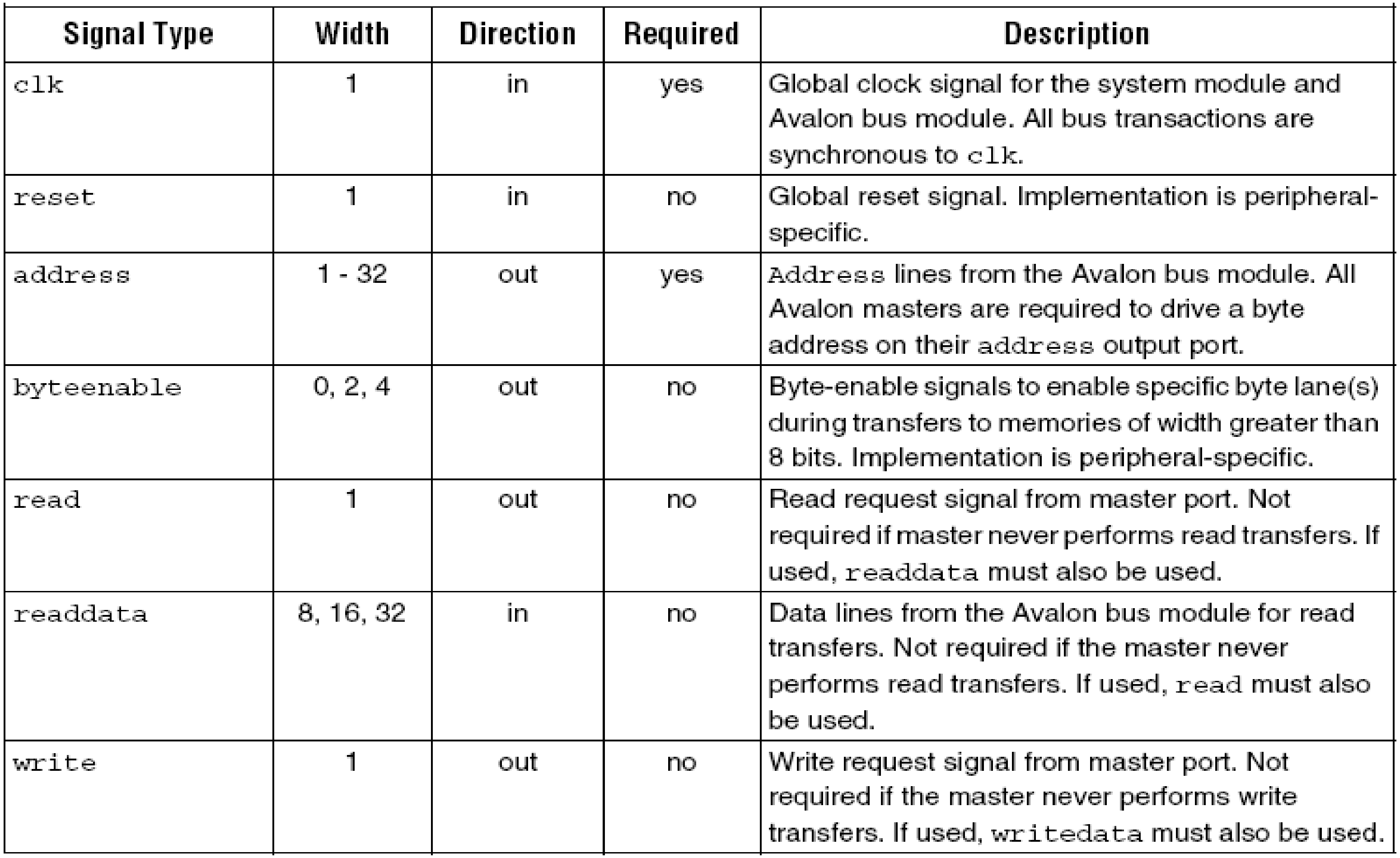

1
2
3
4
5
6
7
clk
address
addr1
beginbursttransfer
burstcount
4
write
writedata
data1
data2
data3
data4
waitrequest

Bus avalon

Master view

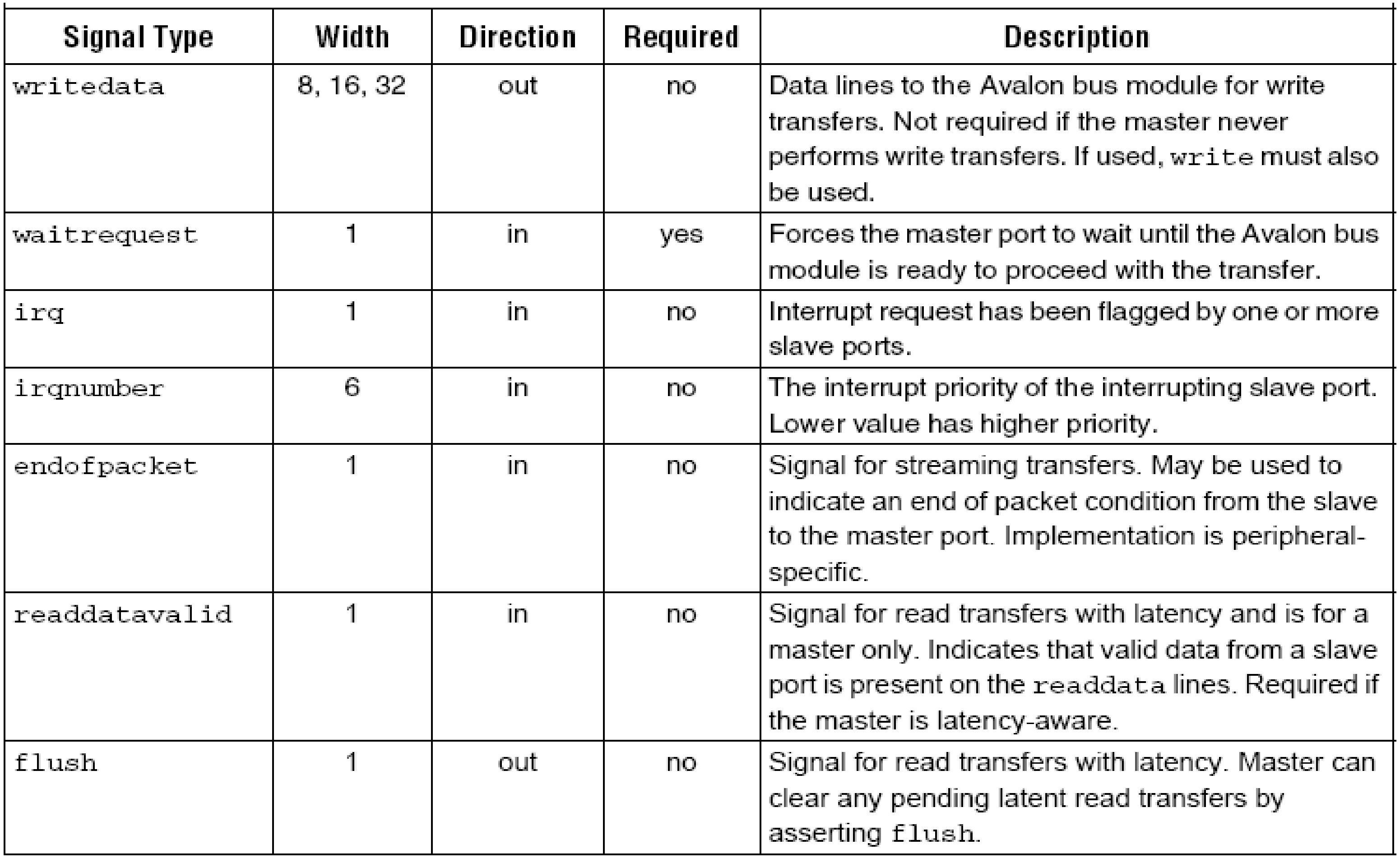
- The master starts a transfer (read or write)
- It provides the Addresses (32 bits on NIOSII)
- It waits on **WaitRequest** signal to resume the transfer

Avalon
master signals (1)



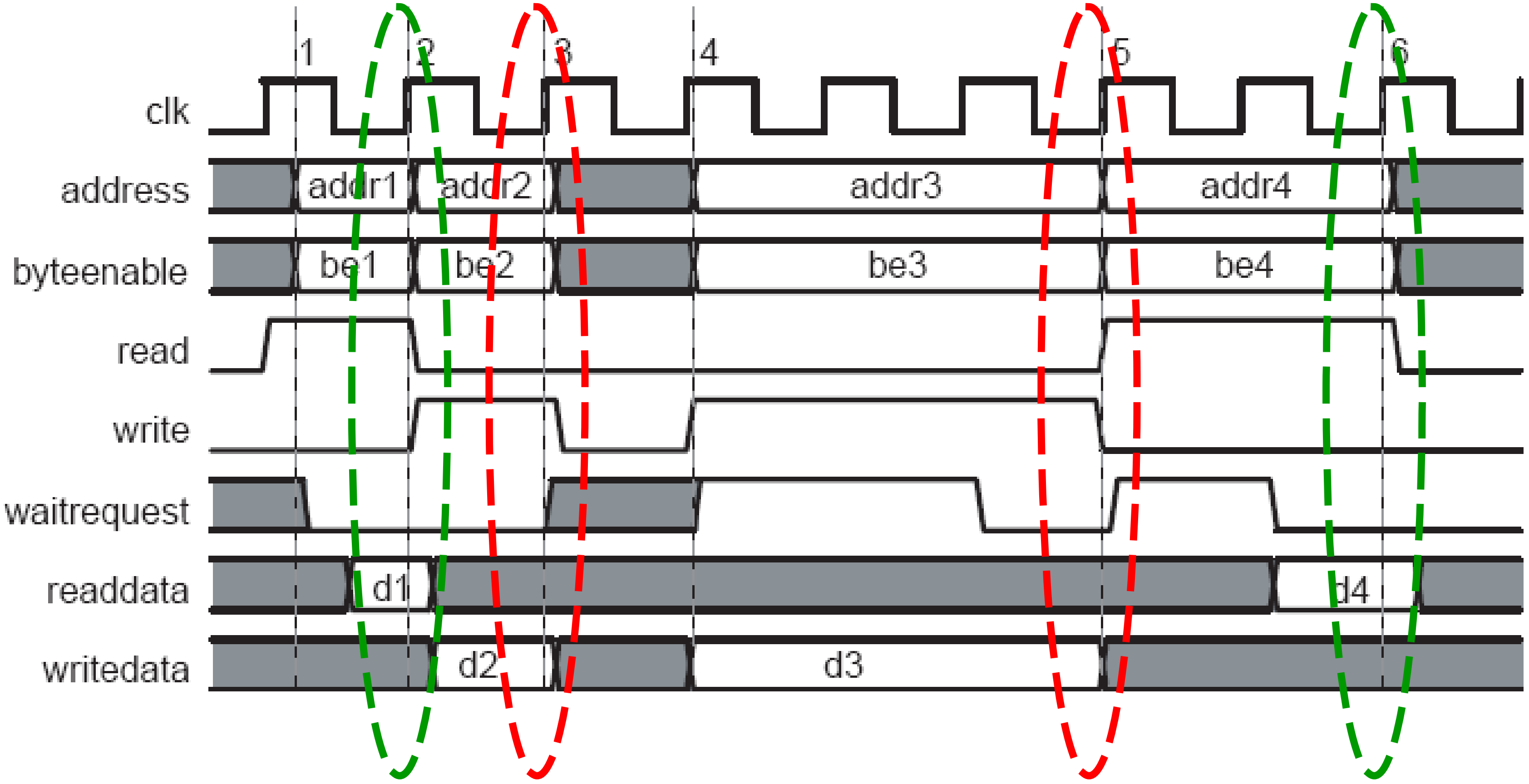
Signal Type	Width	Direction	Required	Description
`clk`	1	in	yes	Global clock signal for the system module and Avalon bus module. All bus transactions are synchronous to `clk`.
`reset`	1	in	no	Global reset signal. Implementation is peripheral-specific.
`address`	1 - 32	out	yes	`Address` lines from the 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)



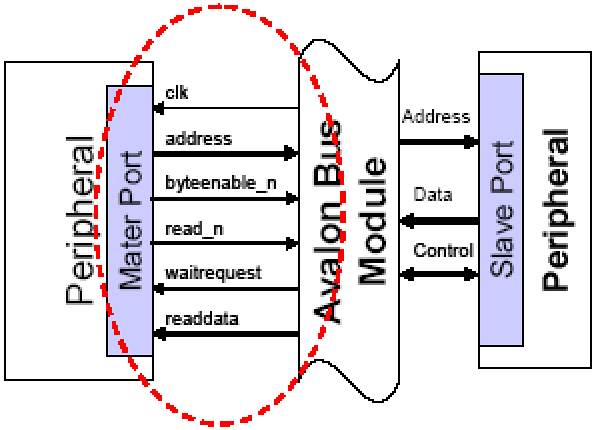
Signal Type	Width	Direction	Required	Description
`writedata`	8, 16, 32	out	no	Data lines to the Avalon bus module for write transfers. Not required if the master never performs write transfers. If used, `write` must also be used.
`waitrequest`	1	in	yes	Forces the master port to wait until the Avalon 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 `readdata` 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 `flush`.

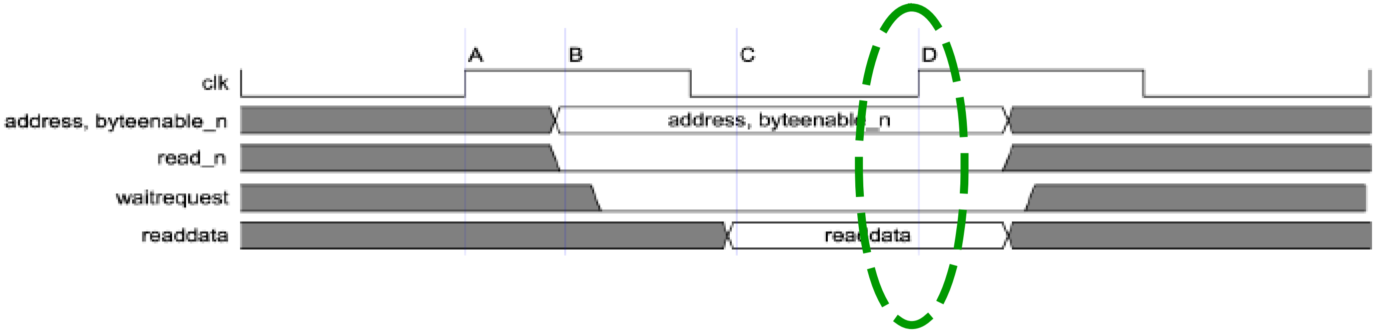
Avalon Master
Basic fundamental transfers


1
2
3
4
5
6
clk
address
addr1
addr2
addr3
addr4
byteenable
be1
be2
be3
be4
read
write
waitrequest
readdata
d1
d4
writedata
d2
d3

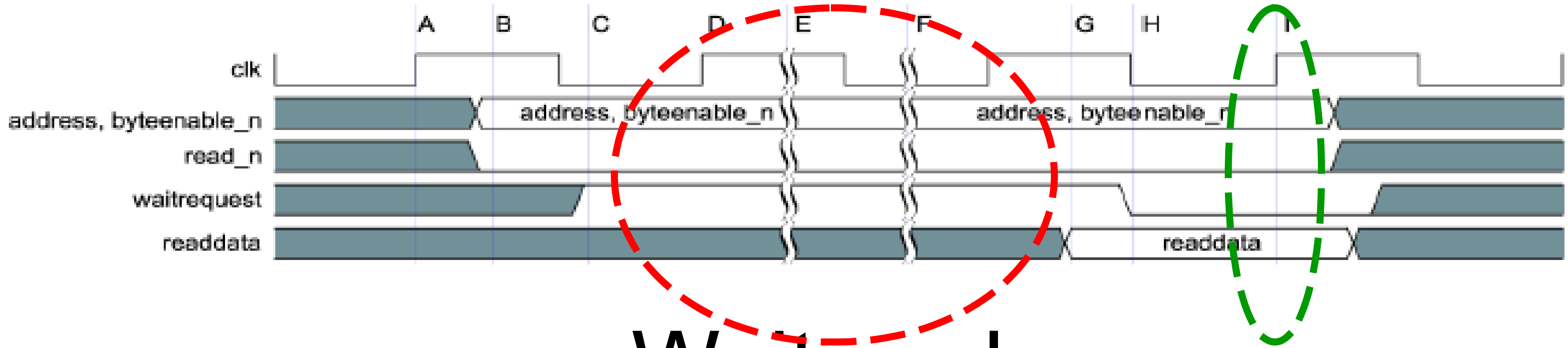
Wait Wait

Avalon read master, 0 wait


Peripheral
Mater Port
clk
address
byteenable_n
read_n
waitrequest
readdata
Avalon Bus Module
Address
Data
Control
Slave Port
Peripheral

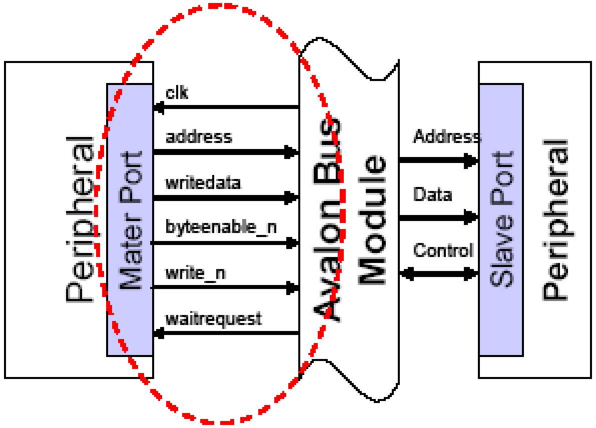

A
B
C
D
clk
address, byteenable_n
read_n
waitrequest
readdata
address, byteenable_n
readdata

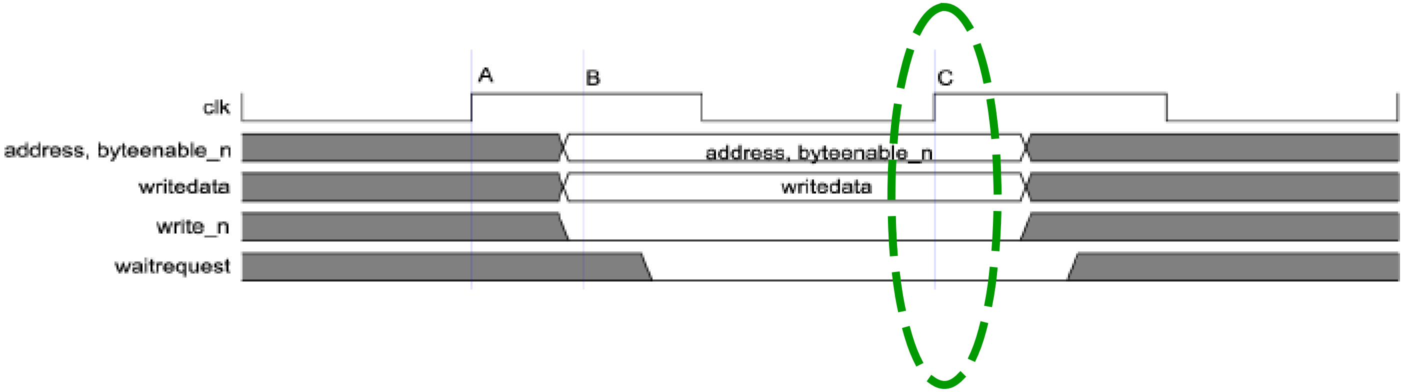
Avalon read
master, wait generated by slave/Avalon bus


A
B
C
D
E
F
G
H
I
clk
address, byteenable_n
read_n
waitrequest
readdata
address, byteenable_n
address, byteenable_n
readdata

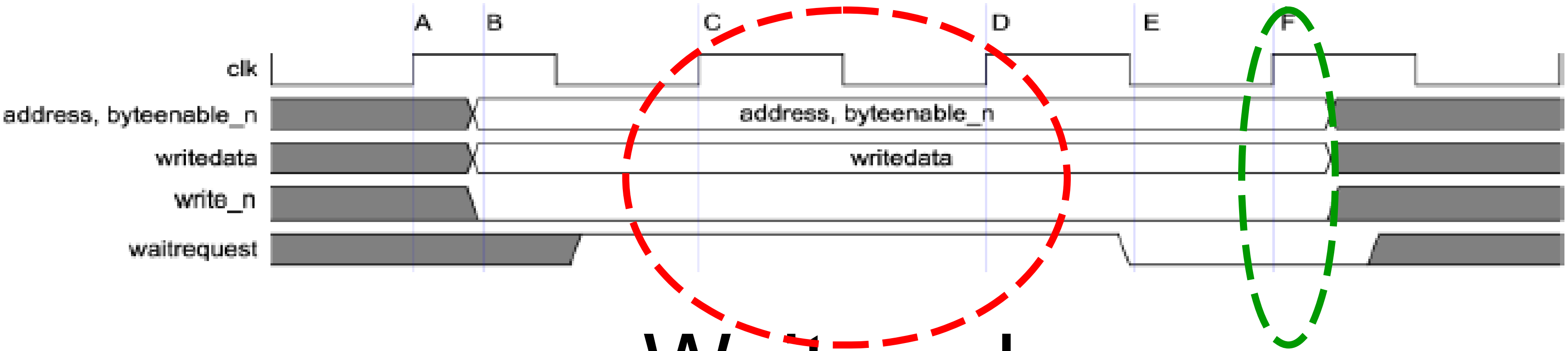
Wait cycles

Avalon write master, 0 wait


Peripheral
Mater Port
clk
address
writedata
byteenable_n
write_n
waitrequest
Avalon Bus Module
Address
Data
Control
Slave Port
Peripheral

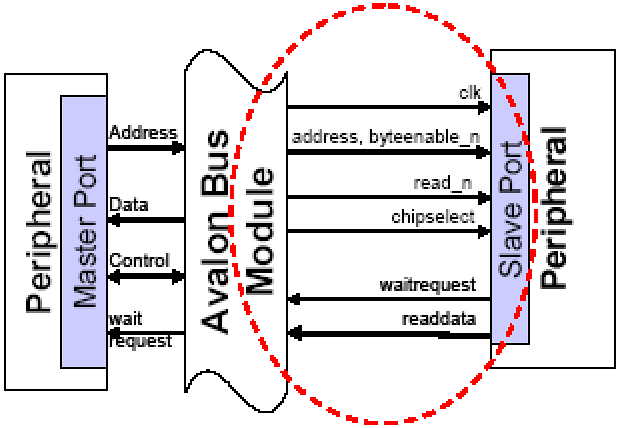

A
B
C
clk
address, byteenable_n
writedata
write_n
waitrequest
address, byteenable_n
writedata

Avalon write
master, wait generated by slave


A
B
C
D
E
F
clk
address, byteenable_n
writedata
write_n
waitrequest
address, byteenable_n
writedata

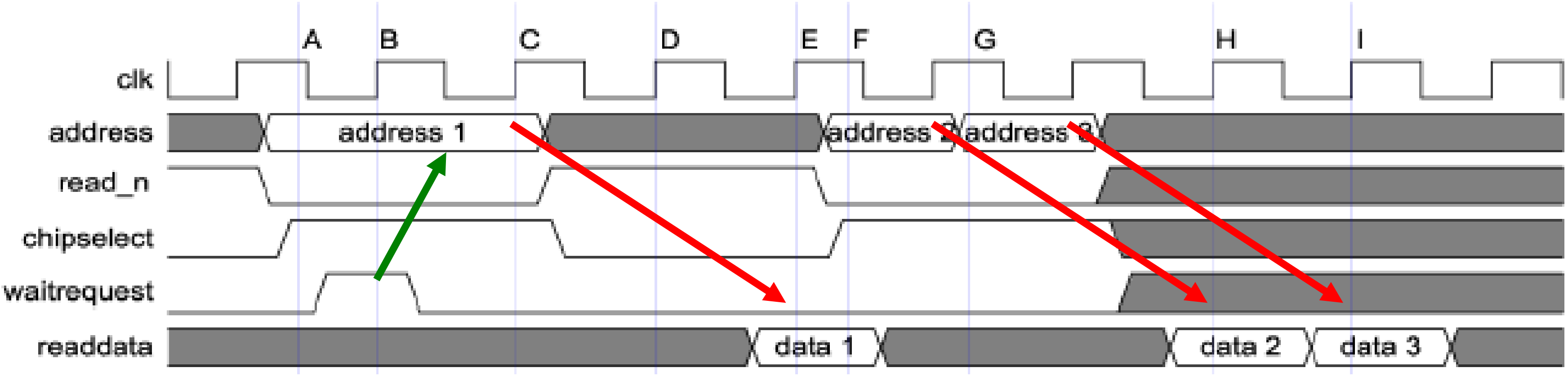
Wait cycles

Avalon read
transfers with latency (ex. 2 cycles)


Peripheral
Master Port
Address
Data
Control
wait request
Avalon Bus Module
clk
address, byteenable_n
read_n
chipselect
waitrequest
readdata
Slave Port
Peripheral

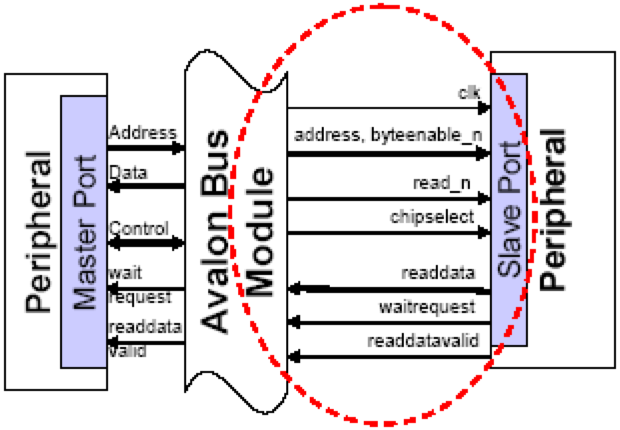
Wait request here means :

delay address cycle
Fixed latency (here 2)

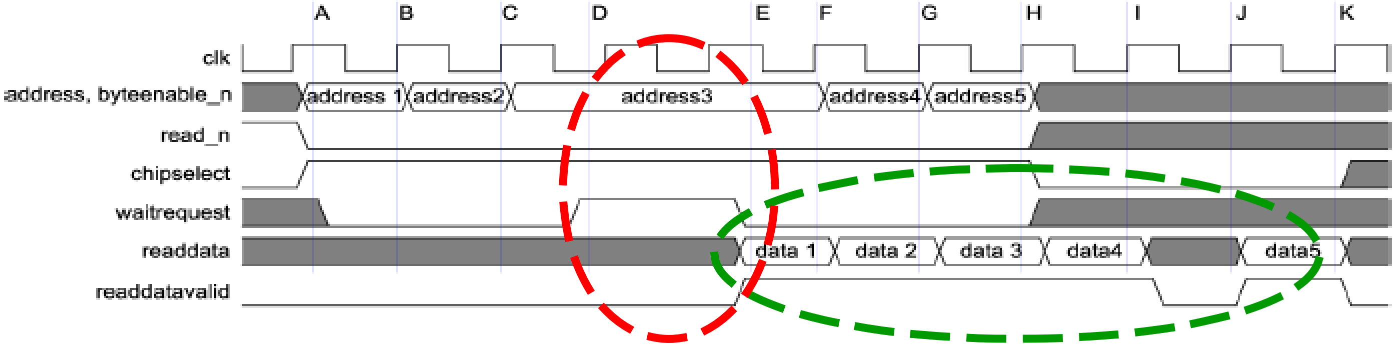

A
B
C
D
E
F
G
H
I
clk
address
address 1
address 2
address 3
read_n
chipselect
waitrequest
readdata
data 1
data 2
data 3

Avalon read

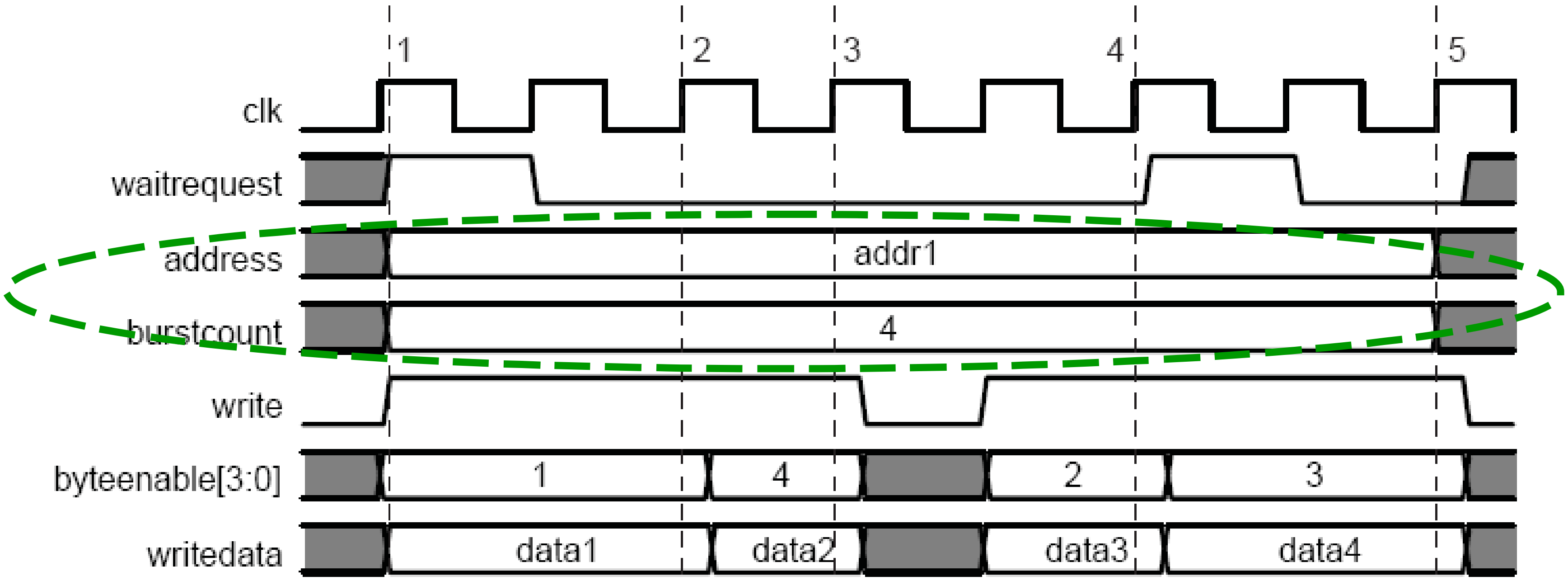
transfers with latency, and ***readdatavalid*** generated by slave


Peripheral
Master Port
Address
Data
Control
wait request
readdata valid
Avalon Bus Module
clk
address, byteenable_n
read_n
chipselect
readdata
waitrequest
readdatavalid
Slave Port
Peripheral

Readdatavalid specify when data are ready

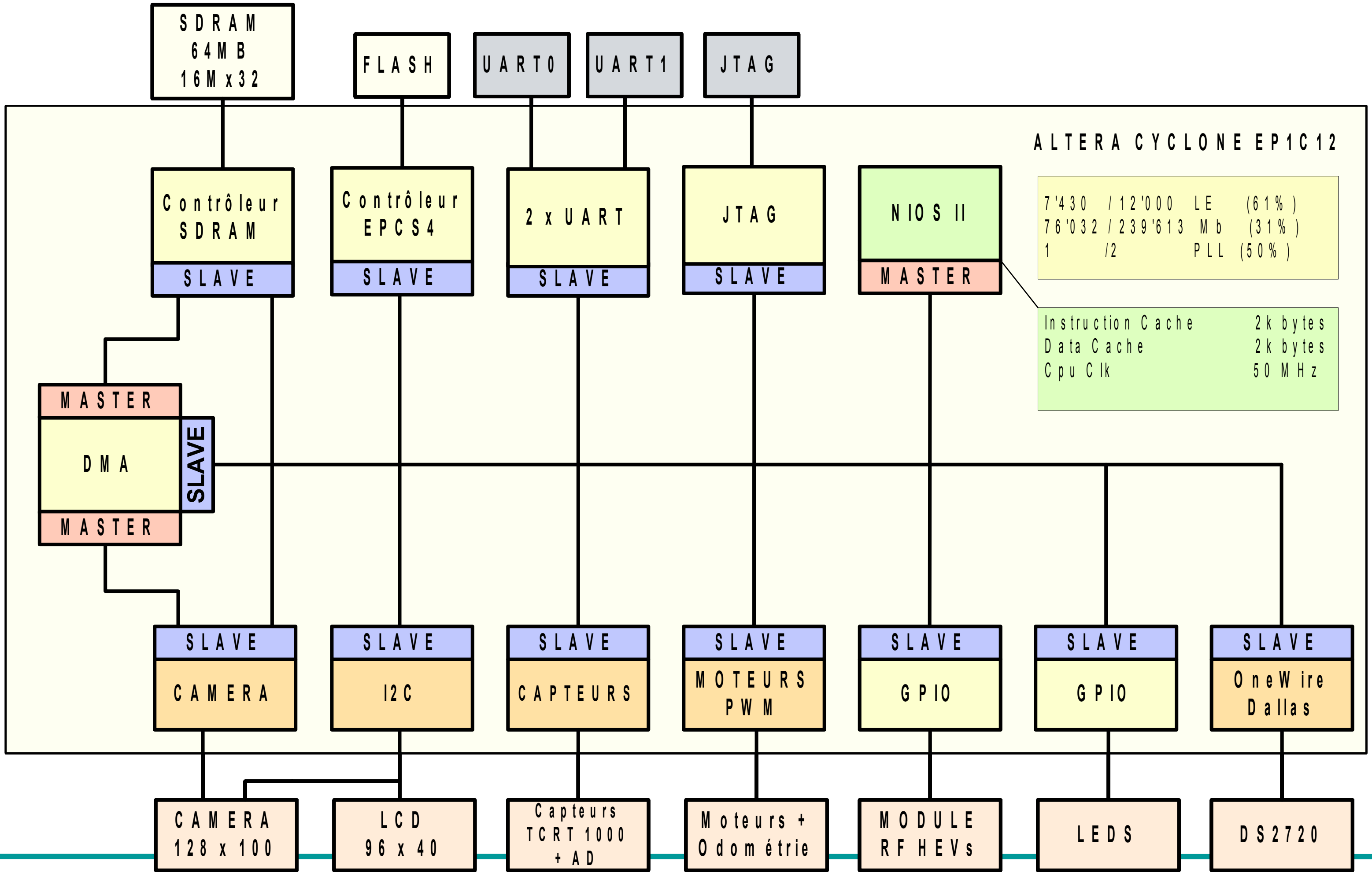

A
B
C
D
E
F
G
H
I
J
K
clk
address, byteenable_n
address 1
address2
address3
address4
address5
read_n
chipselect
waitrequest
readdata
data 1
data 2
data 3
data4
data5
readdatavalid

Avalon burst Write


1
2
3
4
5
clk
waitrequest
address
addr1
burstcount
4
write
byteenable[3:0]
1
4
2
3
writedata
data1
data2
data3
data4

Address and BurstCount available for the whole transfer
Write can be deactivated by the master
The number of burstcount needs to be generated

Embedded System on FPGA (example)



SDRAM
64MB
16M x 32

FLASH

UART0

UART1

JTAG

ALTERA CYCLONE EP1C12

Contrôleur SDRAM – SLAVE

Contrôleur EPCS4 – SLAVE

2 x UART – SLAVE

JTAG – SLAVE

NIOS II – MASTER

7'430 / 12'000 LE (61%)
76'032 / 239'613 Mb (31%)
1 /2 PLL (50%)

Instruction Cache 2k bytes
Data Cache 2k bytes
Cpu Clk 50 MHz

MASTER – DMA – SLAVE – MASTER

SLAVE – CAMERA

SLAVE – I2C

SLAVE – CAPTEURS

SLAVE – MOTEURS PWM

SLAVE – GPIO

SLAVE – GPIO

SLAVE – OneWire Dallas

CAMERA
128 x 100

LCD
96 x 40

Capteurs
TCRT 1000
+ AD

Moteurs +
Odométrie

MODULE
RF HEVs

LEDS

DS2720

Conclusion

Some positive points of a softcore architecture

- ❑ **Fast implementation**
- ❑ **Modular Architecture**
- ❑ **Affordable system complexity**
- ❑ **Good documentation**
- ❑ **Reuse of existing IP cores**
- ❑ **Ease of development of our own programmable interface on internal bus (i.e. Avalon in VHDL, Verilog)**
- ❑ **Full system on FPGA, easily adaptable**
- ❑ **Operating Systems available**

Some negative points

- ❑ **Several complex tools to develop a system: hardware and software specific**
- ❑ **Several debugging levels to deal with for hardware and software.**