Le bus Avalon

(bus propriétaire ALTERA)

Objectifs du bus:

- Assurer l'interconnexion entre le processeur (NIOS II) et des circuits périphériques (embarqués dans le FPGA ou non)

=> Deux parties:

* Avalon switch Fabric (gérée entièrement par l'outil SOPC builder)

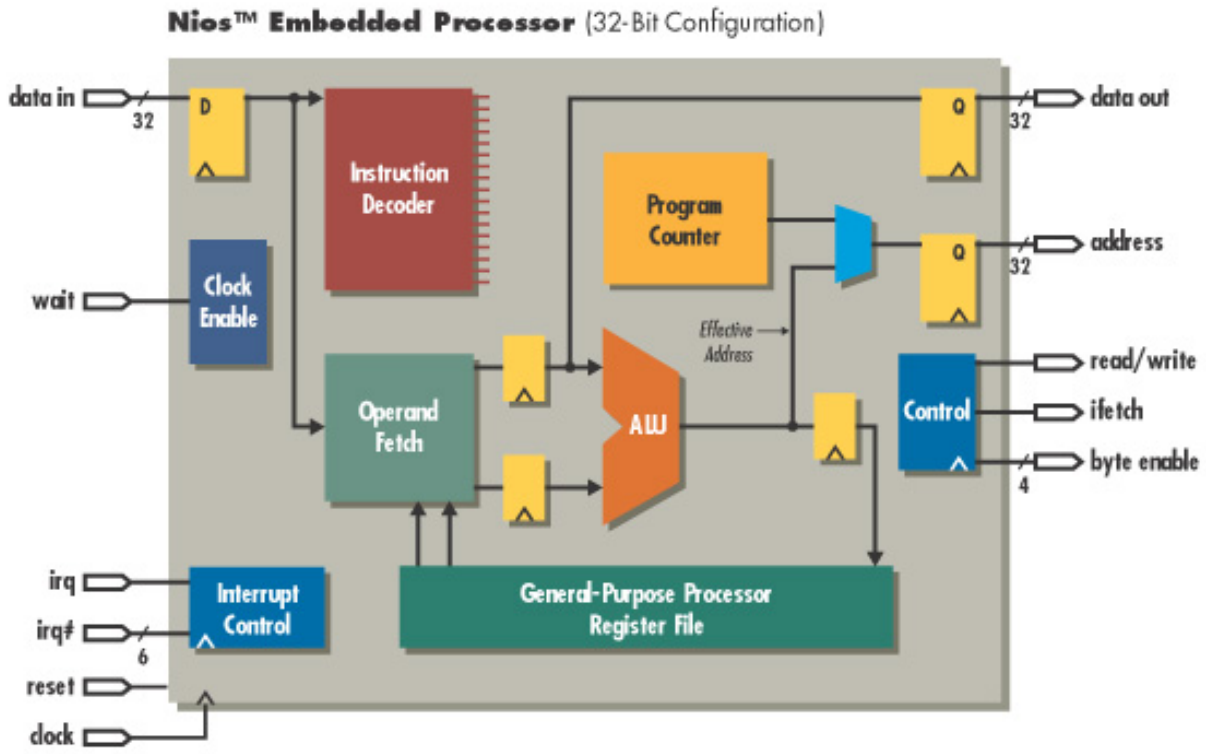
* Interface Avalon côté circuit périphérique (gérée par le concepteur si embarqué)

NIOS II Overview

- Soft IP Core
  - A *soft-core processor* is a microprocessor fully described in software, usually in an HDL, which can be synthesized in programmable hardware, such as FPGAs.
- Reduced Instruction Set Computer (RISC)
- No pipeline, 5 or 6 stages pipeline configurations
- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- 32 external interrupt sources
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Software development environment based on the GNU C/C++ tool chain and Eclipse IDE

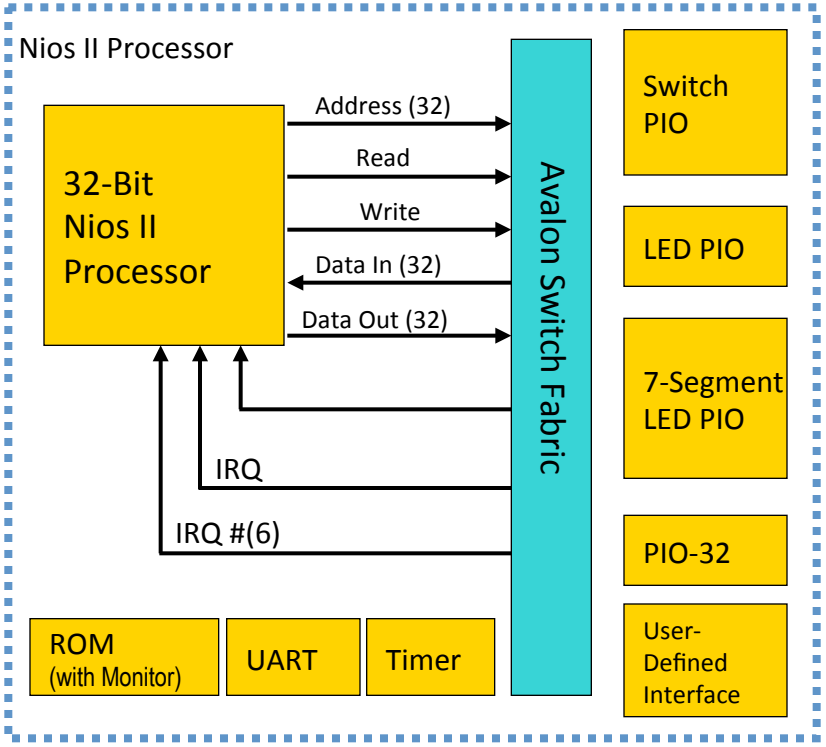
Nios II RISC Processor Block Diagram

- Standard RISC Components
- Harvard architecture
- Fully-Synchronous Interface
- Native Verilog
- Native VHDL

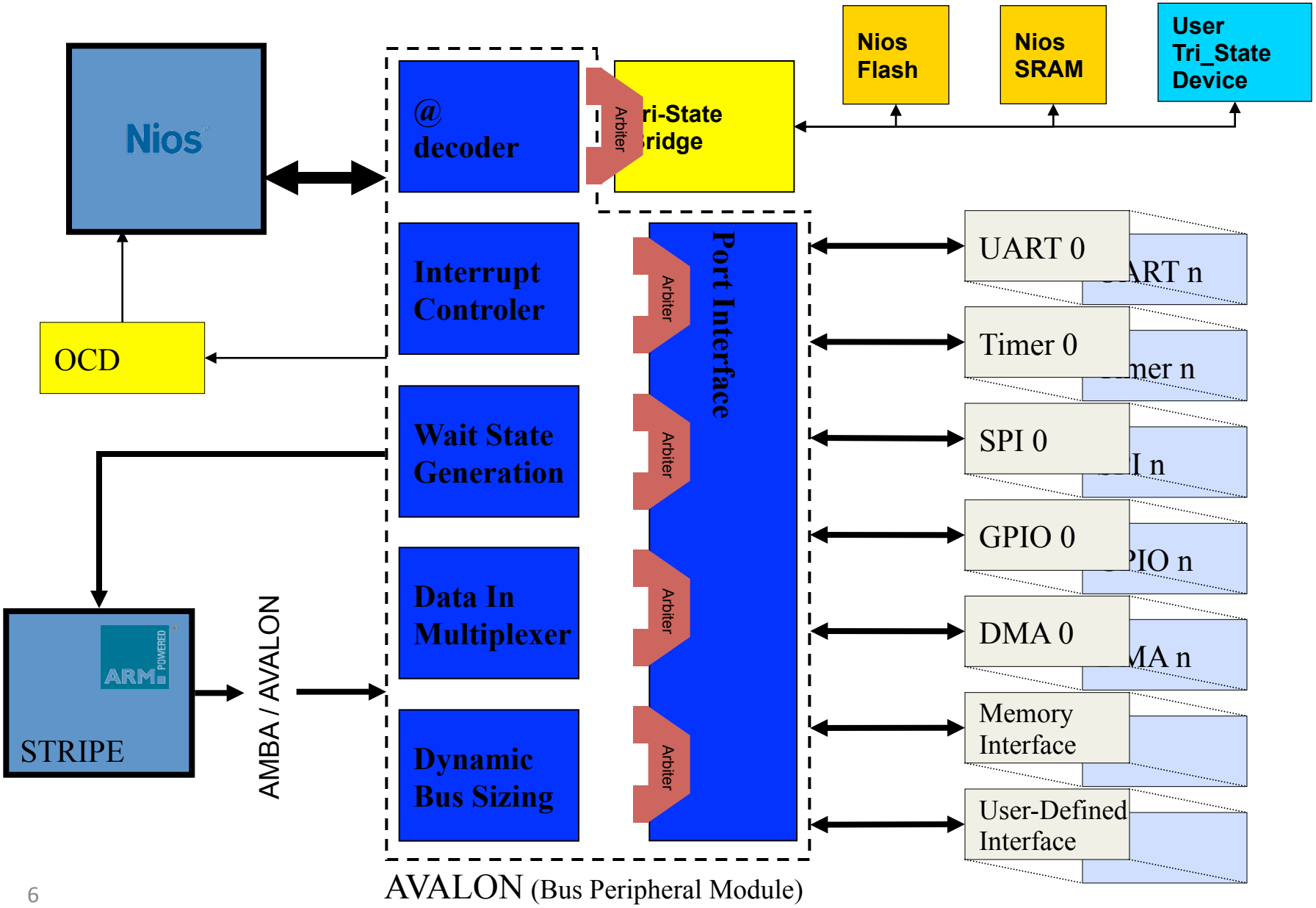


Avalon Switch Fabric

- Proprietary interconnect specification used with Nios II
- Principal design goals
  - Low resource utilization for bus logic
  - Simplicity
  - Synchronous operation
- Transfer Types
  - Slave Transfers
  - Master Transfers
  - Streaming Transfers
  - Latency-Aware Transfers
  - Burst Transfers



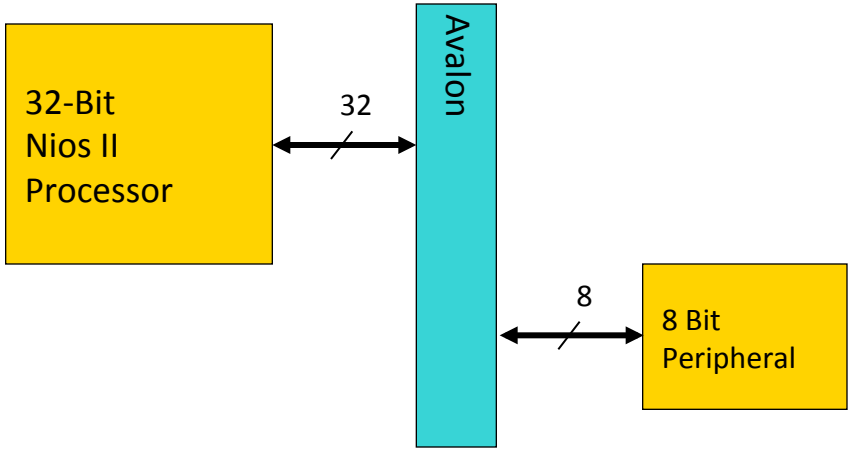
The Nios™ System Architecture



Avalon Switch Fabric

- Custom-Generated for Peripherals
  - Contingencies are on a Per-Peripheral Basis
  - System is Not Burdened by Bus Complexity
- SOPC Builder Automatically Generates
  - Arbitration
  - Address Decoding
  - Data Path Multiplexing
  - Bus Sizing
  - Wait-State Generation
  - Interrupts

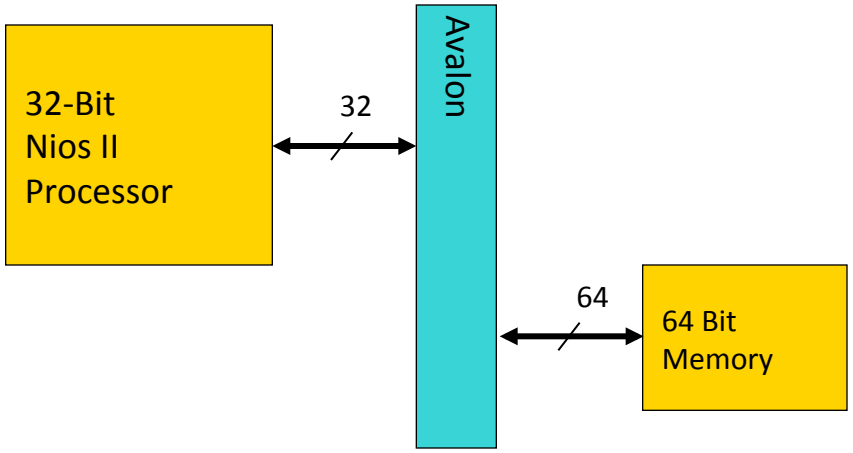
Address Alignment/Bus sizing – Narrow Slave



Peripheral Registers	
Base	aa
Base + 0x1	bb
Base + 0x2	cc
Base + 0x3	dd
Base + 0x4	ee

- **Dynamic Address Alignment** (set as **Memory Slave**)
  - LD from Base + 0x0: dd cc bb aa
  - LD from Base + 0x4: uu uu uu ee
- **Native Address Alignment** (set as **Avalon Register Slave**)
  - LD from Base + 0x0: uu uu uu aa
  - LD from Base + 0x4: uu uu uu bb
  - LD from Base + 0x8: uu uu uu cc

Address Alignment/Bus sizing – Narrow Master



Memory Contents	
Base	77 66 55 44 33 22 11 00
Base + 0x8	ff ee dd cc bb aa 99 88
Base + 0x16	?? ?? ?? ?? ?? ?? ?? ??

- **Dynamic Address Alignment**
  - LD from Base + 0x0: 33 22 11 00
  - LD from Base + 0x4: 77 66 55 44
  - LD from Base + 0x8: bb aa 99 88
- **Native Address Alignment**
  - LD from Base + 0x0: 33 22 11 00
  - LD from Base + 0x4: bb aa 99 88
  - LD from Base + 0x8: ?? ?? ?? ??
  - *High bytes are unobtainable – warning issued*

Avalon Slave Port Signals

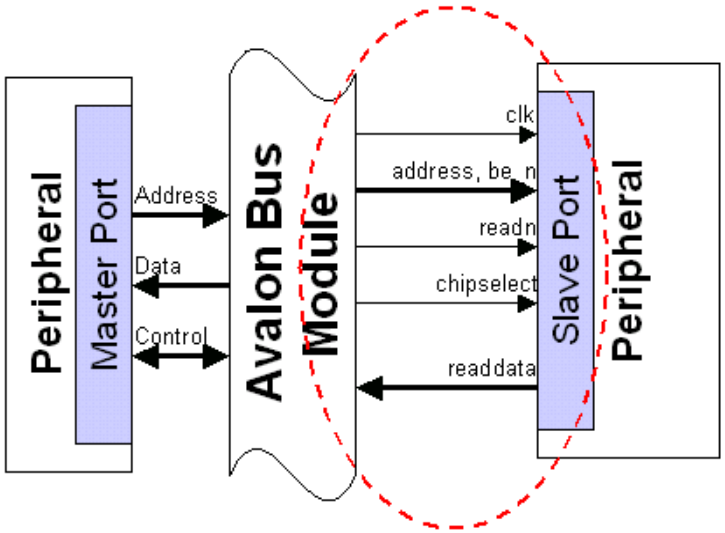
- A basic slave port contains
  - clock
  - address
  - read,write
  - readdata,writedata
  - Begintransfer (si burst)
  - chipselect
  - Byteenable (selon type de périphérique)

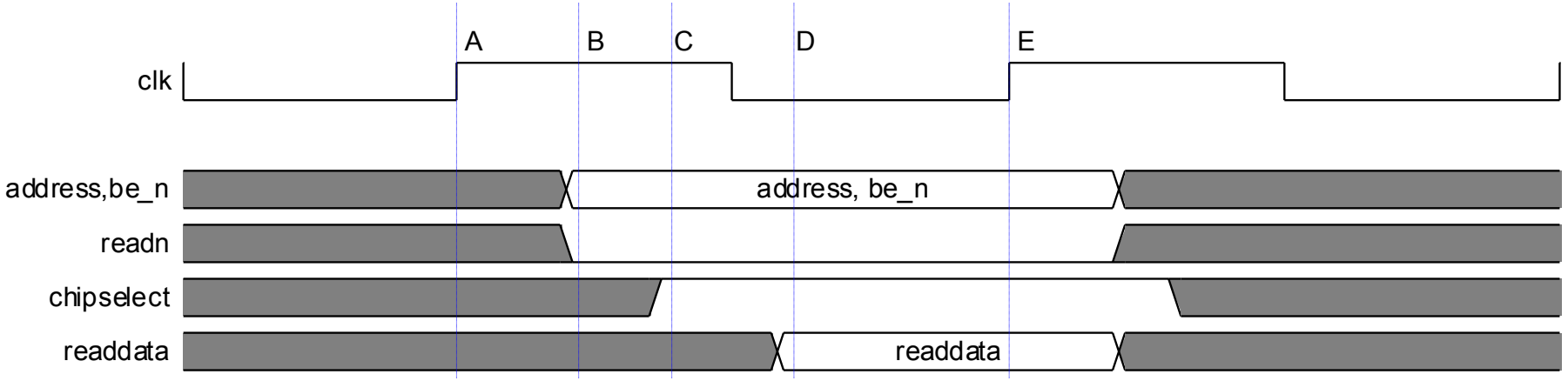
Avalon
« slave » main signals

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*	*Old signal*	*Selection of the Avalon bus module*
read/ read_n	1	In	No	Read request to the slave
ReadData	8, 16, 32, .. (1024)	Out	No	Read data from the slave module
write/ write_n	1	In	No	Write request to the slave
WriteData	8, 16, 32, .. (1024)	In	No	Data from Master to Slave module
Irq	1	Out	No	Interrupt request to the master

Slave Read Transfer

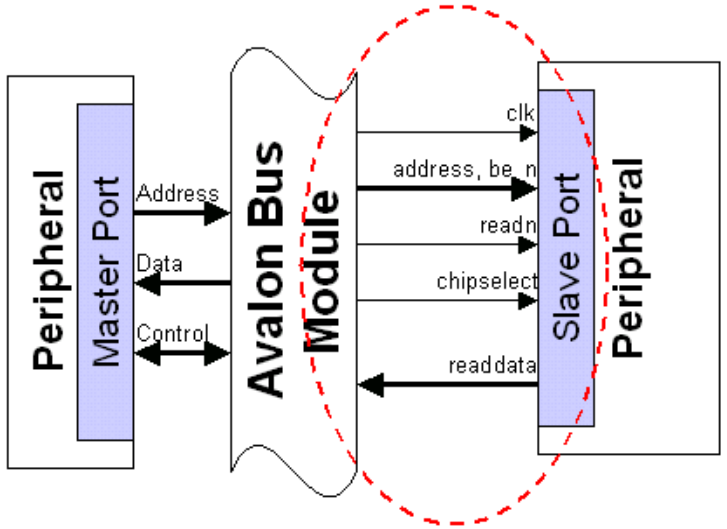
- 0 Setup Cycles
- 0 Wait Cycles

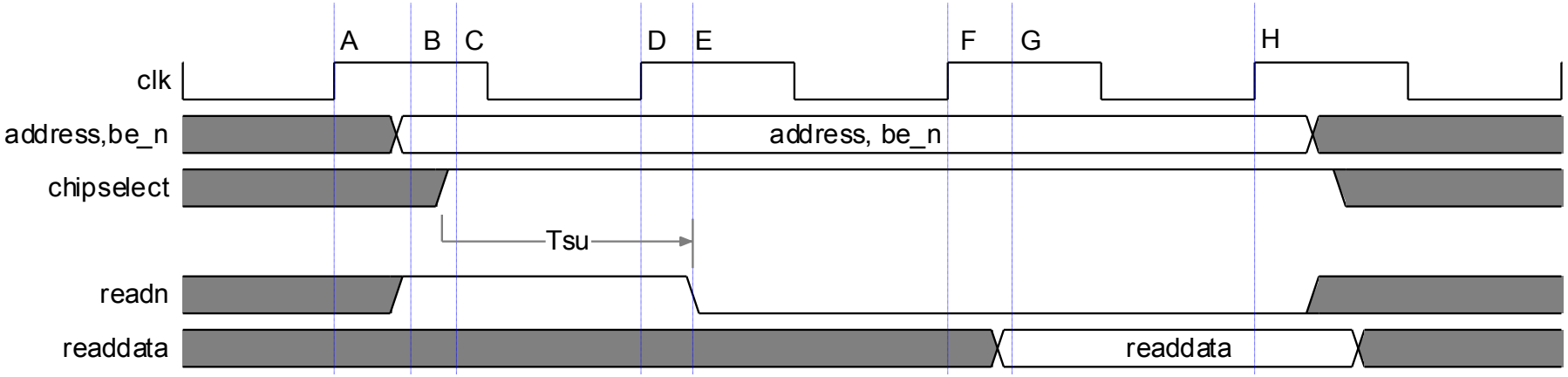




Slave Read Transfer with Wait States

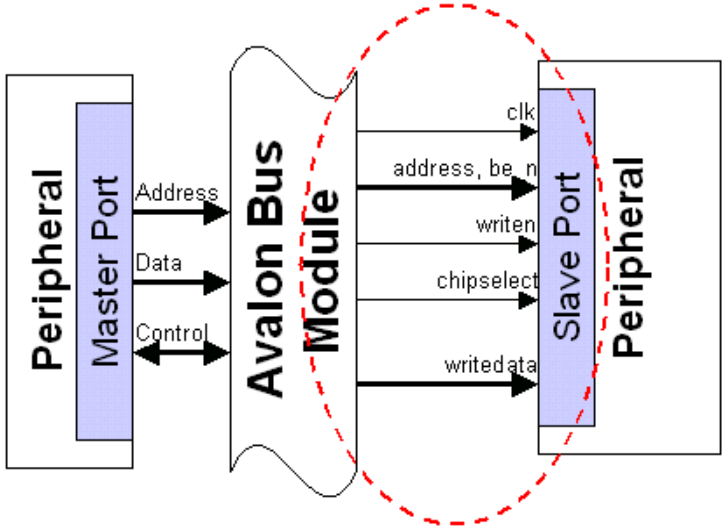
- 1 Setup Cycle
- 1 Wait Cycle

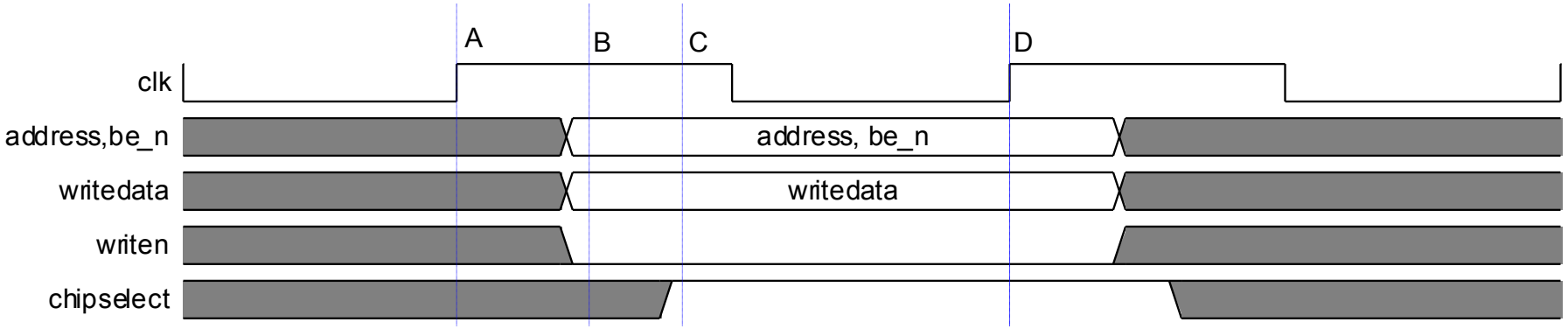




Slave Write Transfer

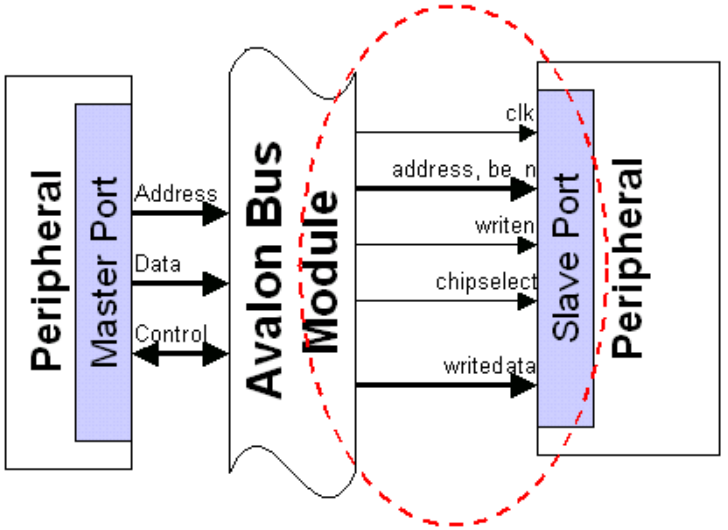
- 0 Setup Cycles
- 0 Wait Cycles
- 0 Hold Cycles

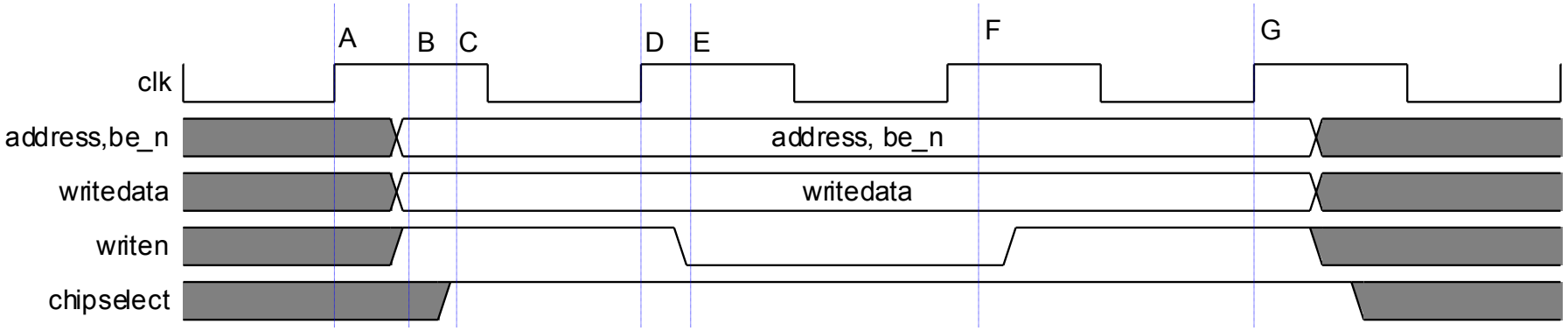




Slave Write Transfer with Wait States

- 1 Setup Cycle
- 0 Wait Cycles
- 1 Hold Cycle





Circuits propriétaires

- Pourquoi ajouter un circuit propriétaire?
  - Nécessité de créer des fonctions non présentes en librairies
  - Nécessité d'accélérer certains traitements
  - Nécessité de décharger le processeur de certaines tâches

- Comment connecter le circuit au NIOS via Avalon
  - Directement
  - Via des ports //

Connexion directe
Circuits propriétaires- L'entité

```
Entity mon_circuit is
port (
clk, chipselect, write_n, address, reset_n : in std_logic;
in_pwm : in std_logic;
writedata : in std_logic_vector (31 downto 0);
readdata : out std_logic_vector (31 downto 0)
);
end entity;
```

Nota: Signaux en rouge indispensables

Circuits propriétaires- L'interface Avalon
Ecriture

```
registers: process (clk, reset_n)
begin
        if reset_n = '0' then
        config <= (others => '0');
        elsif clk'event and clk = '1' then
                if chipselect ='1' and write_n = '0' then
                        if address = '0' then
                        config <= (writedata (2 downto 0));
                        end if;
                end if;
        end if;
end process registers;
```

Nota: config déclaré comme signal dans l'architecture:
signal config: std_logic_vector (2 downto 0);

Circuits propriétaires- L’interface Avalon Lecture

```
readdata(2 downto 0) <= config when address = '0' ;
```

Nota: Un test sur le chipselect n’est pas indispensable car pris en compte au niveau du Avalon Switch Fabric

Programme en C: exemple

```
#define freq_verin (volatile int *) GESTION_VERIN_0_BASE
#define duty_verin (volatile int *) (GESTION_VERIN_0_BASE + 4)
*freq_verin= 2000;   (le registre pointé par @ freq_verin =2000)
*duty_verin=1500;
a=*freq_verin;
```

Nota: On peut utiliser les instructions suivantes:

- IORD(BASE, offset)
- IOWR(BASE, offset, DATA)

Flot de conception du SOPC