LYON CPE ÉCOLE SUPÉRIEURE DE CHIMIE PHYSIQUE ÉLECTRONIQUE DE LYON

Conception de contrôleur moteur BLDC sur FPGA

Abdelbassat MASSOURI

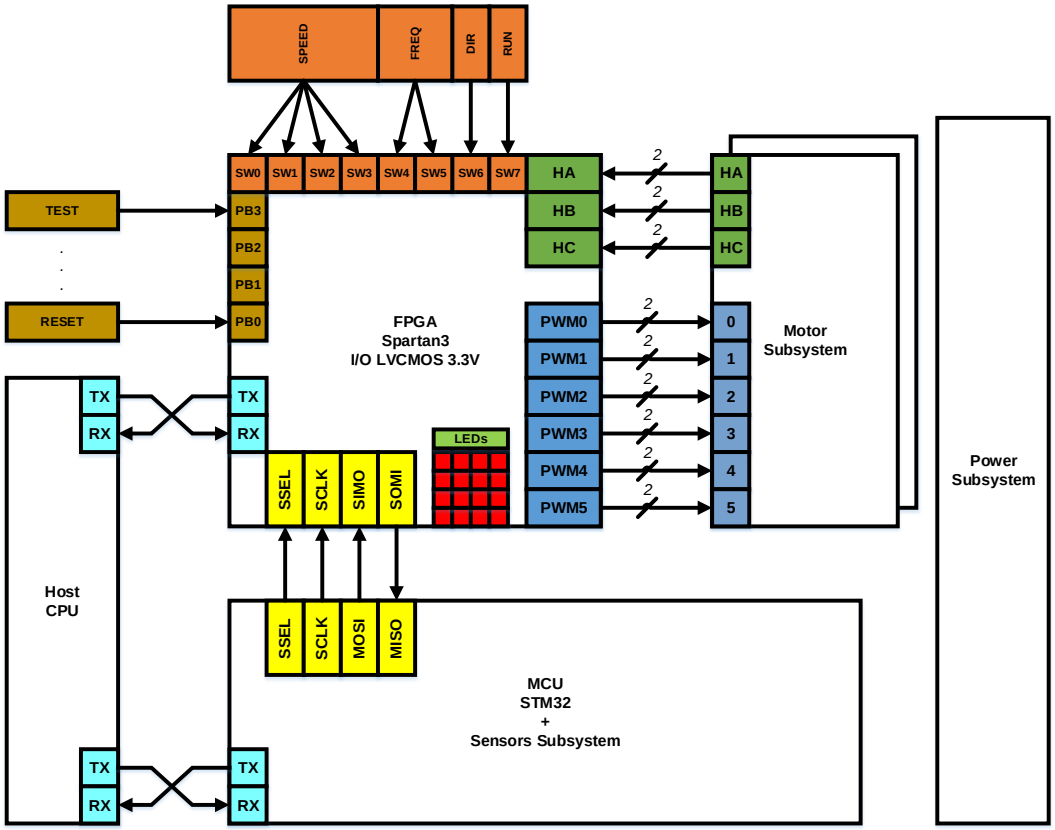
abdelbassat.massouri@cpe.fr

5ETI – 2019/2020

Plan

- Contexte du Projet
- Partie #1 : ISE/VHDL – Séquenceur
  - **I. Séquenceur**
    - **1. Exemple de séquence de commutation d'un moteur BLDC**
    - **2. Block et I/O**
  - **II. Plateforme FPGA**
  - **III. Rappel VHDL**
  - **IV. Prise en main ISE et ISim**
    - **1. Exemple #1:**
    - **2. Exemple #2:**
    - **3. Exemple #3:**
- Partie #2 : ISE/VHDL – Communication Série SPI/UART
  - **I. Protocole UART**
    - **1. Chronogramme UART**
    - **2. FSM UART**
  - **II. Protocole SPI**
    - **1. Chronogramme SPI**
    - **2. FSM SPI**
- Partie #3 : ISE/VHDL – Protocole de communication, configuration et commande moteur
  - **I. Commandes de configuration**
  - **II. Format du paquet**
  - **III. Diagramme de séquence**
  - **IV. FSM/ASM**

Contexte : Architecture système du projet



- Commande Moteur
- Séquence de commutation
- 2 sens de rotation
- Capteur à Effet-Hall
- PWM – Vitesse ou Duty Cycle

- Boutons Switch
- Marche/Arrêt,
- Sens de rotation, Vitesse

- Communication UART PC/FPGA
- Protocole – control des 2 moteurs

- Communication SPI STM32/FPGA
- Protocole – control des 2 moteurs
- PC -> STM32 -> FPGA -> Moteurs

- Intégration FPGA, Moteur, STM32, Capteurs…
- Séquence de démarrage envoyée depuis l'ordinateur pour démarrer et arrêter les 2 moteurs
- Interface graphique (GUI – PyQT) – Interaction et commande temps-réel des deux moteurs

Partie #1 : ISE/VHDL – Séquenceur

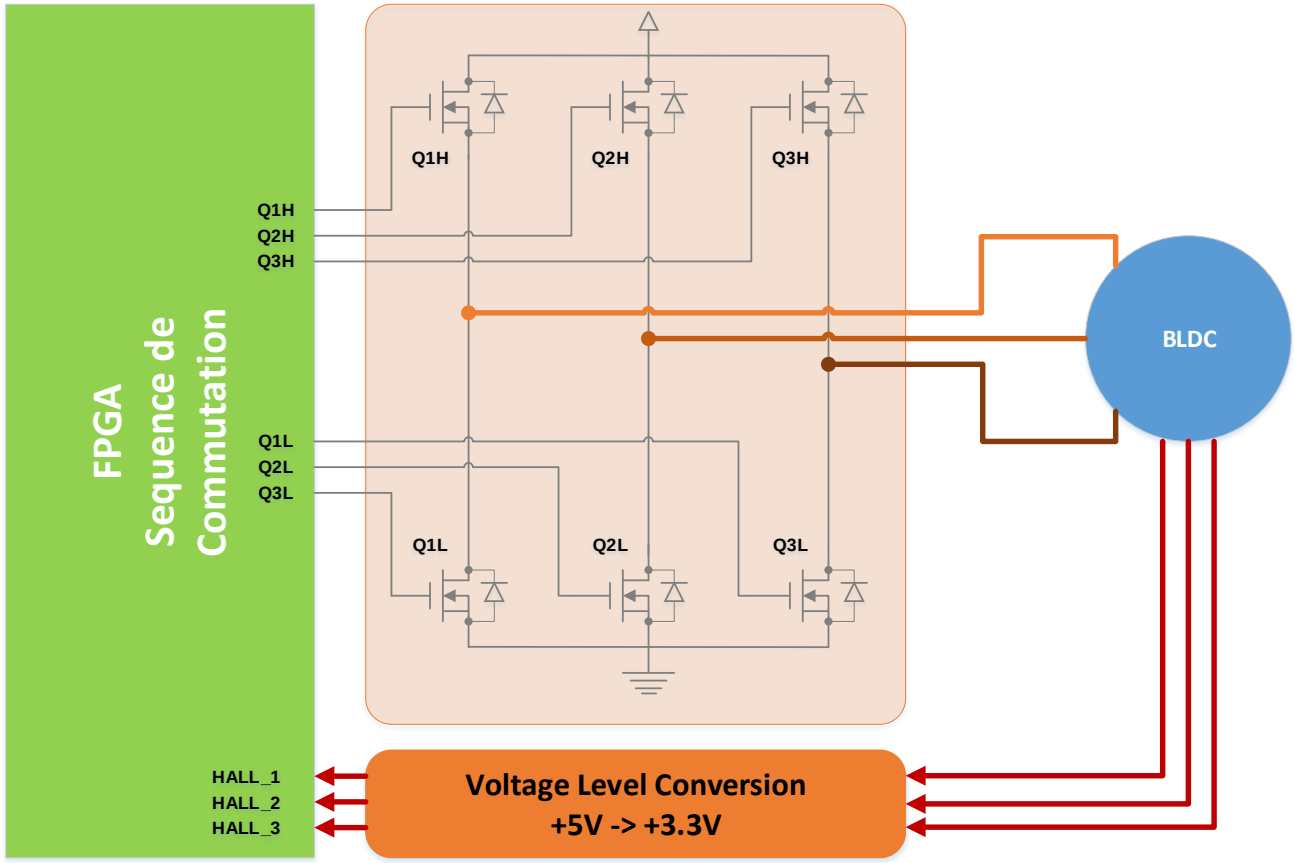
I. **Séquenceur**
   1. **Exemple de séquence de commutation d'un moteur BLDC**
   2. **Block et I/O**

II. Plateforme FPGA
   I. Kit d'évaluation Spartan-3
   II. Architecture FPGA SPARTAN-3

III. Rappel VHDL
   1. Combinatoire
   2. Séquentiel
   3. Concurrentiel

IV. Prise en main ISE et ISim
   1. Exemple #1: Switch - Afficheur 7-segment
   2. Exemple #2: Compteur et Diviseur de fréquence
      a. Compteur – Afficheur 7-segment
      b. Horloge 1MHz dérivée à partir de la référence 50 MHz
   3. Exemple #3: Implémentation de module PWM

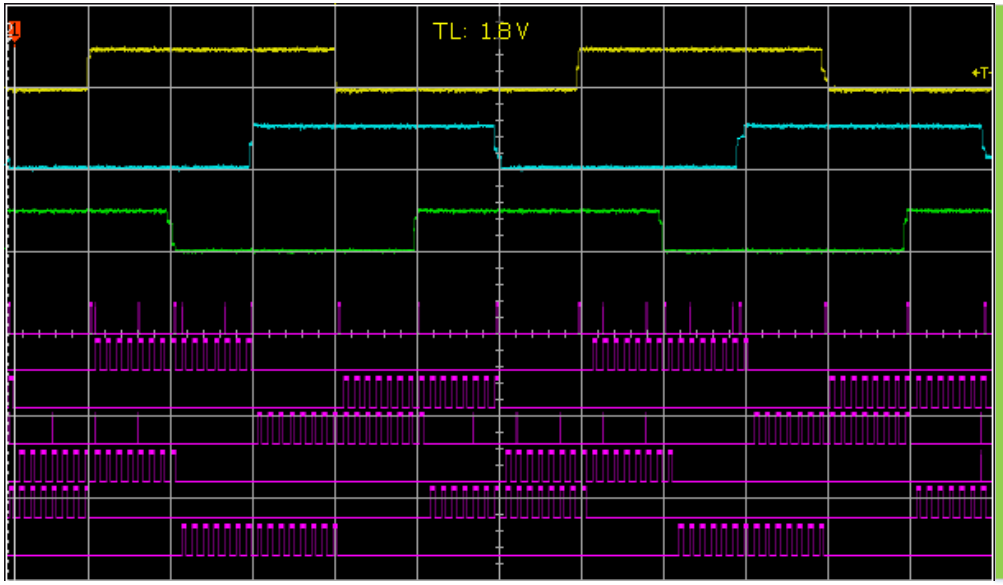
Séquenceur moteur BLDC

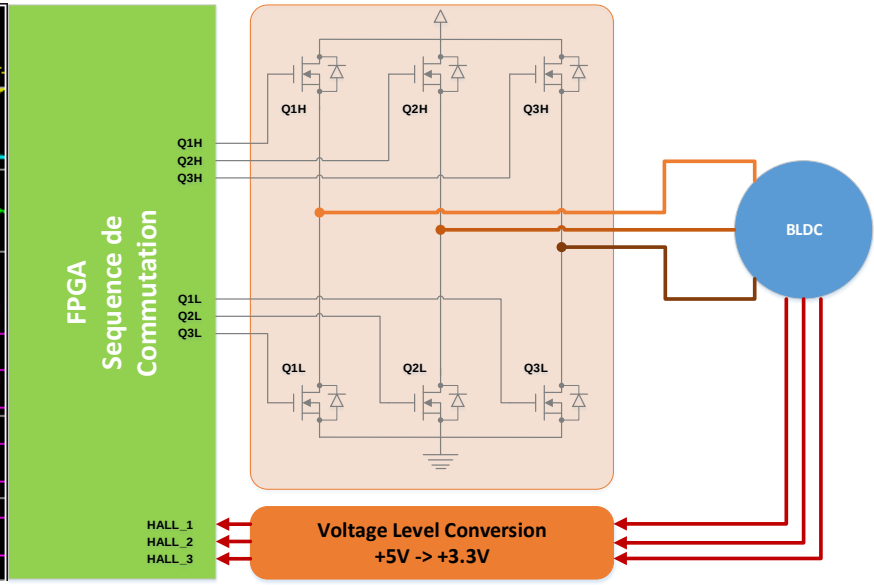


- ❑ 3 entrées – Capteurs Effet Hall – LVCMOS +3.3V
- ❑ 6 sorties – Moteurs 3 Phases
- ❑ PWM (Pulse Modulation Width)

Exemple de séquence de commutation
Sens de rotation horaire

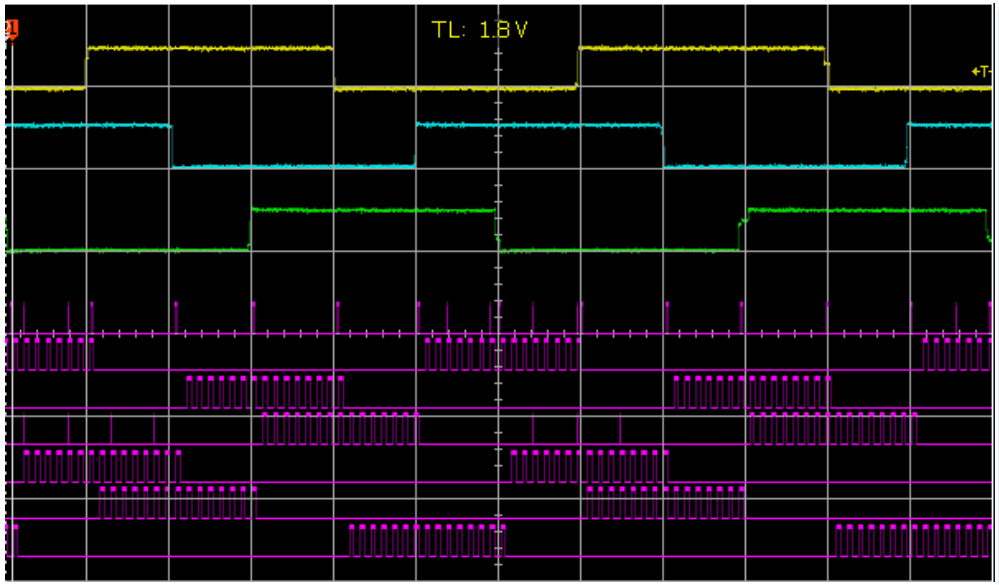
	Hall sensors			Switchs						Phases			Windings		
Phase	H3	H2	H1	Q1L	Q1H	Q2L	Q2H	Q3L	Q3H	P1	P2	P3	V_{1-2}	V_{2-3}	V_{3-1}
I	1	0	1	0	1	1	0	0	0	$+V_m$	Gnd	NC	$-V_m$	-	-
II	0	0	1	0	1	0	0	1	0	$+V_m$	NC	Gnd	-	-	$+V_m$
III	0	1	1	0	0	0	1	1	0	NC	$+V_m$	Gnd	-	$-V_m$	-
IV	0	1	0	1	0	0	1	0	0	Gnd	$+V_m$	NC	$+V_m$	-	-
V	1	1	0	1	0	0	0	0	1	Gnd	NC	$+V_m$	-	-	$-V_m$
VI	1	0	0	0	0	1	0	0	1	NC	Gnd	$+V_m$	-	$+V_m$	-

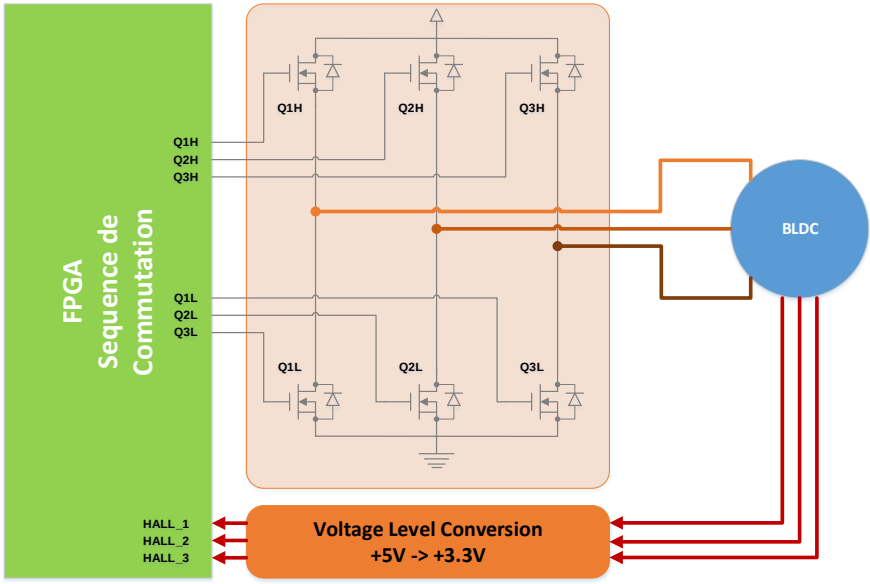




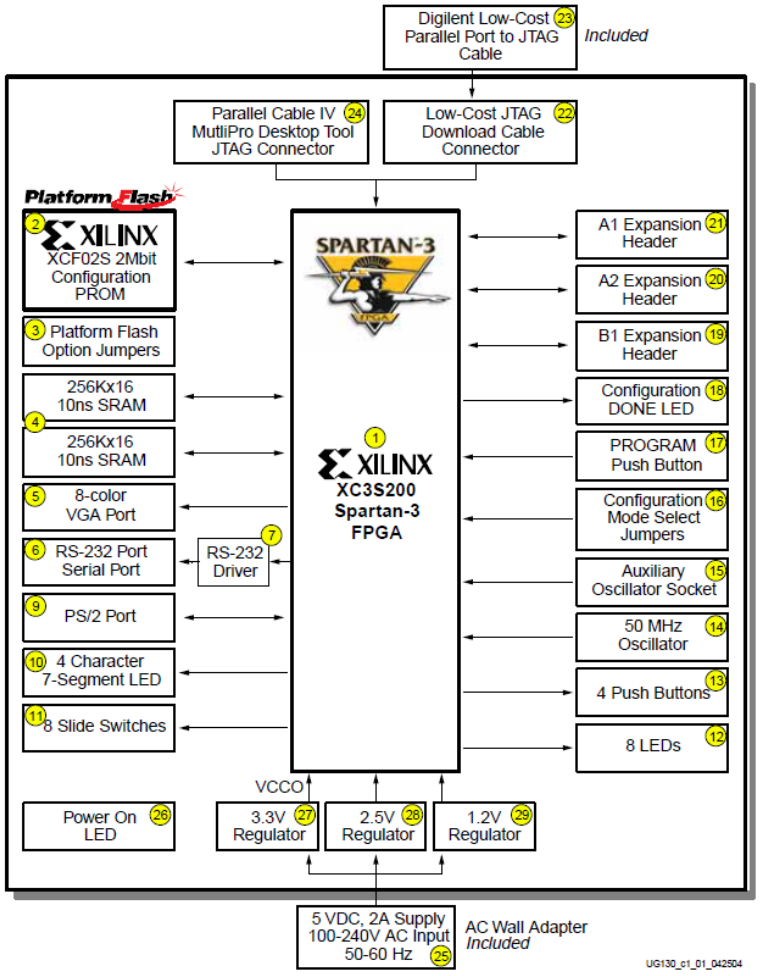
Exemple de séquence de commutation
Sens de rotation anti-horaire

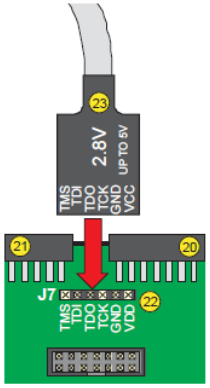
	Hall sensors			Switchs						Phases			Windings		
Phase	H3	H2	H1	Q1L	Q1H	Q2L	Q2H	Q3L	Q3H	P1	P2	P3	V_{1-2}	V_{2-3}	V_{3-1}
VI	1	0	0	0	0	0	1	1	0	NC	$+V_m$	Gnd	-	$-V_m$	-
V	1	1	0	0	1	0	0	1	0	$+V_m$	NC	Gnd	-	-	$+V_m$
IV	0	1	0	0	1	1	0	0	0	$+V_m$	Gnd	NC	$-V_m$	-	-
III	0	1	1	0	0	1	0	0	1	NC	Gnd	$+V_m$	-	$+V_m$	-
II	0	0	1	1	0	0	0	0	1	Gnd	NC	$+V_m$	-	-	$-V_m$
I	1	0	1	1	0	0	1	0	0	Gnd	$+V_m$	NC	$+V_m$	-	-

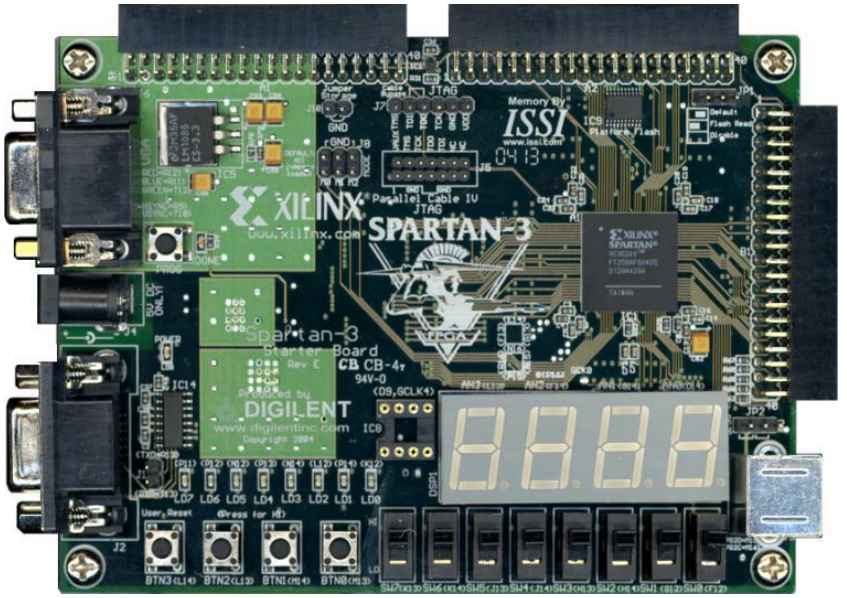




Carte d'évaluation SPARTAN-3

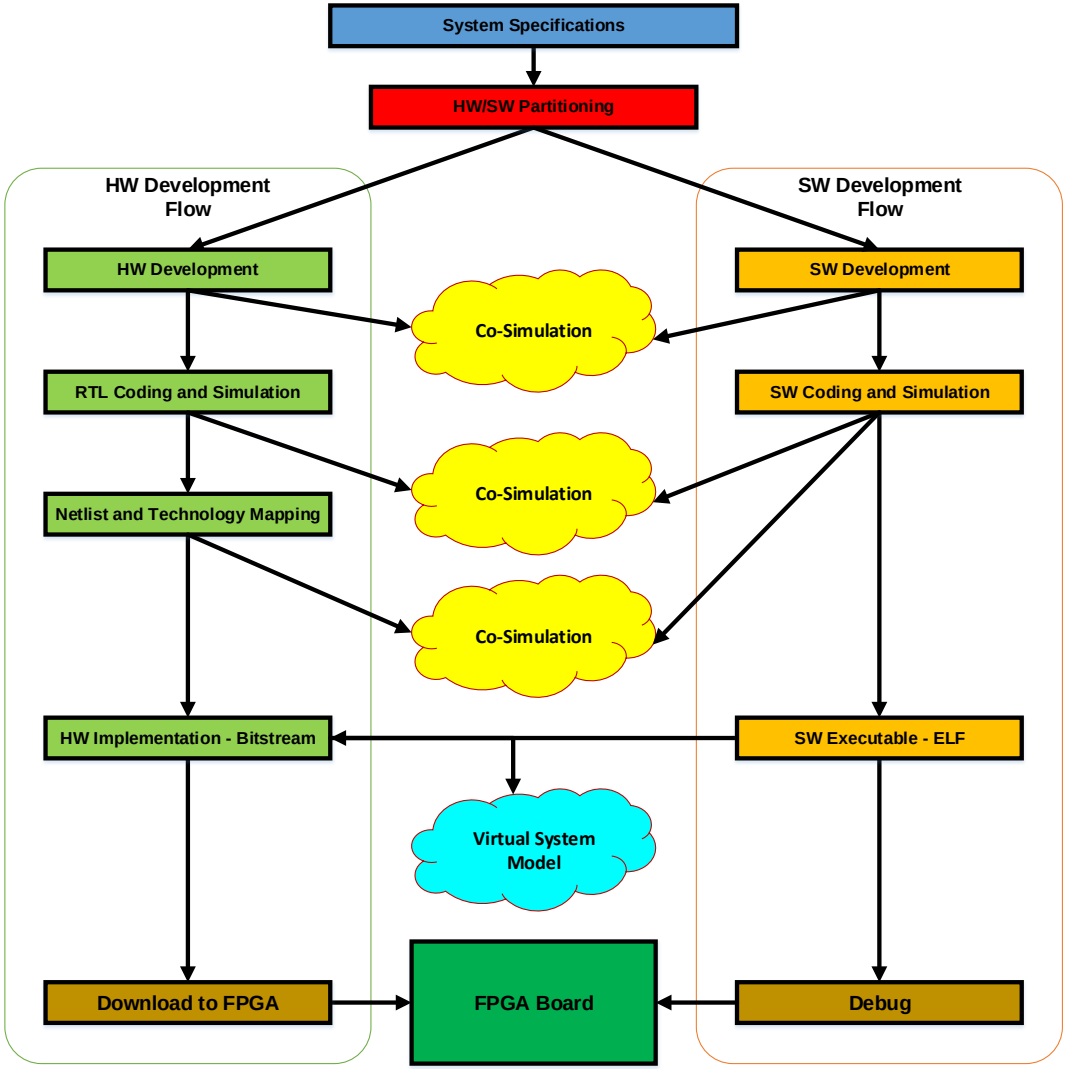






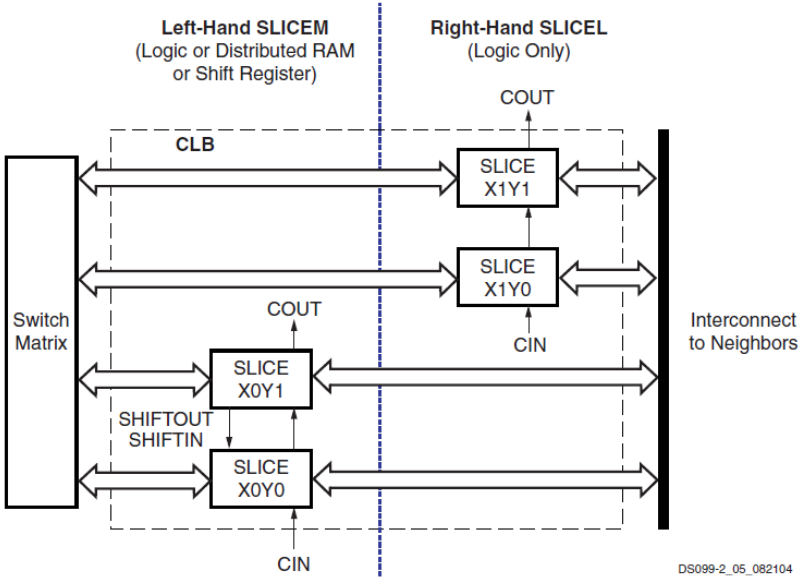
- Spartan-3 XC3S200FT256
- Boutons Switch, LED
- JTAG, Mémoire FLASH (XCF02S 2Mbit)
- RS-232 – UART – TTL Pas besoin de conversion
- IO Expansion Header LVSMOS +3.3V
  - Commande Moteur
    - 2x3 IO Effet Hall – Conversion +5V/+3.3V
    - 2x6 IO Phases Moteurs
  - Communication SPI – FPGA/STM32
    - 4x IO (SSEL, SCLK, MOSI, MISO)

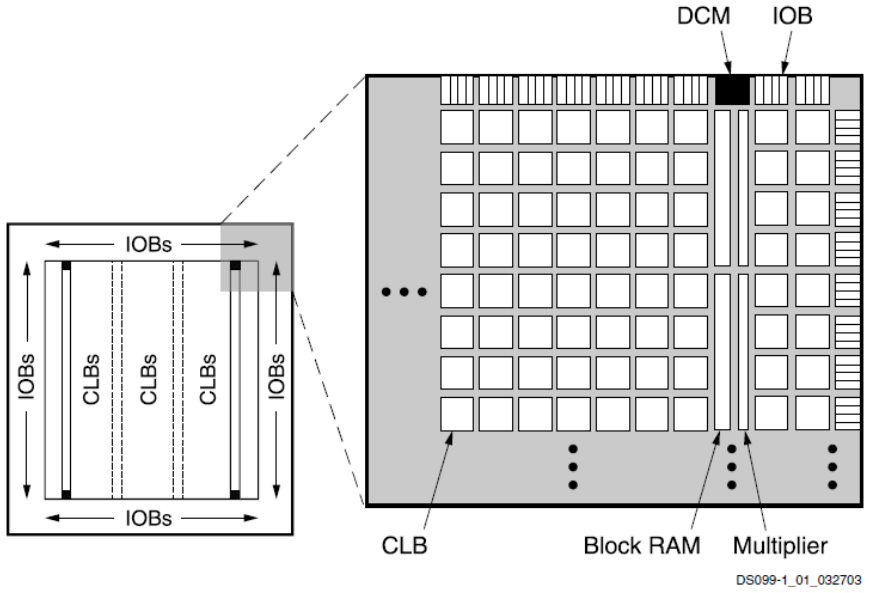
Conception conjointe HW/SW

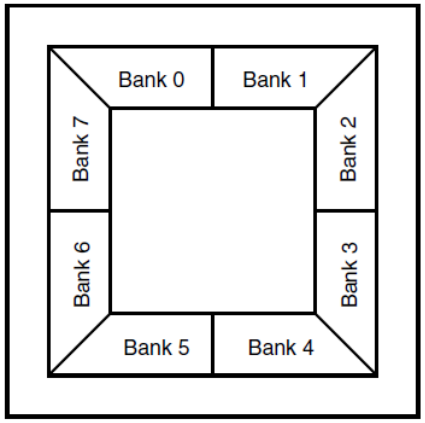


SPARTAN-3 XCS200FT256

- 200,000-gate
- Package BGA 256-ball (XC3S200FT256)
- 4,320 logic cell equivalents
- 12x 18K-bit block RAMs (216K bits)
- 12x 18x18 hardware multipliers
- 4x Digital Clock Managers (DCMs)
- Up to 173 user-defined I/O signals

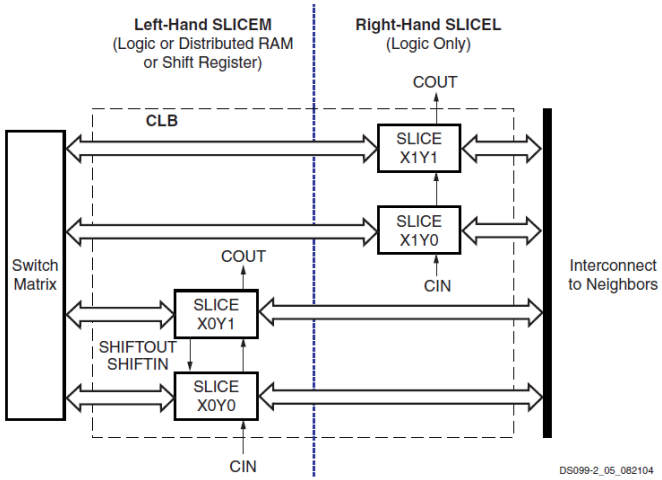






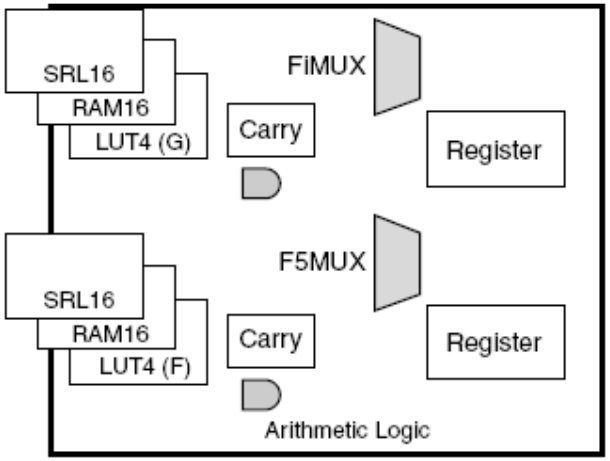
SPARTAN-3 – Slices

- SRL16 = 16-bit shift register
- RAM16 = 16-bit RAM (16x1 bit memory)
- LUT4 = four-bit lookup table (16x1 bit memory)
- SLICEM = slice that can be memory or logic
- SLICEL = slice that can only be logic



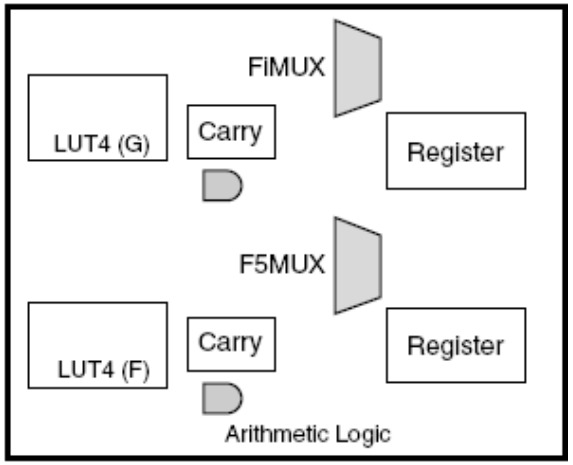
SLICEM

Slice that can be memory or logic



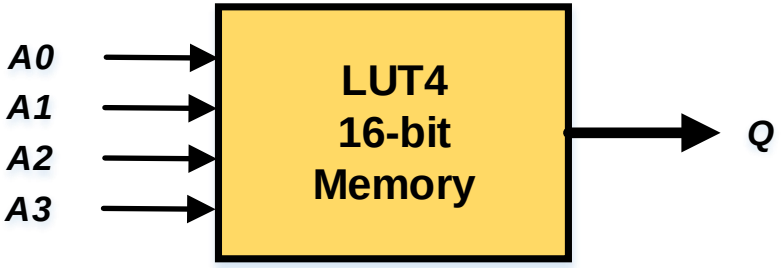
SLICEL

Slice that can only be logic

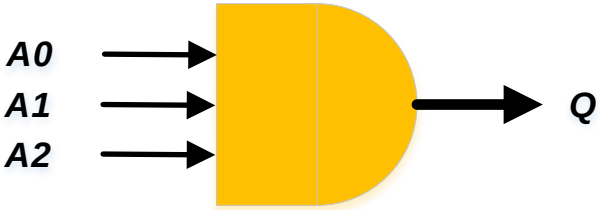


SPARTAN-3 – LUT4

- LUT4 = four-bit lookup table (16x1 bit memory)
- Mémoire RAM : 4-bit input and 1-bit output
- Implémente des fonctions logiques à 4 entrées



- Exemple : Fonction logique ET (AND) à 3-entrées
- Seulement les entrée A0, A1 et A2 sont utilisées

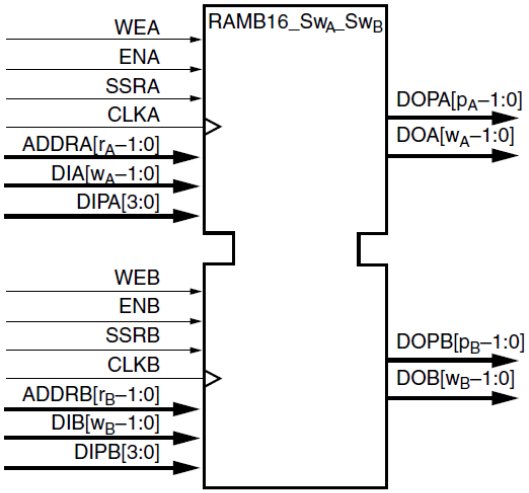


Q <= A0 and A1 and A2

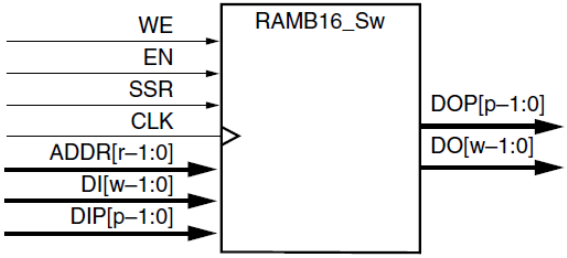
A0	A1	A2	A3	Q
0	0	0	x	0
0	0	1	x	0
0	1	0	x	0
0	1	1	x	0
1	0	0	x	0
1	0	1	x	0
1	1	0	x	0
1	1	1	x	1

SPARTAN-3 – BRAM & Multipliers

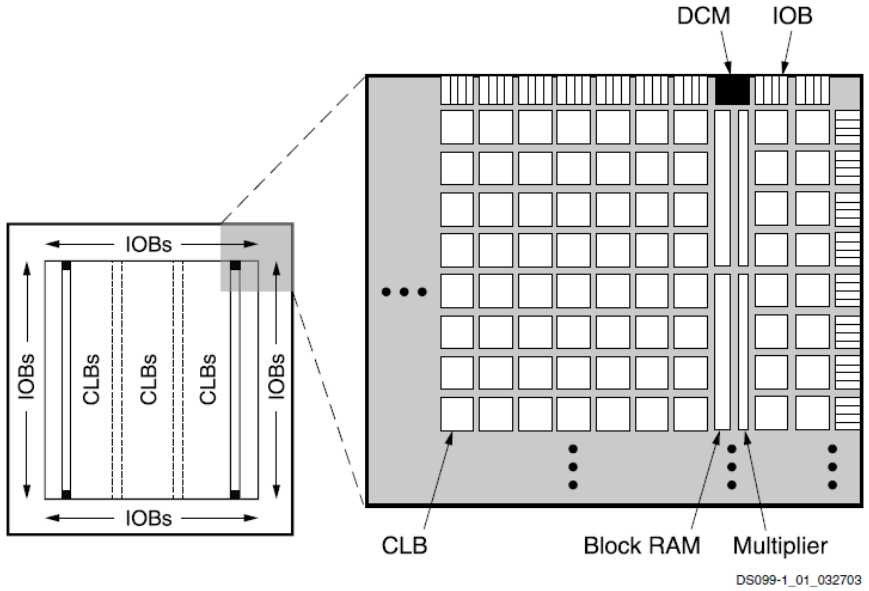
Block RAM Primitives



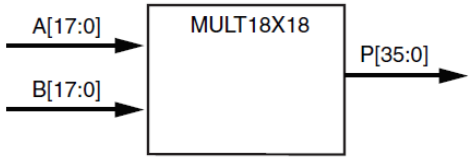
(a) Dual-Port



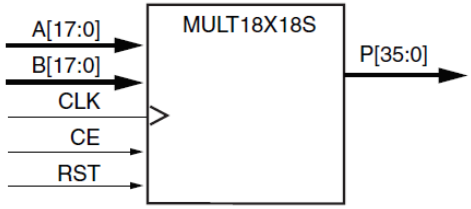
(b) Single-Port



Embedded Multiplier Primitives



(a) Asynchronous 18-bit Multiplier



(b) 18-bit Multiplier with Register

SPARTAN-3 XCS200FT256 – Foot Print

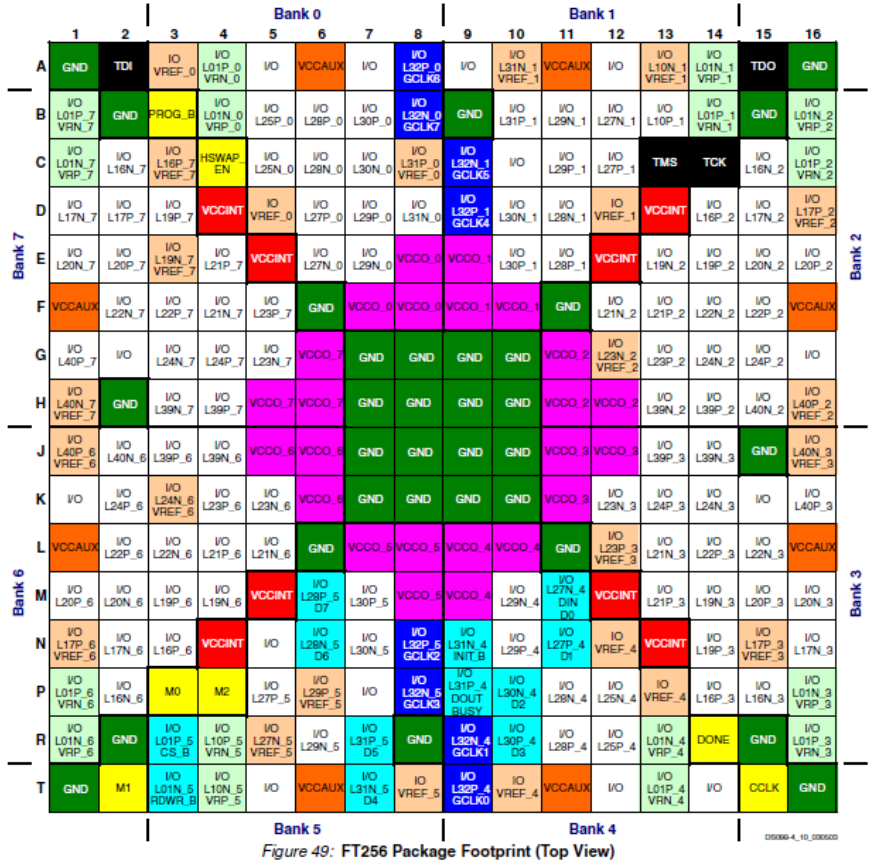


Figure 49: **FT256 Package Footprint (Top View)**

113	**I/O:** Unrestricted, general-purpose user I/O	12	**DUAL:** Configuration pin, then possible user I/O	24	**VREF:** User I/O or input voltage reference for bank
16	**DCI:** User I/O or reference resistor input for bank	8	**GCLK:** User I/O or global clock buffer input	24	**VCCO:** Output voltage supply for bank
7	**CONFIG:** Dedicated configuration pins	4	**JTAG:** Dedicated JTAG port pins	8	**VCCINT:** Internal core voltage supply (+1.2V)
0	**N.C.:** No unconnected pins in this package	32	**GND:** Ground	8	**VCCAUX:** Auxiliary voltage supply (+2.5V)

- Co-design Matériel/Logiciel
- Conception PCB
  - Affectation des I/O
  - Contrainte de niveau de tension des I/O
  - Optimisation du placement et le routage
- Conception HDL
  - Affectation des I/O
  - Définition du UCF
  - Placement et routage

UCF – SPARTAN-3

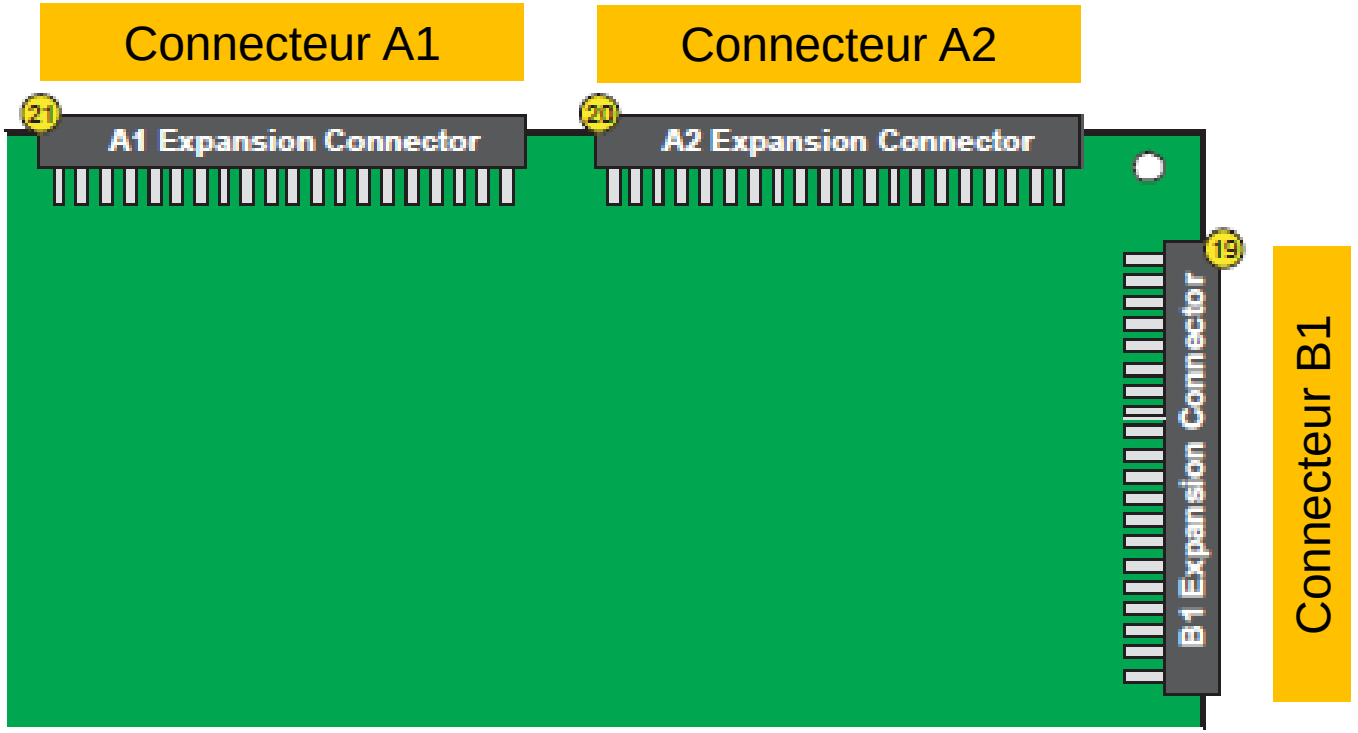
```
#Clock sources
#On-board 50 MHz crystal oscillator
NET "CLK"   LOC = "T9"  |  IOSTANDARD = LVCMOS33 ;

#Discrete LEDs
NET "LEDS<0>"   LOC = "K12"  |  IOSTANDARD = LVCMOS33;
NET "LEDS<1>"   LOC = "P14"  |  IOSTANDARD = LVCMOS33;
NET "LEDS<2>"   LOC = "L12"  |  IOSTANDARD = LVCMOS33;
NET "LEDS<3>"   LOC = "N14"  |  IOSTANDARD = LVCMOS33;
NET "LEDS<4>"   LOC = "P13"  |  IOSTANDARD = LVCMOS33;
NET "LEDS<5>"   LOC = "N12"  |  IOSTANDARD = LVCMOS33;
NET "LEDS<6>"   LOC = "P12"  |  IOSTANDARD = LVCMOS33;
NET "LEDS<7>"   LOC = "P11"  |  IOSTANDARD = LVCMOS33;

#Slide switches
NET "SW<0>"   LOC = "F12"   |  IOSTANDARD = LVCMOS33 ;
NET "SW<1>"   LOC = "G12"  |  IOSTANDARD = LVCMOS33 ;
NET "SW<2>"   LOC = "H14"  |  IOSTANDARD = LVCMOS33 ;
NET "SW<3>"   LOC = "H13"  |  IOSTANDARD = LVCMOS33 ;
NET "SW<4>"   LOC = "J14"   |  IOSTANDARD = LVCMOS33 ;
NET "SW<5>"   LOC = "J13"   |  IOSTANDARD = LVCMOS33 ;
NET "SW<6>"   LOC = "K14"  |  IOSTANDARD = LVCMOS33 ;
NET "SW<7>"   LOC = "K13"  |  IOSTANDARD = LVCMOS33 ;
```

UCF – SPARTAN-3

```
#UART RS-232 Serial Port
NET "TXD"   LOC = "R13"  | IOSTANDARD = LVCMOS33;
NET "RXD"    LOC = "T13"  | IOSTANDARD = LVCMOS33;

#Auxilary UART RS-232 Serial Port
#NET "TXDA"   LOC = "T14"  | IOSTANDARD = LVCMOS33  |  PULLDOWN ;
#NET "RXDA"    LOC = "N10"  | IOSTANDARD = LVCMOS33  |  PULLDOWN ;

#Four-Digit, Seven-Segment LED Display
NET "SEG<0>"          LOC = "E14"  | IOSTANDARD = LVCMOS33;          # Segment A
NET "SEG<1>"          LOC = "G13" | IOSTANDARD = LVCMOS33;          # Segment B
NET "SEG<2>"          LOC = "N15" | IOSTANDARD = LVCMOS33;          # Segment C
NET "SEG<3>"          LOC = "P15"  | IOSTANDARD = LVCMOS33;          # Segment D
NET "SEG<4>"          LOC = "R16"  | IOSTANDARD = LVCMOS33;          # Segment E
NET "SEG<5>"          LOC = "F13"  | IOSTANDARD = LVCMOS33;          # Segment F
NET "SEG<6>"          LOC = "N16" | IOSTANDARD = LVCMOS33;          # Segment G
NET "SEG<7>"          LOC = "P16"  | IOSTANDARD = LVCMOS33;          # DP

#Digit Enable (Anode Control) Signals(Active Low)
NET "ANODE<0>"        LOC = "D14" | IOSTANDARD = LVCMOS33;          #AN0
NET "ANODE<1>"        LOC = "G14" | IOSTANDARD = LVCMOS33;          #AN1
NET "ANODE<2>"        LOC = "F14"  | IOSTANDARD = LVCMOS33;          #AN2
NET "ANODE<3>"        LOC = "E13"  | IOSTANDARD = LVCMOS33;          #AN3
```

UCF – SPARTAN-3 – Expansion Connectors

- 6x Input – Capteurs Effet-Hall – Moteur 1 et 2
- 12x Sorties – Switch MOSFET - Phases Moteurs 1 et 2
- 3x Entrées SPI – SSEL, SCLK, MOSI
- 1x Sortie SPI - MISO



UCF – SPARTAN-3 – Expansion Connectors

Connecteur A1

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(N8)	ADR0
DB0	(N7) SRAM IC10 IO0	5	6	(L5) SRAM A0	ADR1
DB1	(T8) SRAM IC10 IO1	7	8	(N3) SRAM A1	ADR2
DB2	(R6) SRAM IC10 IO2	9	10	(M4) SRAM A2	ADR3
DB3	(T5) SRAM IC10 IO3	11	12	(M3) SRAM A3	ADR4
DB4	(R5) SRAM IC10 IO4	13	14	(L4) SRAM A4	ADR5
DB5	(C2) SRAM IC10 IO5	15	16	(G3) SRAM WE#	WE
DB6	(C1) SRAM IC10 IO6	17	18	(K4) SRAM OE#	OE
DB7	(B1) SRAM IC10 IO7	19	20	(P9) FPGA DOUT/BUSY	CSA
LSBCLK	(M7)	21	22	(M10)	MA1-DB0
MA1-DB1	(F3) SRAM A6	23	24	(G4) SRAM A5	MA1-DB2
MA1-DB3	(E3) SRAM A8	25	26	(F4) SRAM A7	MA1-DB4
MA1-DB5	(G5) SRAM A10	27	28	(E4) SRAM A9	MA1-DB6
MA1-DB7	(H4) SRAM A12	29	30	(H3) SRAM A11	MA1-ASTB
MA1-DSTB	(J3) SRAM A14	31	32	(J4) SRAM A13	MA1-WRITE
MA1-WAIT	(K5) SRAM A16	33	34	(K3) SRAM A15	MA1-RESET
MA1-INT	(L3) SRAM A17	35	36	JTAG Isolation	JTAG Isolation
TMS	(C13) FPGA JTAG TMS	37	38	(C14) FPGA JTAG TCK	TCK
TDO-ROM	Platform Flash JTAG TDO	39	40	Header J7, pin 3	TDO-A

Connecteur A2

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(E6)	PA-IO1
PA-IO2	(D5)	5	6	(C5)	PA-IO3
PA-IO4	(D6)	7	8	(C6)	PA-IO5
PA-IO6	(E7)	9	10	(C7)	PA-IO7
PA-IO8	(D7)	11	12	(C8)	PA-IO9
PA-IO10	(D8)	13	14	(C9)	PA-IO11
PA-IO12	(D10)	15	16	(A3)	PA-IO13
PA-IO14	(B4)	17	18	(A4)	PA-IO15
PA-IO16	(B5)	19	20	(A5)	PA-IO17
PA-IO18	(B6)	21	22	(B7)	MA2-DB0
MA2-DB1	(A7)	23	24	(B8)	MA2-DB2
MA2-DB3	(A8)	25	26	(A9)	MA2-DB4
MA2-DB5	(B10)	27	28	(A10)	MA2-DB6
MA2-DB7	(B11)	29	30	(B12)	MA2-ASTB
MA2-DSTB	(A12)	31	32	(B13)	MA2-WRITE
MA2-WAIT	(A13)	33	34	(B14)	MA2-RESET
MA2-INT/GCK4	(D9) Oscillator socket	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

UCF – SPARTAN-3 – Expansion Connectors

Connecteur B1

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(C10)	PB-ADR0
PB-DB0	(T3) FPGA RD_WR_B config	5	6	(E10)	PB-ADR1
PB-DB1	(N11) FPGA D1 config	7	8	(C11)	PB-ADR2
PB-DB2	(P10) FPGA D2 config	9	10	(D11)	PB-ADR3
PB-DB3	(R10) FPGA D3 config	11	12	(C12)	PB-ADR4
PB-DB4	(T7) FPGA D4 config	13	14	(D12)	PB-ADR5
PB-DB5	(R7) FPGA D5 config	15	16	(E11)	PB-WE
PB-DB6	(N6) FPGA D6 config	17	18	(B16)	PB-OE
PB-DB7	(M6) FPGA D7 config	19	20	(R3) FPGA CS_B config	PB-CS
PB-CLK	(C15)	21	22	(C16)	MB1-DB0
MB1-DB1	(D15)	23	24	(D16)	MB1-DB2
MB1-DB3	(E15)	25	26	(E16)	MB1-DB4
MB1-DB5	(F15)	27	28	(G15)	MB1-DB6
MB1-DB7	(G16)	29	30	(H15)	MB1-ASTB
MB1-DSTB	(H16)	31	32	(J16)	MB1-WRITE
MB1-WAIT	(K16)	33	34	(K15)	MB1-RESET
MB1-INT	(L15)	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

Partie #1 : Rappel VHDL

I. Séquenceur
   1. Exemple de séquence de commutation d'un moteur BLDC
   2. Block et I/O
II. Plateforme FPGA
   I. Kit d'évaluation Spartan-3
   II. Architecture FPGA SPARTAN-3
III. Rappel VHDL
   1. Combinatoire
   2. Séquentiel
   3. Concurrentiel
IV. Prise en main ISE et ISim
   1. Exemple #1: Switch - Afficheur 7-segment
   2. Exemple #2: Compteur et Diviseur de fréquence
      a. Compteur – Afficheur 7-segment
      b. Horloge 1MHz dérivée à partir de la référence 50 MHz
   3. Exemple #3: Implémentation de module PWM

Opérateurs Logiques

Opérateur	Description
AND	“ET” Logique
OR	“OU” Logique
XOR	“OU eXclusif” Logique
NAND	“NON ET” Logique
NOR	“NON OU” Logique
XNOR	“NON OU eXclusif” Logique
NOT	“NON” Logique ou Inverseur

Opérateurs Arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*****	Multiplication
/	Division
&	Concatination: B <= “0101”; C <= “10” A <= B & C; -- A = “010110”
mod	Modulo : A mod B = A – B * N , où N est un entier
rem	Reste de la división : A rem B = A – (A/B) * B
******	exposant
abs	Valeur absolue

Opérateurs de comparaison

Opérateur	Description
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur
>=	Supérieur ou égal à
=	Exactement égale à
/=	Différent

Opérateurs de décalage

Opérateur	Description
sll	Décalage logique à gauche
slr	Décalage logique à droit
rol	Rotation à gauche
ror	Rotation à droit
Librairie numeric_std	
shift_left	Décalage logique à gauche
shift_right	Décalage logique à droit

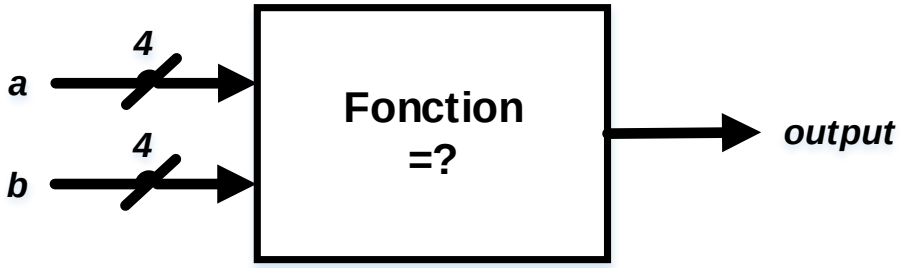
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;   -- unsigned/signed & shift_left/shift_right

  signal slv4_data      : std_logic_vector(3 downto 0)  := "1000";
  signal uv4_data_sl    : unsigned(3 downto 0)          := "0000";
  signal uv4_data_sr    : unsigned(3 downto 0)          := "0000";
  signal sv4_data_sl    : signed(3 downto 0)            := "0000";
  signal sv4_data_sr    : signed(3 downto 0)            := "0000";

begin
    -- Décalage à gauche
    uv4_data_sl     <= shift_left(unsigned(slv4_data), 1);
    sv4_data_sl     <= shift_left(signed(slv4_data), 1);

    -- Décalage à droit
    uv4_data_sr     <= shift_right(unsigned(slv4_data), 2);
    sv4_data_sr     <= shift_right(signed(slv4_data), 2);
```

Opérateurs – ordre de priorité

- Ordre de priorité décroissant

Opérateurs misc	**, abs, not
Opérateur de multiplication	*, /, mod, rem
Signes	+, -
Opérateurs d'addition	+, -, &
Opérateurs de décalage	sll, srl, sla, sra, rol, ror
Opérateurs de comparaison	=, /=, <, <=, >, >=
Opérateurs logiques	and, or, nand, nor, xor, xnor

Entity - VHDL



- Entity : Déclaration des ports d'E/S du module
- Architecture : Implémente la fonction du module
- Exemple: comparaison de eux vecteurs de 4-bits.

```
Library ieee;
Use ieee.std_logic_1164.all;
```

```
entity compare is
    port (          a       : in    std_logic_vector(3 downto 0);
                    b       : in    std_logic_vector(3 downto 0);
                    output  : out   std_logic
          );
end compare;
```

```
architecture  bahavioral of compare is
begin
                    output  <=  '1' when (a= b) else '0';
end behavioral;
```

Entity – Process

```
Library ieee;
Use ieee.std_logic_1164.all;
```

```
entity register4 is
    port (      clk     : in    std_logic_vector(3 downto 0);
                en      : in    std_logic_vector(3 downto 0);
                d       : in    std_logic_vector(3 downto 0);
                q       : out   std_logic _vector(3 downto 0)
          );
end register4;
```

- Déclaration des ports d'E/S

```
architecture  bahavioral of register4 is
signal dreg : std_logic_vector(3 downto 0);
Begin
        q <= dreg;
        proc : process(clk, en, d)
        begin
                if(rising_edge(clk) and en='1') then
                    dreg <= d;
                end if;
        end process proc;
end behavioral;
```

- Architecture
- Déclaration des signaux
- Déclaration de Process
- Liste de sensibilité
- Affectation des signaux et sorties

À l'extérieur d'un Process	À l'intérieur d'un Process
WHEN … ELSE	IF … ELSIF … ELSE … END IF
WITH … SELECT … WHEN	CASE … IS … WHEN … END CASE
	FOR … LOOP … END LOOP

Librairie NUMERIC_STD

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
```

Opérateurs arithmétiques

+	-	*	/	rem	mod
<	<=	>	>=	=	/=

Type op1	OP	Type op2
UNSIGNED	.	UNSIGNED
UNSIGNED	.	NATURAL
NATURAL	.	UNSIGNED
SIGNED	.	SIGNED
SIGNED	.	INTEGER
INTEGER	.	SIGNED

Opérateurs de décalage

sll	srl	rol	ror

Type op1	OP	Type op2
UNSIGNED	.	INTEGER
SIGNED	.	INTEGER

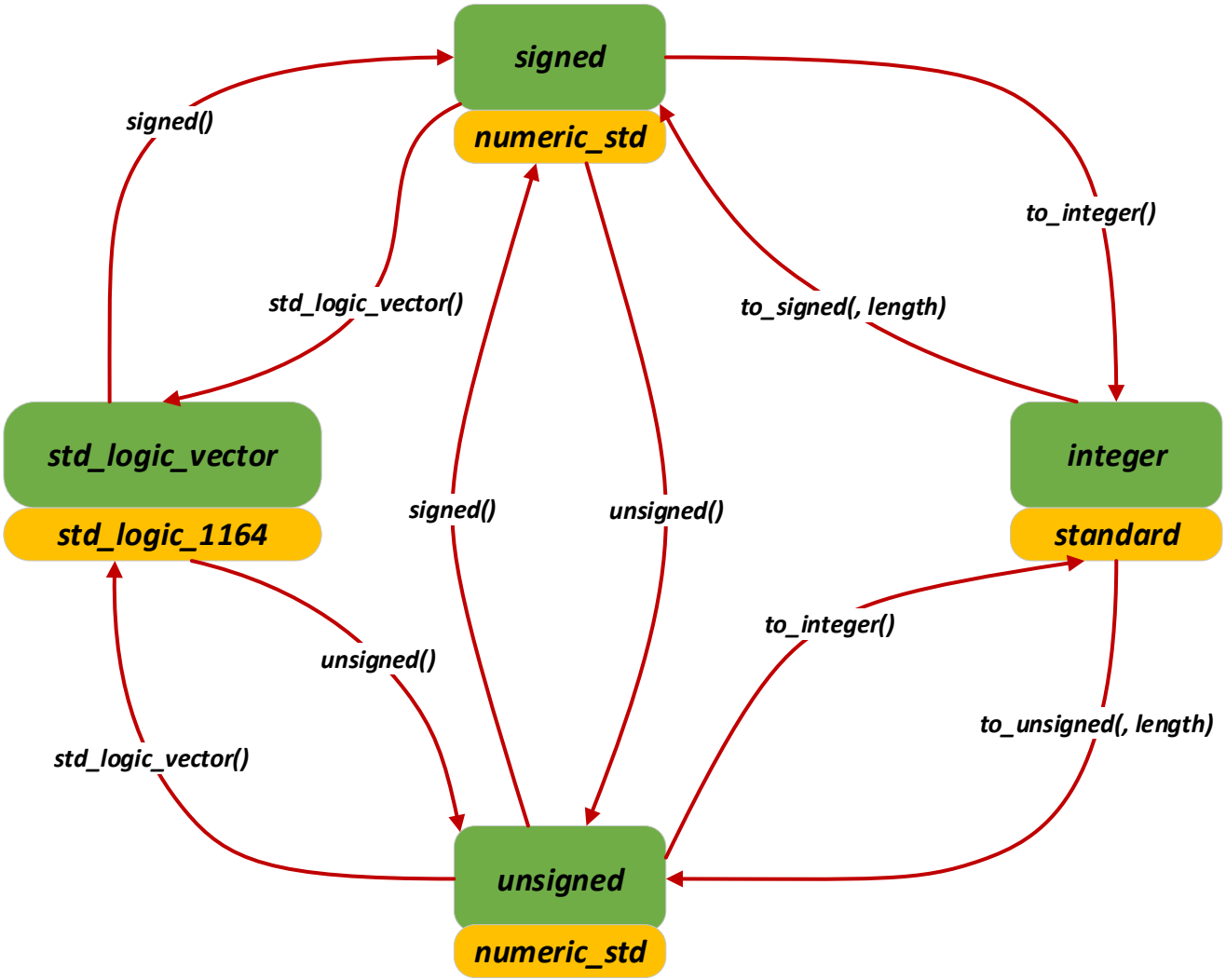
Opérateurs logiques

	and	or	xor
not	nand	nor	nxor

Type op1	OP	Type op2
UNSIGNED	.	UNSIGNED
SIGNED	.	SIGNED

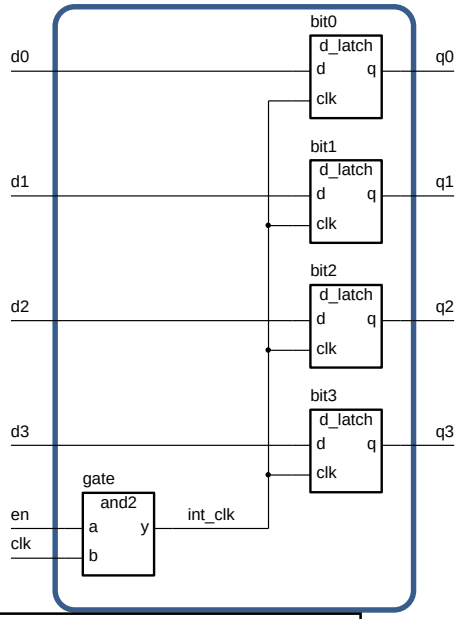
LISTES DE FONCTIONS		
Fonction	**Liste et types des arguments**	**Type retourné**
TO_INTEGER	(UNSIGNED)	return INTEGER
TO_INTEGER	(SIGNED)	return INTEGER
TO_UNSIGNED	(NATURAL, NATURAL)	return UNSIGNED
TO_SIGNED	(INTEGER, NATURAL)	return SIGNED
RESIZE	(UNSIGNED, NATURAL)	return UNSIGNED
RESIZE	(SIGNED, NATURAL)	return SIGNED

Conversions de type NUMERIC_STD



Entity – Structurelle

- Structure Hiérarchique
- Interconnexion de sous-modules
- Déclaration de signaux d'interconnexion des sous-modules
- Déclaration/Instanciation des sous-modules
- Liste d'interconnexion – Port Map



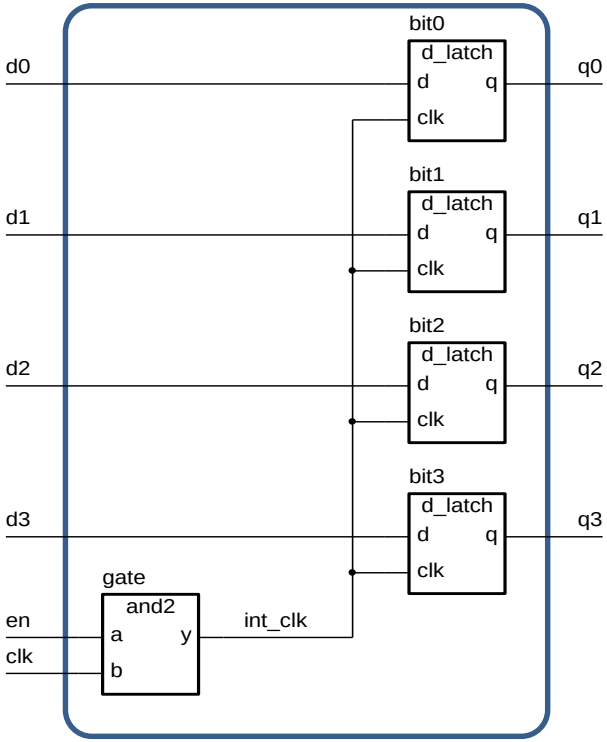

```
entity d_latch is
        port ( d, clk : in std_logic;  q : out std_logic );
end entity d_latch;

architecture basic of d_latch is
begin
        process (clk, d)
        begin
                if clk'event and ssclk = '1' then
                        q <= d after 2 ns;
                end if;
        end process;
end basic;
```

```
entity and2 is
        port ( a, b : in std_logic;  y : out std_logic );
end entity and2;

architecture basic of and2 is
begin
        process (a, b)
        begin
                y <= a and b after 2 ns;
        end process ;
end basic;
```

Entity – Structurelle

- Structure Hiérarchique
- Interconnexion de sous-modules
- Déclaration de signaux d'interconnexion des sous-modules
- Déclaration/Instanciation des sous-modules
- Liste d'interconnexion – Port Map

```
architecture struct of reg4 is
component d_latch
                port ( d, clk : in std_logic;  q : out std_logic );
end component;
component and2
                port ( a, b : in std_logic;  y : out std_logic );
end component;
signal int_clk : std_logic;
begin
bit0 : d_latch
                port map ( d0, int_clk, q0 );
bit1 : d_latch
                port map ( d1, int_clk, q1 );
bit2 : d_latch
                port map ( d2, int_clk, q2 );
bit3 : d_latch
                port map ( d3, int_clk, q3 );
gate : and2
                port map ( en, clk, int_clk );
end struct;
```



Instruction d'affectation de signaux

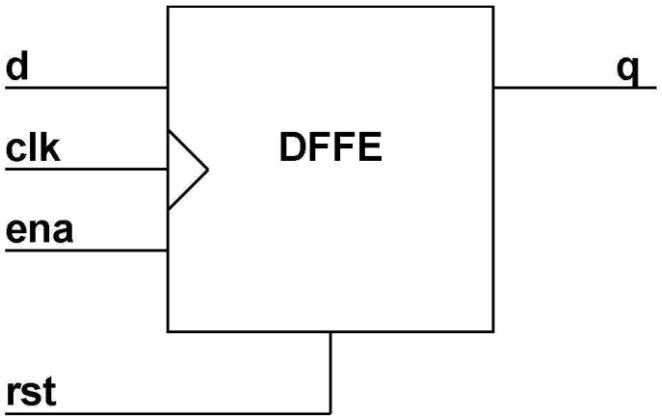
```
SIGNAL a, b, c : std_logic;
SIGNAL a_vect, b_vect, c_vect : std_logic_vector(7 DOWNTO 0);

-- Instruction d'affectation de signaux de manière concurrente
-- NB: les signaux a and a_vect sont produits de manière concurrente
a           <= b AND c;
a_vect      <= b_vect OR c_vect;



-- On peut affecter également des valeurs constantes aux signaux
a           <= '0';
b           <= '1';
c           <= 'Z';
a_vect      <= "00111010";  -- Affecter 0x3A à a_vect
b_vect      <= X"3A";       -- Affecter 0x3A à b_vect
c_vect      <= X"3" & X"A"; -- Affecter 0x3A à c_vect
```

Instruction d'affectation de signaux

```
SIGNAL a, b, c, d, e :std_logic;
SIGNAL a_vect :std_logic_vector(1 DOWNTO 0);
SIGNAL b_vect :std_logic_vector(2 DOWNTO 0);
SIGNAL c_vect, d_vect :std_logic_vector(7 DOWNTO 0);

-- Affectation conditionnelle WHEN ELSE
a <=    '0' WHEN a_vect = "00"  ELSE
        b   WHEN a_vect = "11"  ELSE
        c   WHEN d = '1'        ELSE
        '1';
-- Expression WITH SELECT - Vecteurs
-- NB: Les valeurs de sélection doivent être constantes
-- Concatination
b_vect <= d & a_vect;

WITH b_vect SELECT
a <=    '0' WHEN "000",
        b   WHEN "011",
        c   WHEN "1--", -- Attention à la synthèse du '-' (don't care)!!!
        '1' WHEN OTHERS;

-- Expression WITH SELECT - Vecteurs
WITH e SELECT
c_vect <=   "01010101"  WHEN '1',
            d_vect      WHEN OTHERS;
```

Exemple D Flip-Flop

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY dffe IS
    PORT(   rst :  IN std_logic;
            clk :  IN std_logic;
            ena :  IN std_logic;
            d   :  IN std_logic;
            q   :  OUT std_logic);
END dffe;

ARCHITECTURE behavioural OF dffe IS
BEGIN
    PROCESS (rst, clk)
    BEGIN
        IF (rst = '1') THEN
            q <= '0';
        ELSIF (clk'EVENT) AND (clk = '1') THEN
            IF (ena = '1') THEN
               q <= d;
            END IF;
        END IF;
    END PROCESS;
END behavioural;
```




```
(clk'EVENT) AND (clk = '1')      ≈      rising_edge(clk)
```

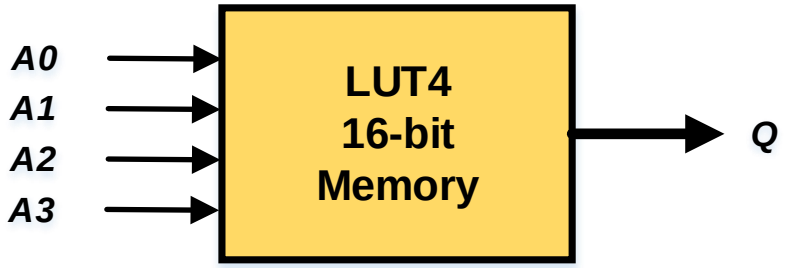
Partie #1 : ISE/VHDL
Prise en main ISE/ISim – Exermples

I. Séquenceur
   1. Exemple de séquence de commutation d'un moteur BLDC
   2. Block et I/O
II. Plateforme FPGA
   I. Kit d'évaluation Spartan-3
   II. Architecture FPGA SPARTAN-3
III. Rappel VHDL
   1. Combinatoire
   2. Séquentiel
   3. Concurrentiel

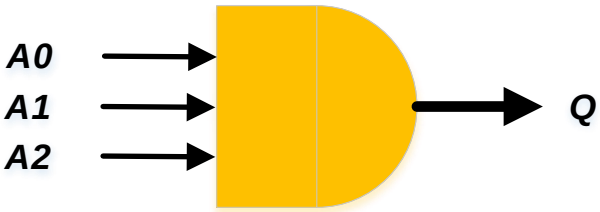
IV. Prise en main ISE et ISim
   1. Exemple #1: Switch - Afficheur 7-segment
   2. Exemple #2: Compteur et Diviseur de fréquence
      a. Compteur – Afficheur 7-segment
      b. Horloge 1MHz dérivée à partir de la référence 50 MHz
   3. Exemple #3: Implémentation de module PWM

Prise en main ISE/ISim

- Démarrer ISE
- Implémenter le circuit suivant : Fonction AND à 3 entrées
- Analyser le schéma technologique
- Créer un testbench pour ce bloc
- Simuler le design avec ISim,
- Valider le fonctionnement de l'implémentation



- Exemple : Fonction logique ET (AND) à 3-entrées
- Seulement les entrée A0, A1 et A2 sont utilisées



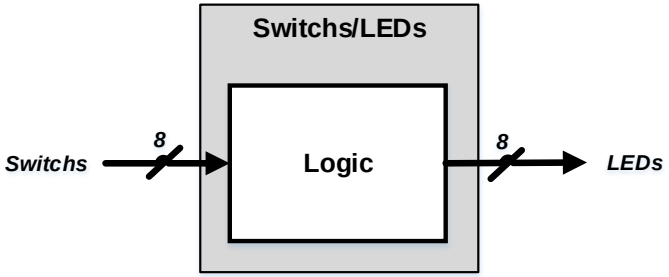
Q <= A0 and A1 and A2

A0	A1	A2	A3	Q
0	0	0	x	0
0	0	1	x	0
0	1	0	x	0
0	1	1	x	0
1	0	0	x	0
1	0	1	x	0
1	1	0	x	0
1	1	1	x	1

Exemple #1: Switch/LEDs – Afficheur 7-segments

Switches

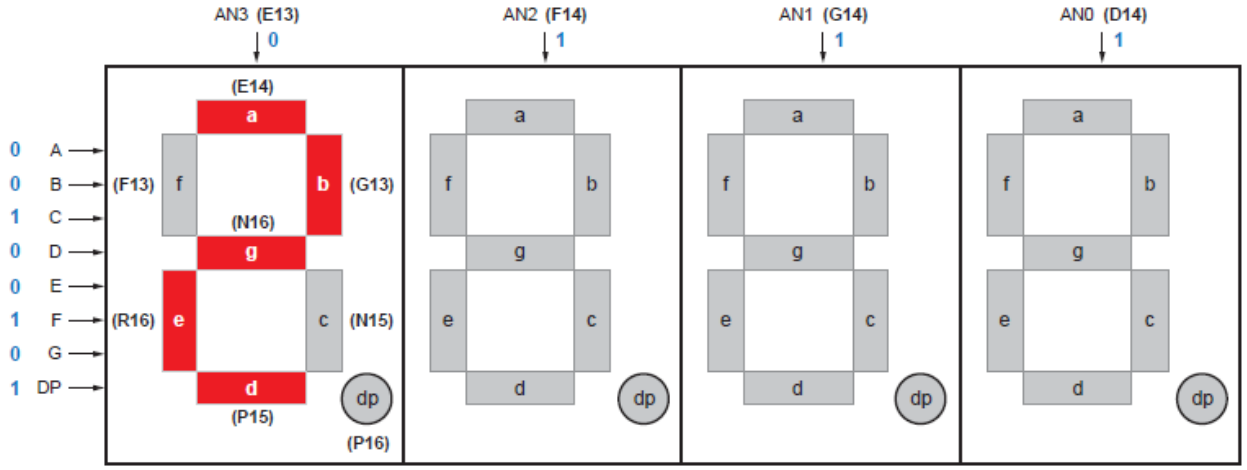
Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12



LEDs

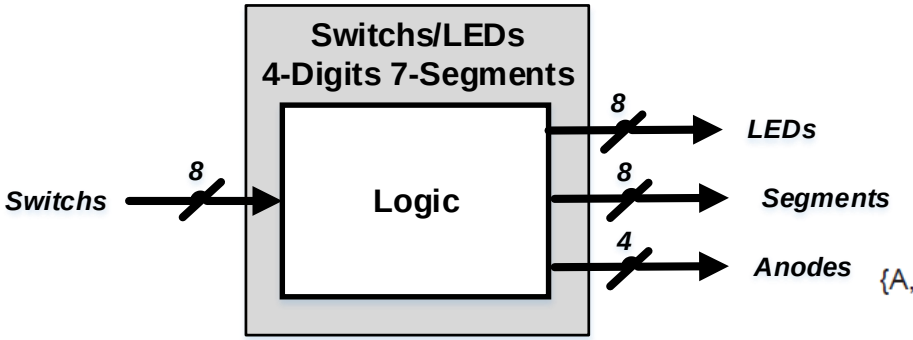
LED	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

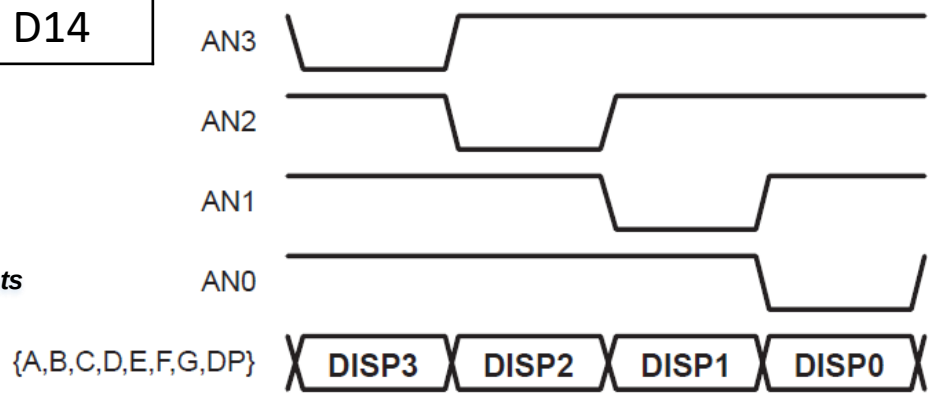
Exemple #1: Switch/LEDs – Afficheur 7-segments



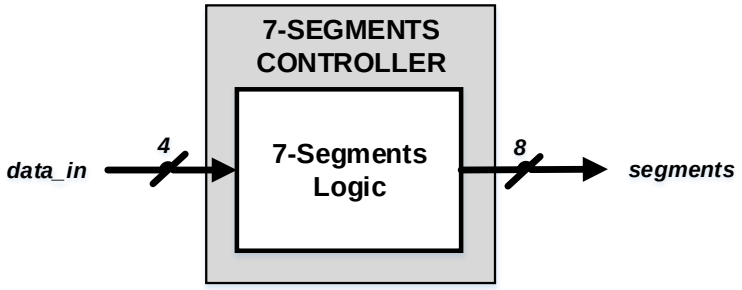
Segment	FPGA Pin
A	E14
B	G13
C	N15
D	P15
E	R16
F	F13
G	N16
DP	P16

Anode Control	AN3	AN2	AN1	AN0
FPGA Pin	E13	F14	G14	D14





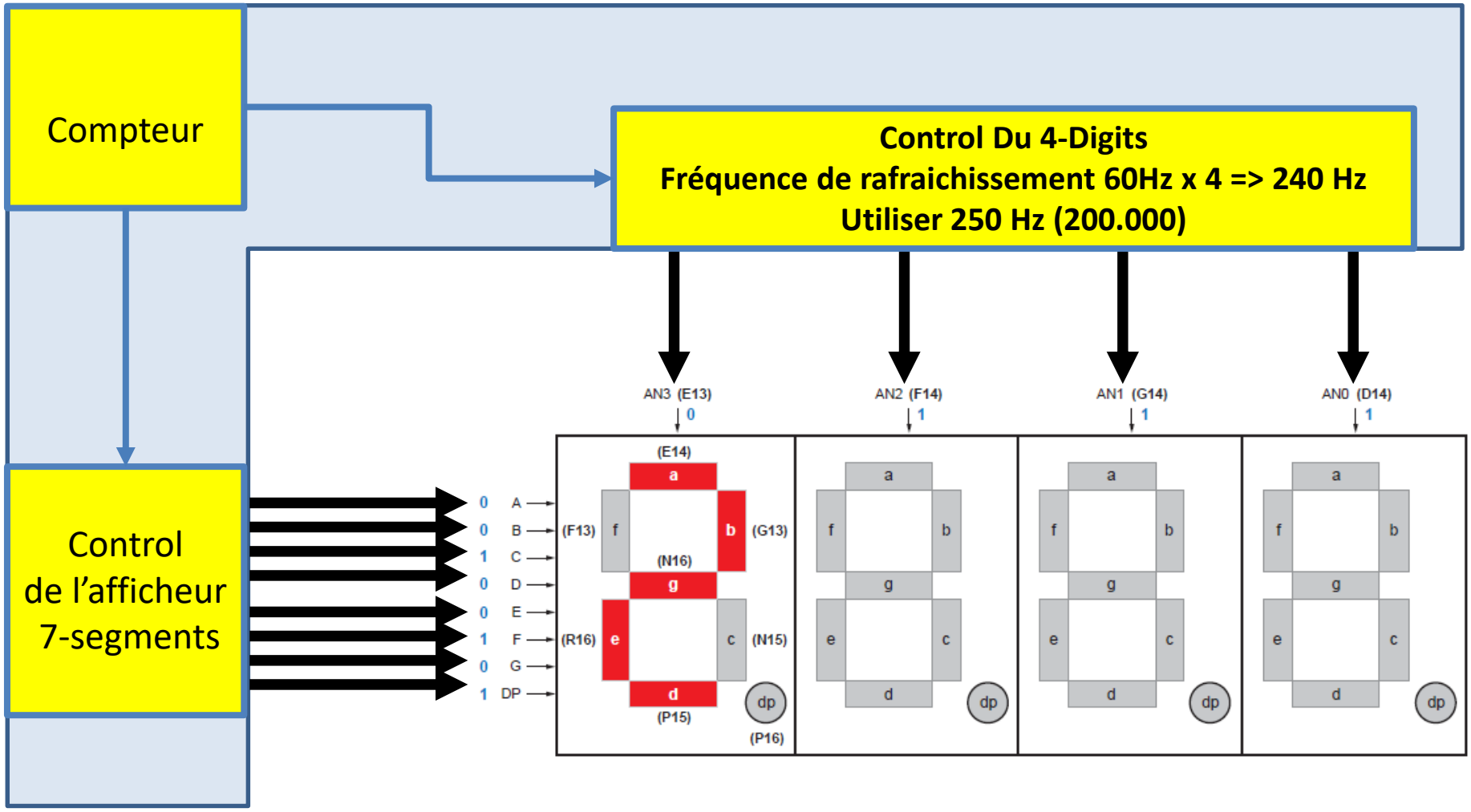
Exemple #1: Afficheur 7-segments




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY seven_segment IS
    PORT( data_in    : IN  std_logic_vector(3 DOWNTO 0);
          segments   : OUT std_logic_vector(7 DOWNTO 0));
END seven_segegment;

ARCHITECTURE behavioural OF seven_segment IS
BEGIN
    WITH data_in SELECT
        segments <= "10000001" WHEN "0000", -- 0
                    "11001111" WHEN "0001", -- 1
                    "10010010" WHEN "0010", -- 2
                    "10000110" WHEN "0011", -- 3
                    "11001100" WHEN "0100", -- 4
                    "10100100" WHEN "0101", -- 5
                    "10100000" WHEN "0110", -- 6
                    "10001111" WHEN "0111", -- 7
                    "10000000" WHEN "1000", -- 8
                    "10000100" WHEN "1001", -- 9
                    "11111111" WHEN OTHERS;
END behavioural;
```

Character	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
b	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0

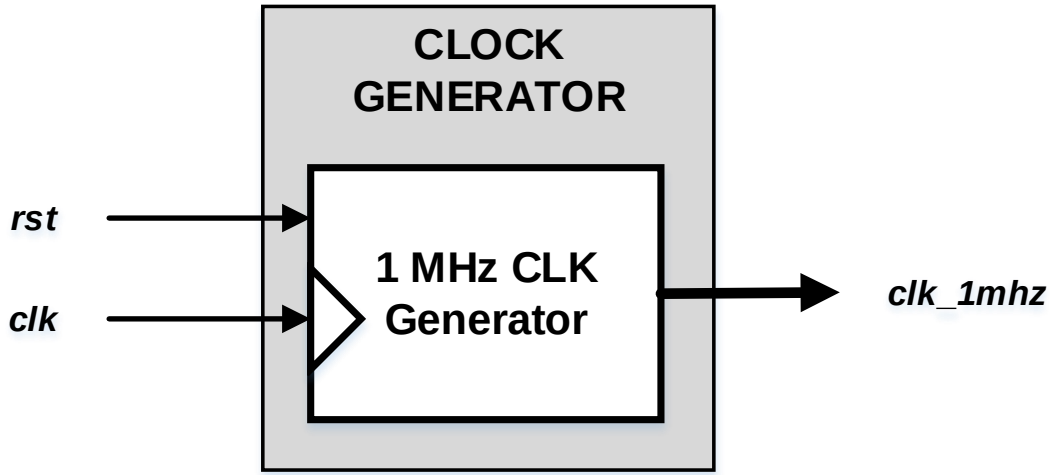
Exemple #2: Compteur – Afficheur 7-segments

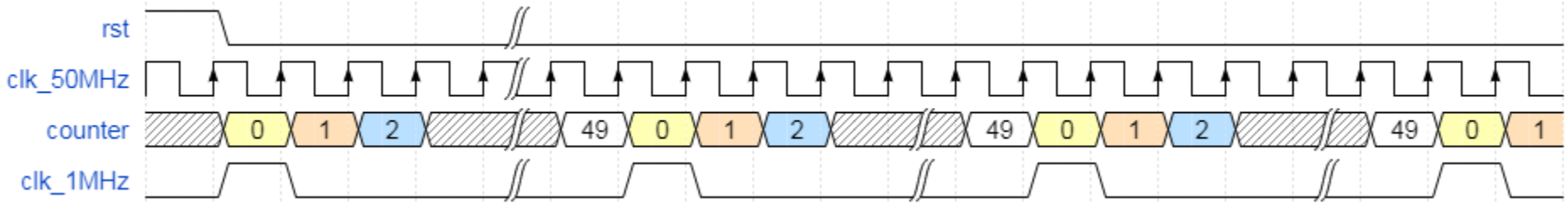


PWM

Exemple #2: Compteur – Horloge 1MHz

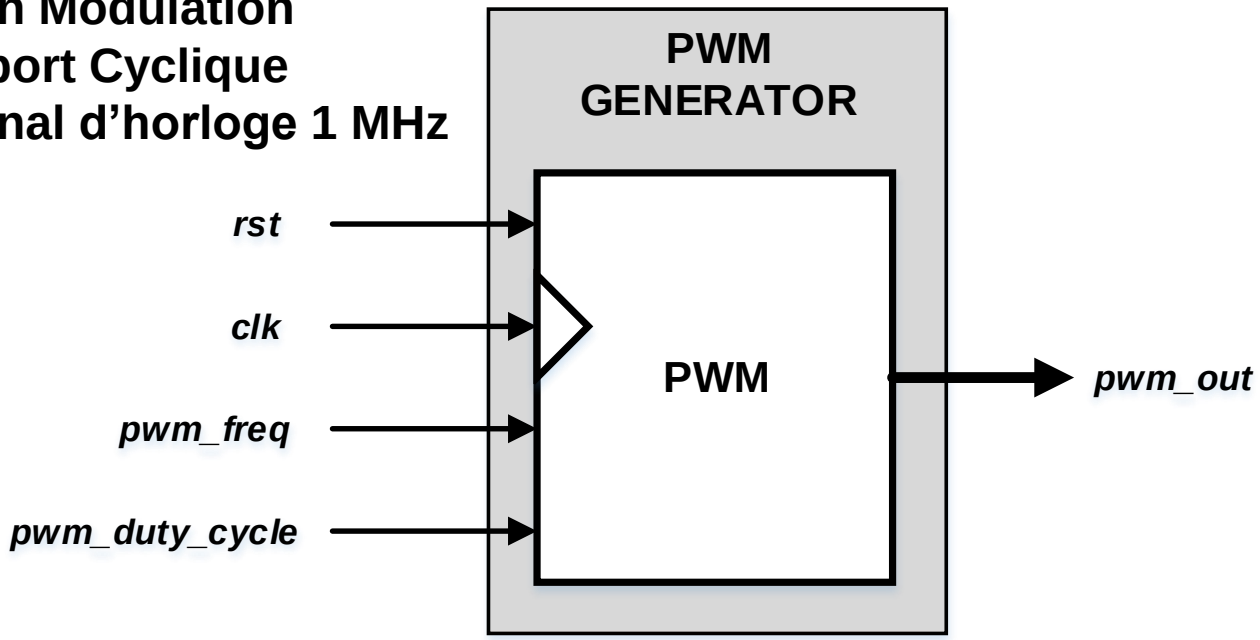
- **Horloge de référence 50 MHz**
- **Génération de signal d'horloge 1 MHz**

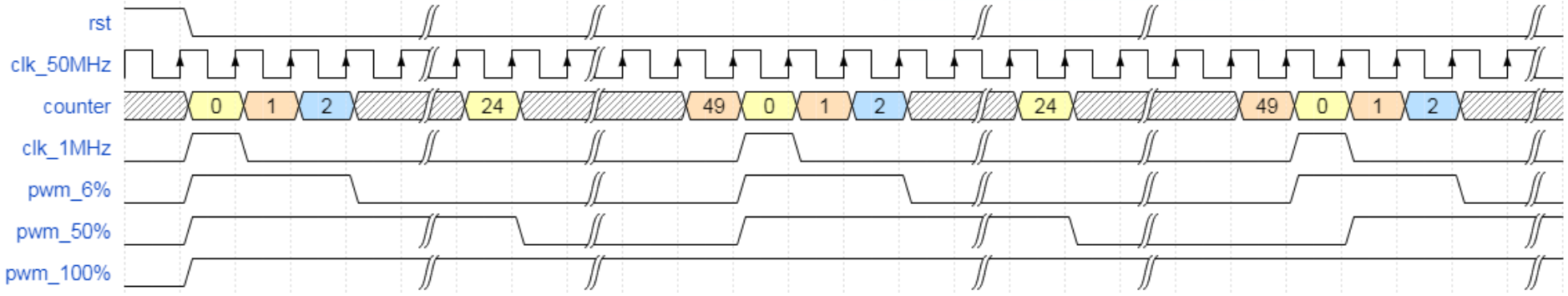




Exemple #3: Génération de signal PWM

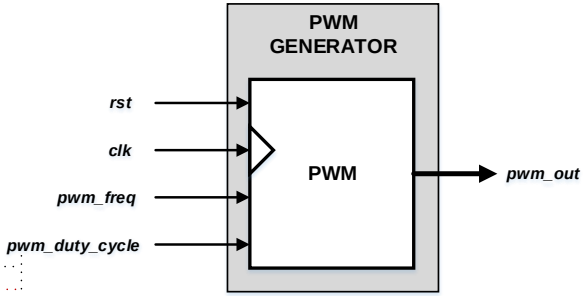
- **PWM : Pulse Width Modulation**
- **Duty Cycle – Rapport Cyclique**
- **Génération de signal d'horloge 1 MHz**

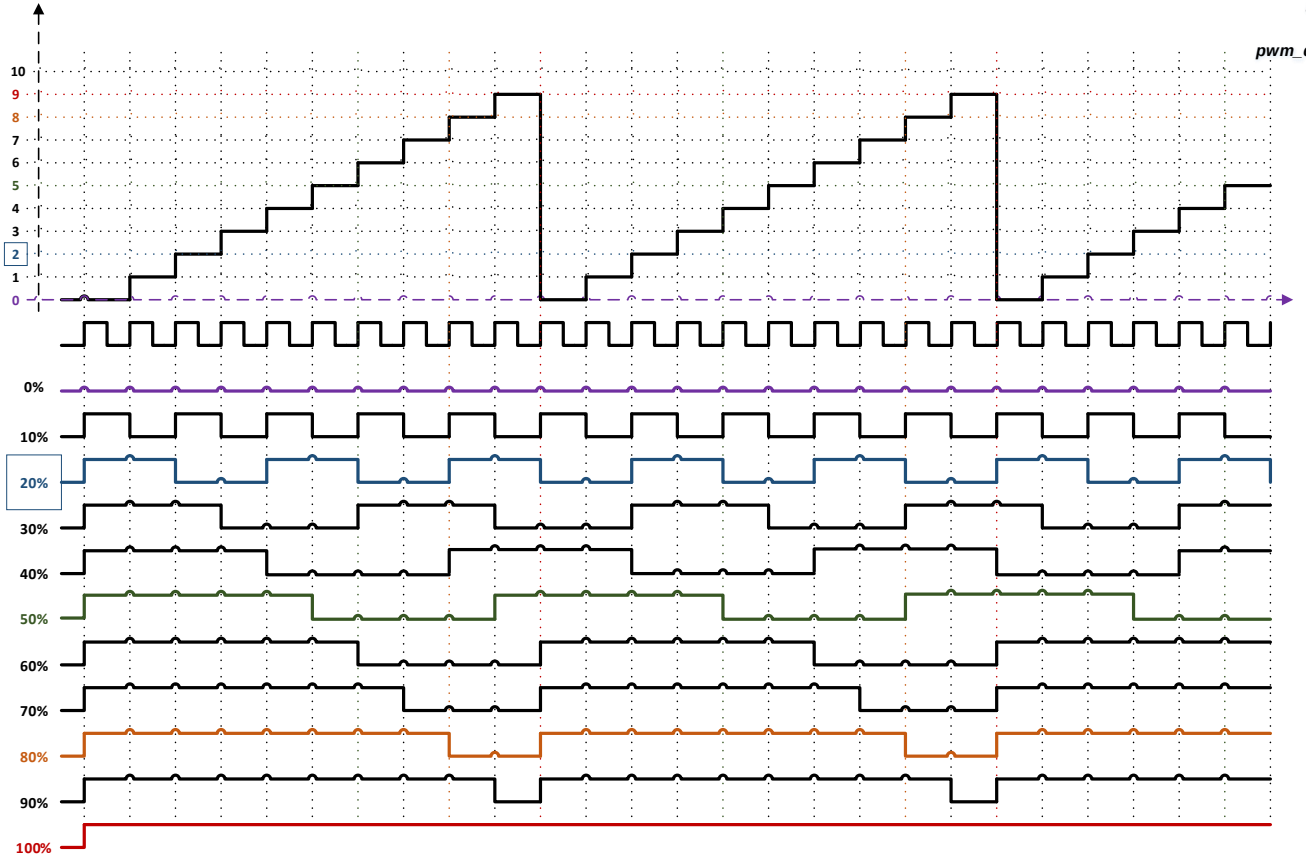




Exemple #3: Génération de signal PWM

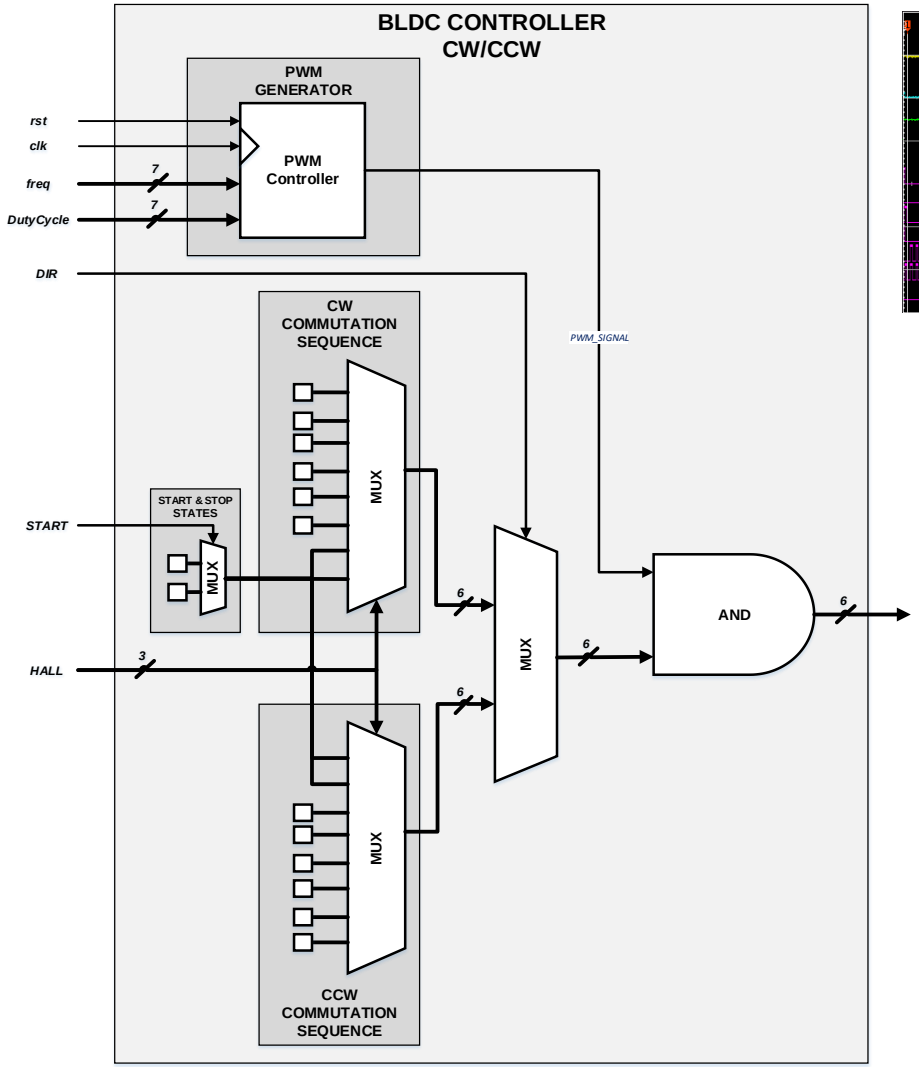
- **PWM : Pulse Width Modulation**
- **Duty Cycle – Rapport Cyclique**
- **Génération de signal d'horloge 1 MHz**

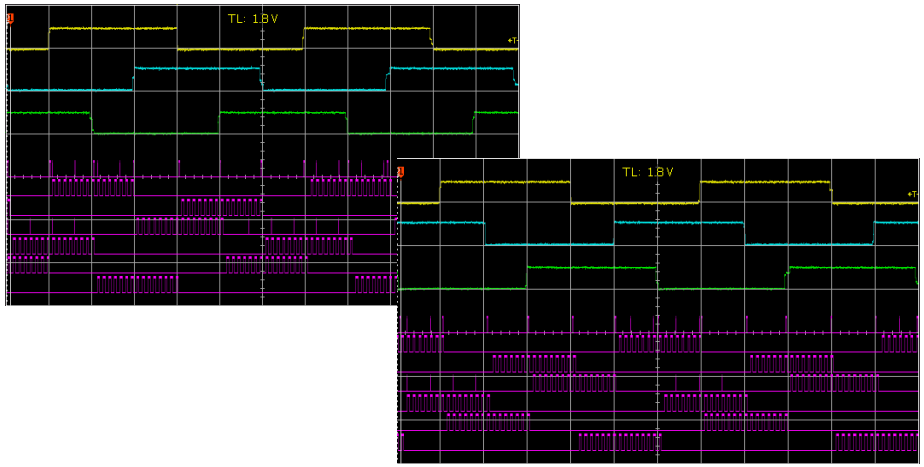


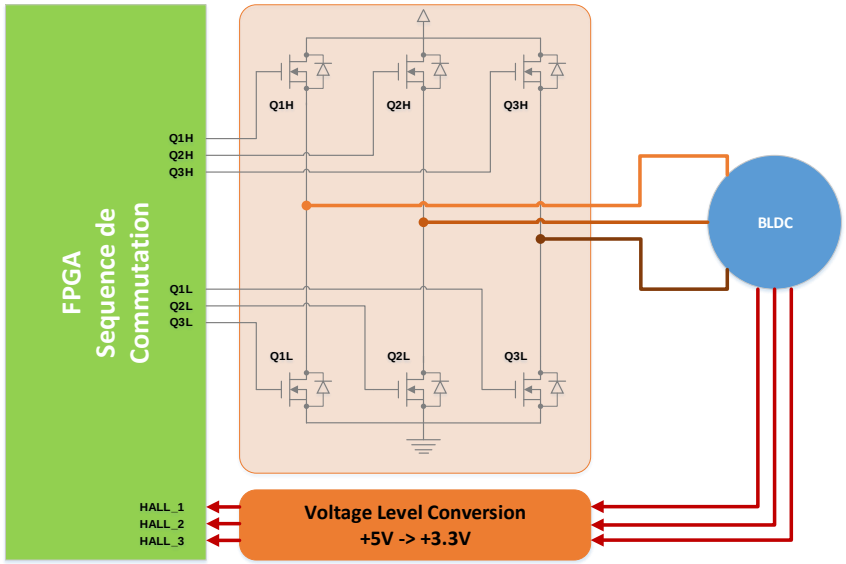


Séquenceur BLDC

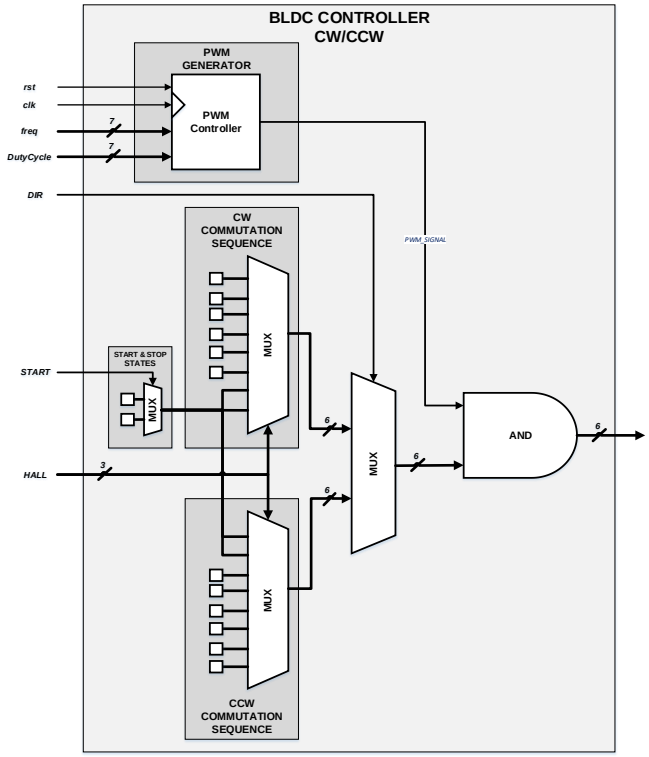
Exemple de séquence de commutation
Sens de rotation horaire

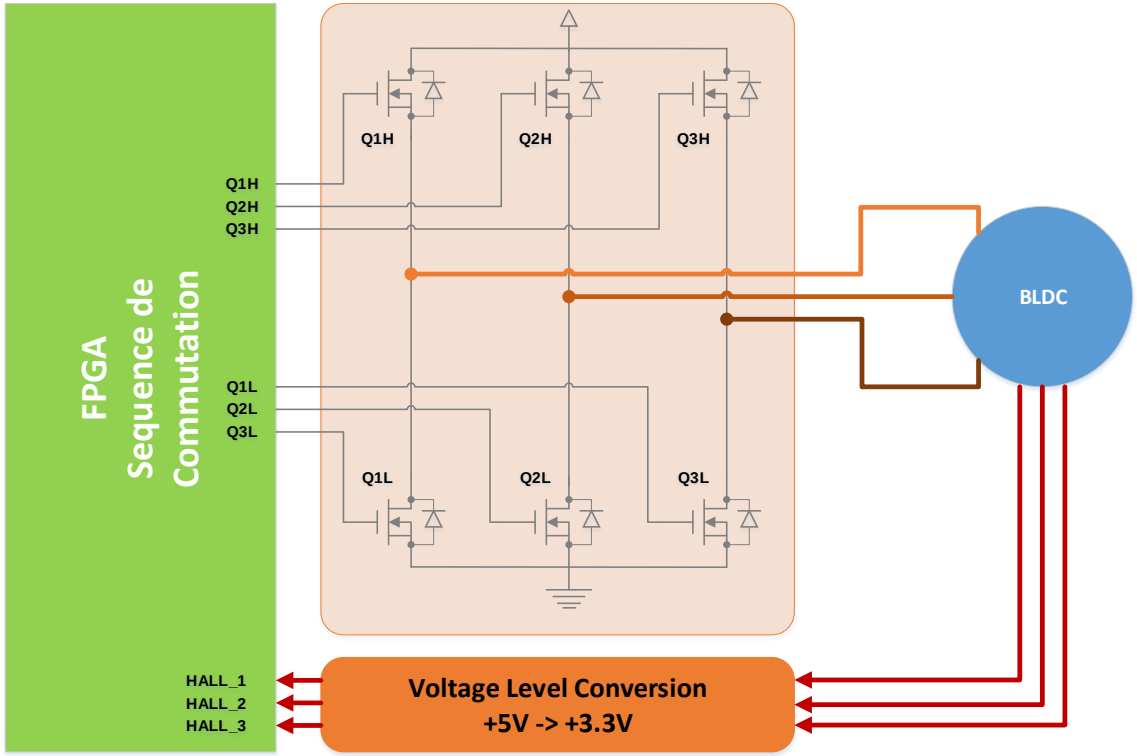


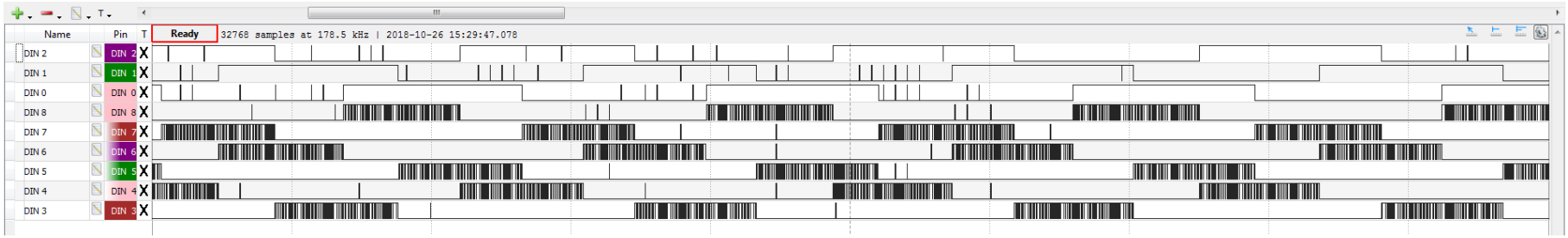




Exemple de séquence de commutation
Sens de rotation horaire



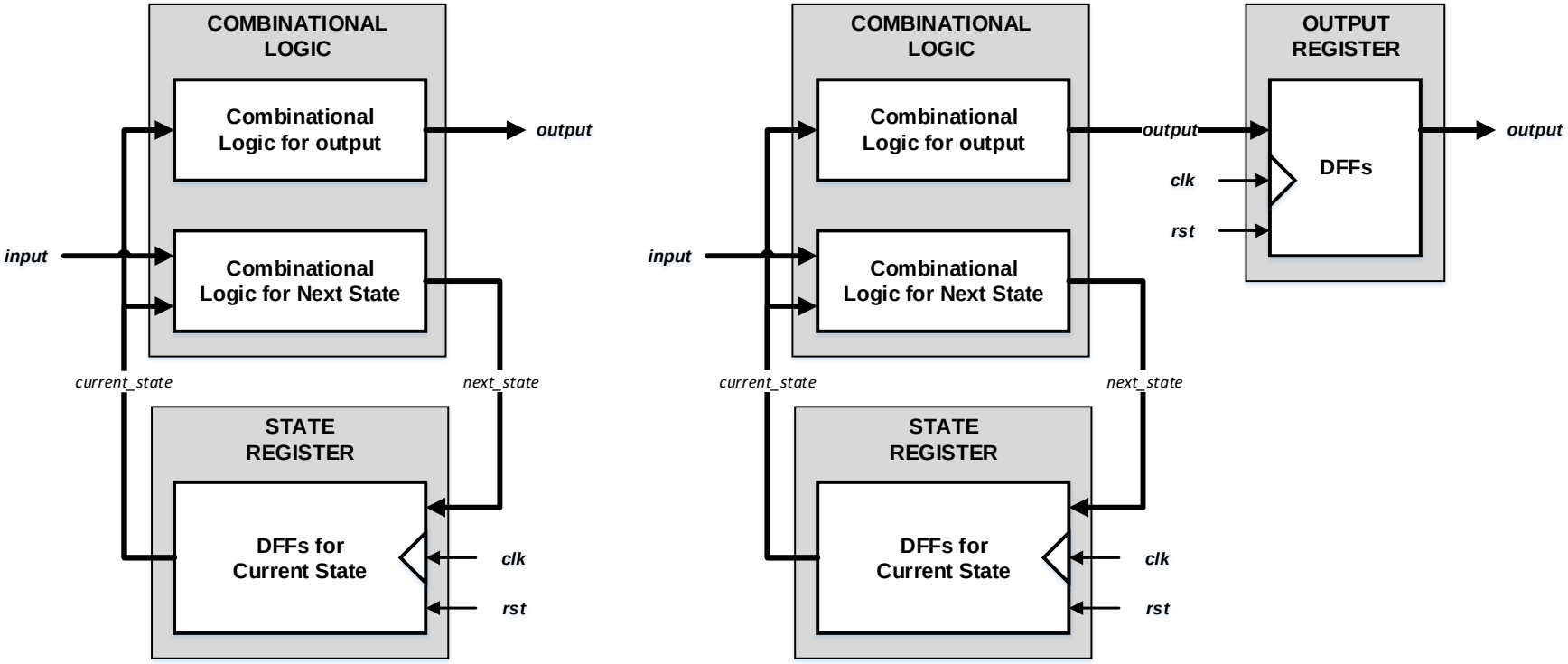




Machine d'état
FSM

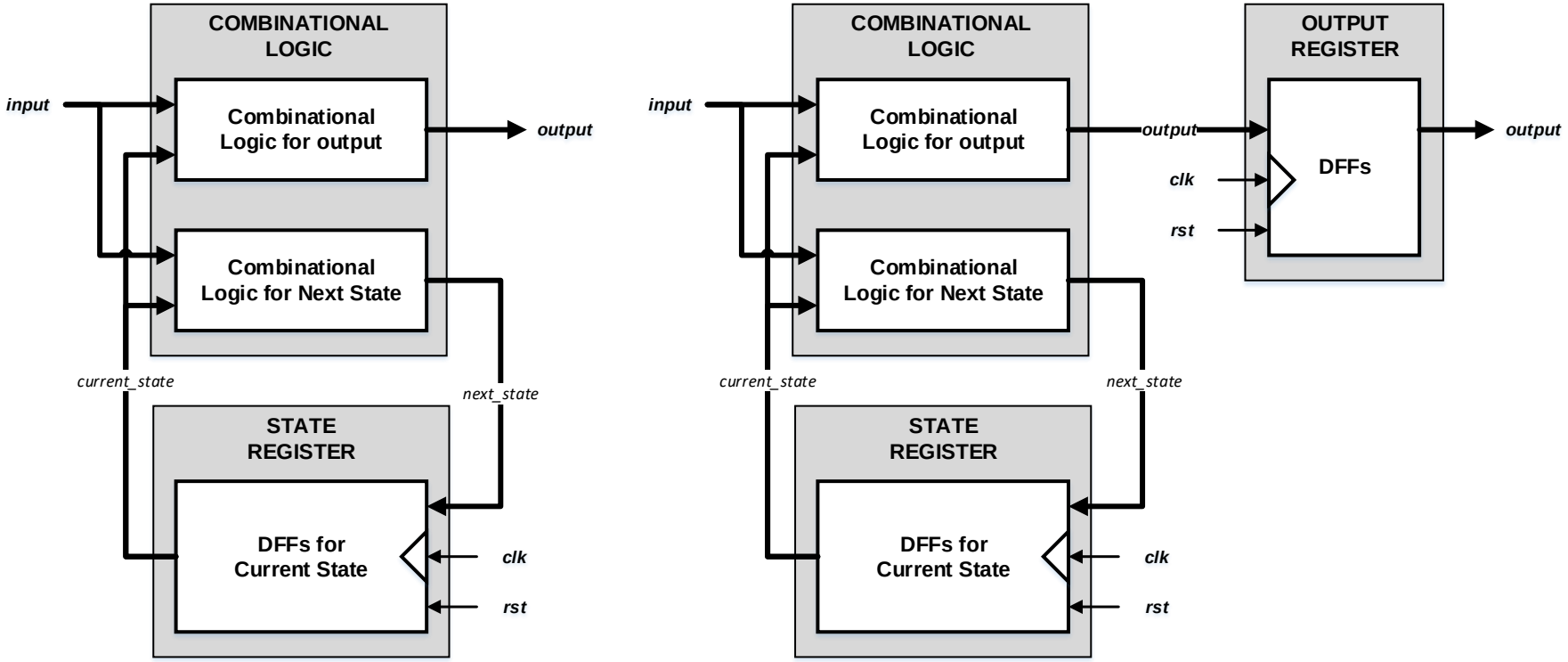
Machine d'état – FSM/ASM

- **Moore FSM**
  1. **La sortie n'est pas fonction de l'entrée**
  2. **current_state, next_state**
  3. **Logic combinatoire pour la sortie**
  4. **Logic combinatoire pour l'état suivant**



Machine d'état – FSM/ASM

- **Mealy FSM**
  1. **La sortie est fonction de l'entrée**
  2. **current_state, next_state**
  3. **Logic combinatoire pour la sortie**
  4. **Logic combinatoire pour l'état suivant**



Encodage d'une FSM

Déclaration des états d'une FSM

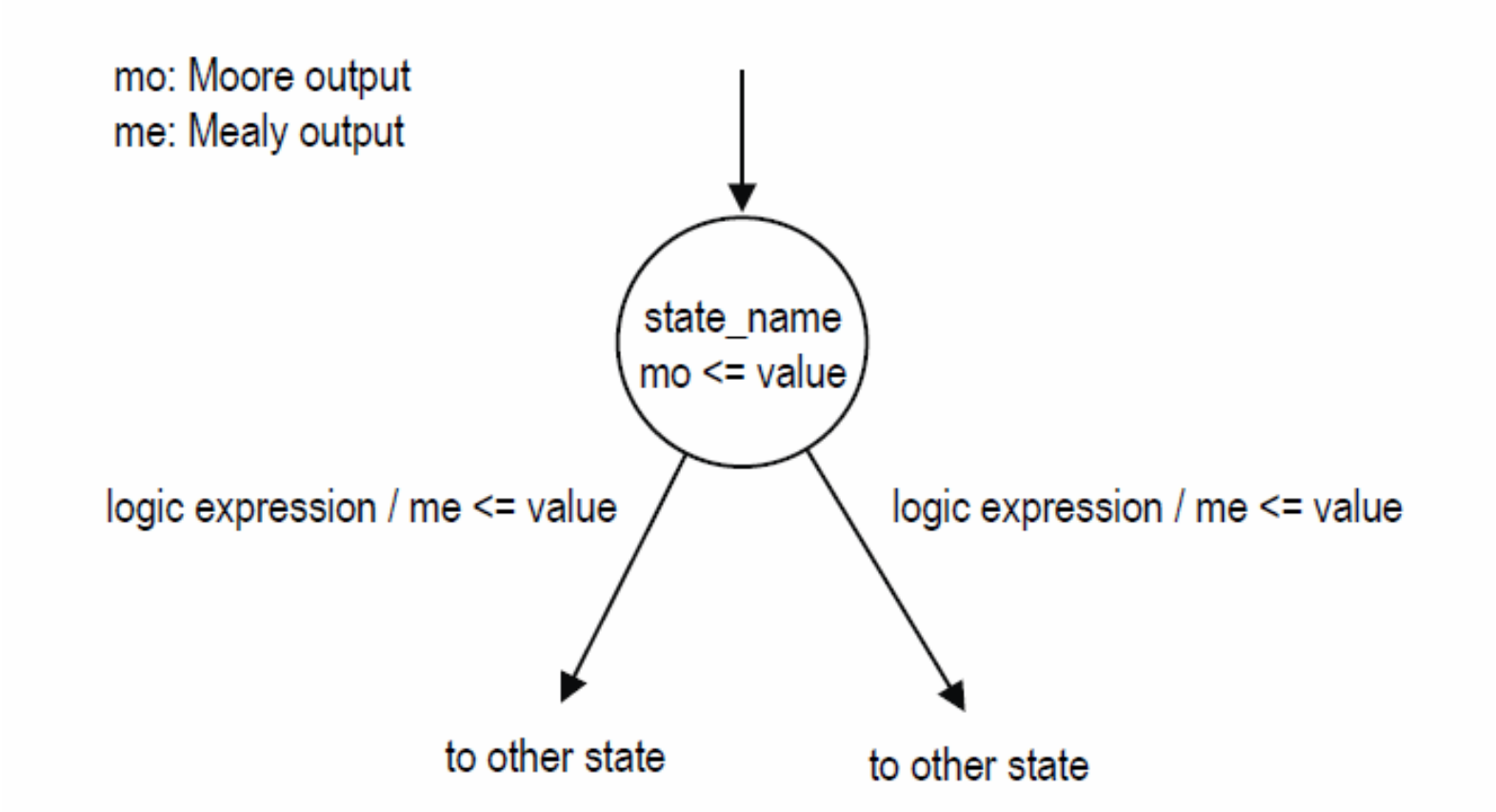
```
TYPE sates IS (A, B, C, D);
SIGNAL current_state, next_state : states := A;
```

- Encodage Binaire Séquentiel
- Encodage One-Hot
- Encodage de Johnson
- Encodage de Gray

- Nombre de bits
- Nombre de Flip-Flops
- Complexité de la logique
- Glitch

State	Encoding			
	Sequential	**Gray**	Johnson	One-Hot
0	000	**000**	0000	00000001
1	001	**001**	0001	00000010
2	010	**011**	0011	00000100
3	011	**010**	0111	00001000
4	100	**110**	1111	00010000
5	101	**111**	1110	00100000
6	110	**101**	1100	01000000
7	111	**100**	1000	10000000

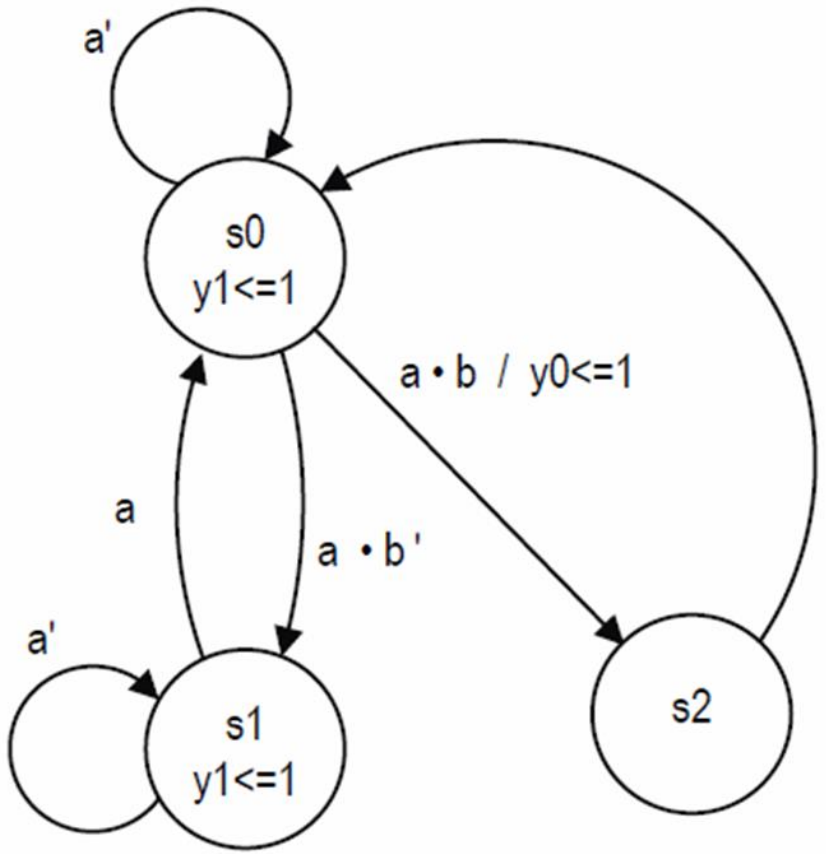
FSM – Diagramme d'états



FSM – Diagramme d'états – Example

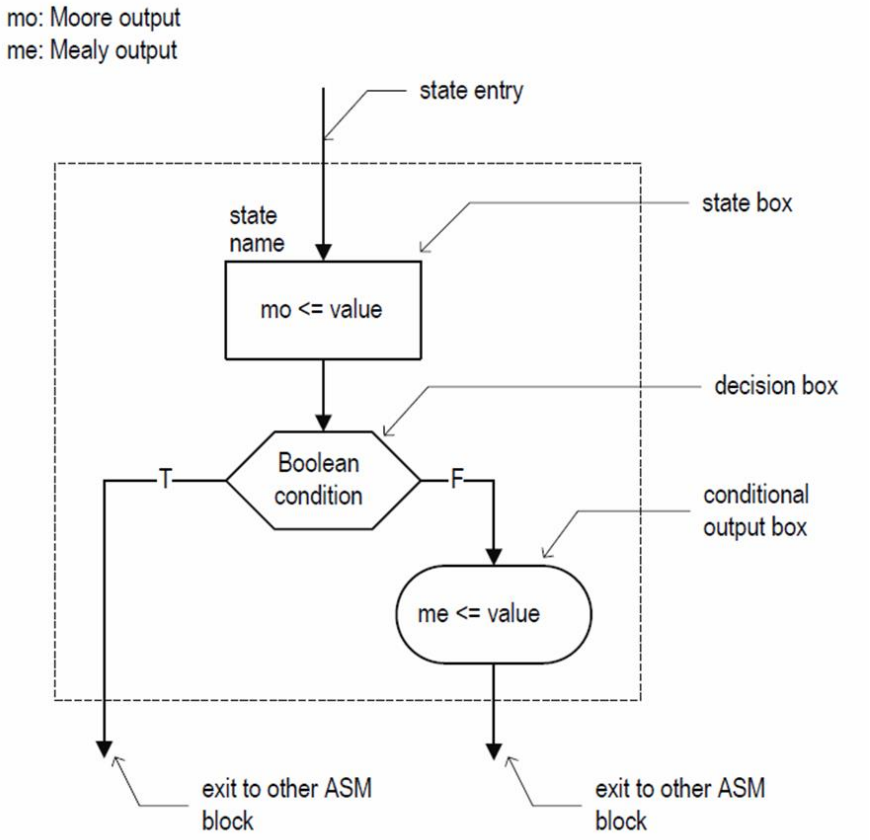
- Valeur par défaut de y0 et y1 est 0
- Déterminer le type de cette machine d'état (Moore ou Mealy

- y1 → Machine de Moore
- y0 → Machine de Mealy



ASM – Organigramme Algorithmique

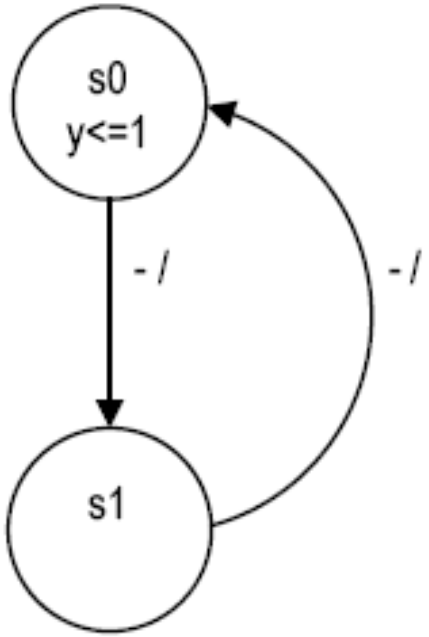
- ASM : Algorithmic State Machine
- L'équivalent d'un organigramme
- Fournit les mêmes informations qu'une représentation FSM
- Plus de description procédurale, utile pour des circuits ou systèmes complexes
- Un bloc par état
- Un ou plusieurs blocs de décisions
- Condition de sortie True/False
- Un ou plusieurs blocs de sortie de traitement

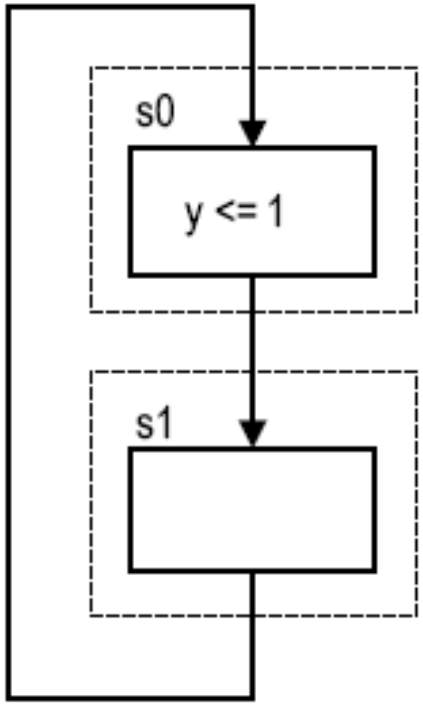


FSM-ASM – Exemple 1

FSM

ASM

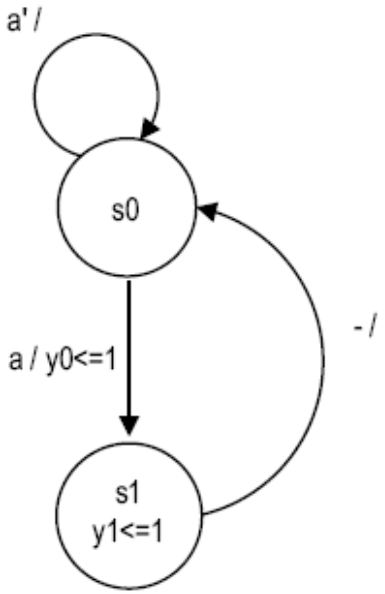


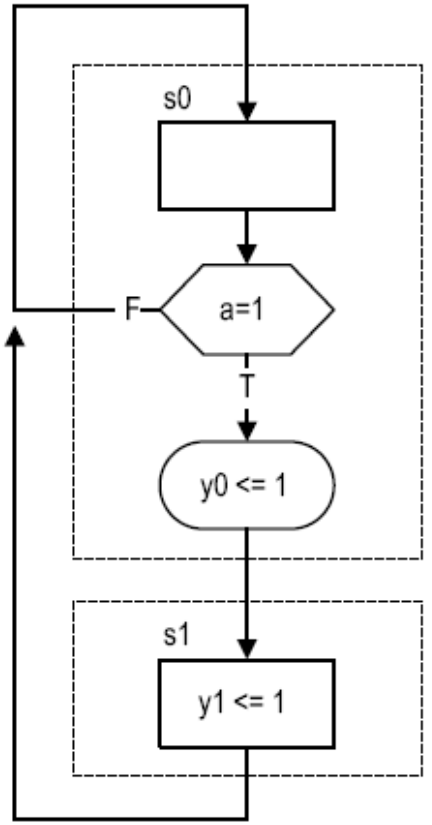


FSM-ASM – Exemple 2

FSM

ASM

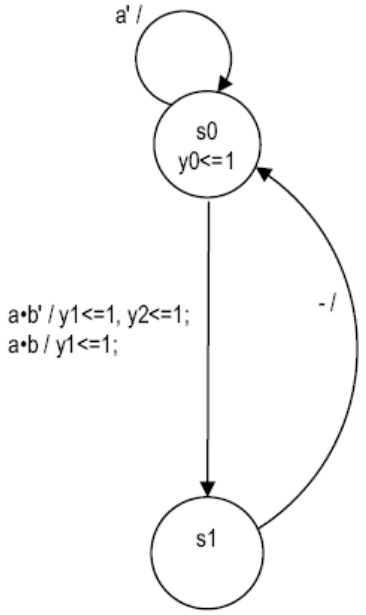


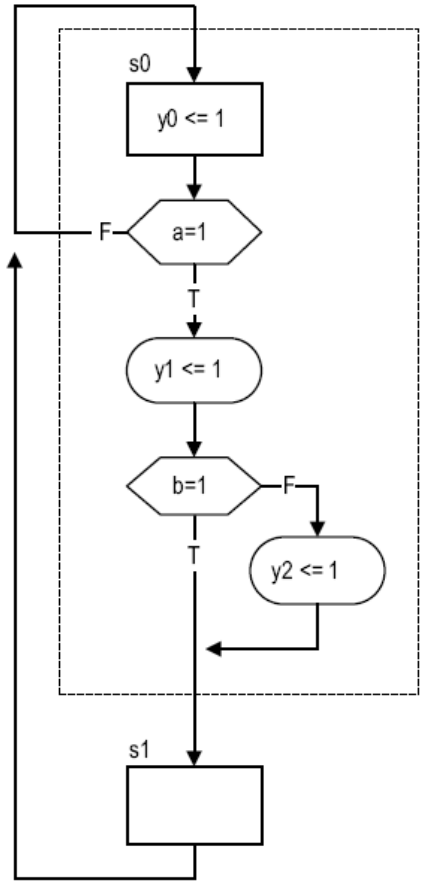


FSM-ASM – Exemple 3

FSM

ASM

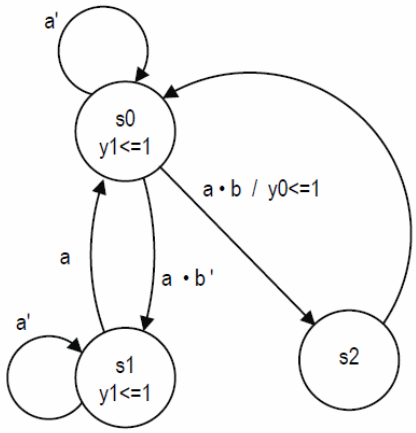


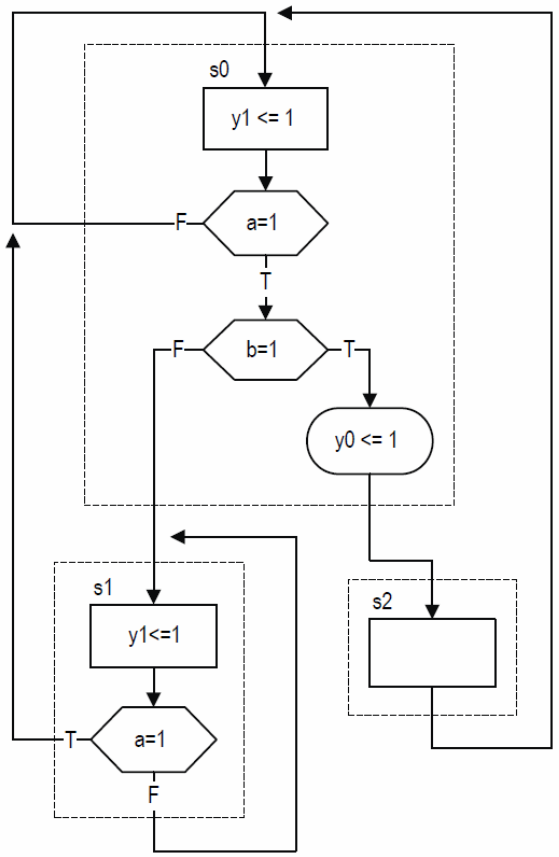


FSM-ASM – Exemple 4

FSM

ASM

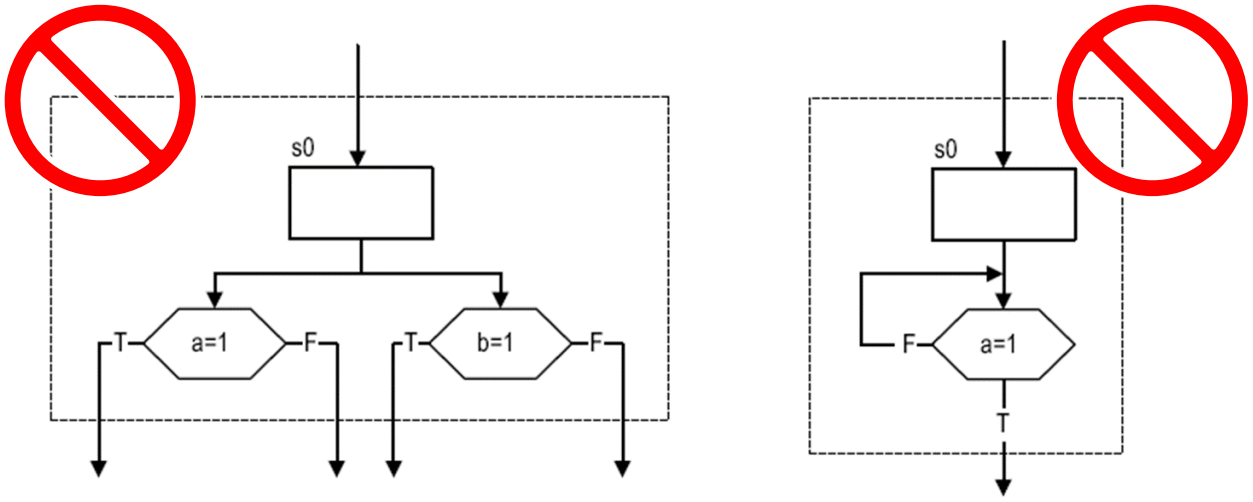




ASM – Règles de base

- Différence entre un organigramme et un ASM
  - La transition est gouvernée par le clock
  - Les transitions sont entre blocs ASM
- Pour chaque combinaison d'entrées, il y a une sortie unique du bloc ASM courant
- La sortie d'un bloc ASM doit mener à un bloc d'état.
- Le bloc d'état de sortie peut être celui du bloc ASM courant ou un autre bloc ASM d'un autre état.

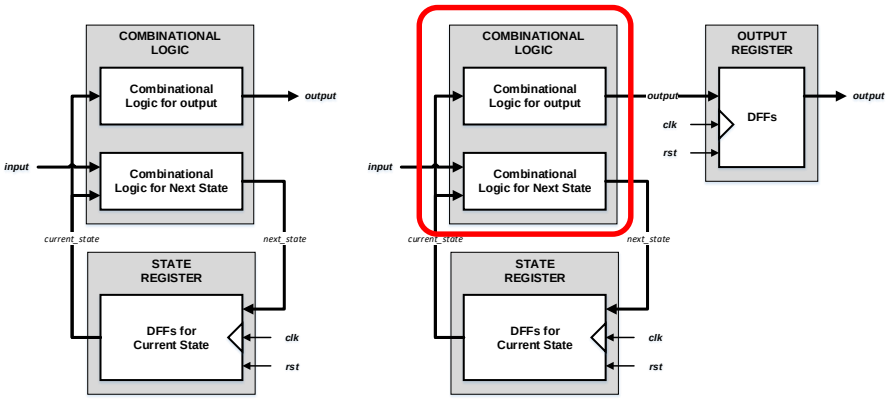
Blocs ASM incorrectes



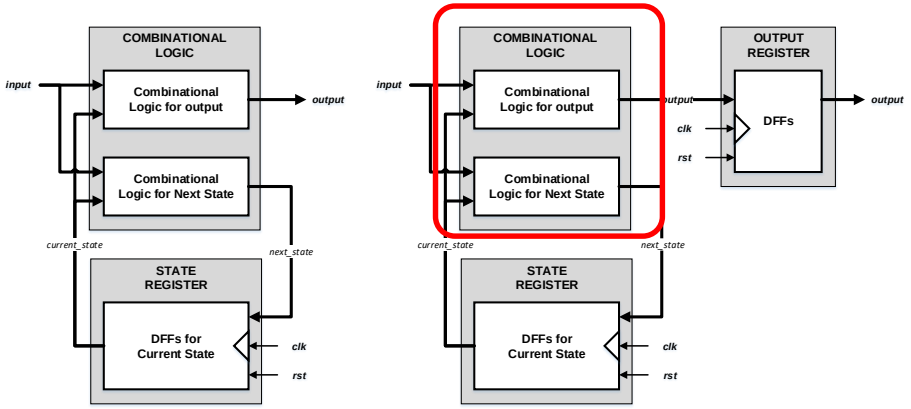
Description de FSM en VHDL

- 4 blocs élémentaires:
  - state register,
  - next-state logic,
  - Moore output logic,
  - and Mealy output logic
- **Style de codage des FSMs**
  - Multiple segments: un segment de code par bloc
  - Deux segments:
    - Régistre d'état (State Register Process)
    - Logic combinatoire (Cambinational Logic Process): next-state logic, Moore output logic, and Mealy output logic

Moore



Mealy



Template – Moore FSM

```
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
------------------------------------------------------------

entity moore_fsm is
    generic (
        param1: std_logic_vector(...) := value1 ;
        param2: integer := value2 ;
        ...);
port (
    clk, rst: in std_logic;
    input1, input2, ...  : in std_logic_vector(...);
    output1, output2, ...: out std_logic_vector(...);
end entity;
------------------------------------------------------------

architecture moore_fsm_arch of moore_fsm is
type states is (A, B, C, ...);
signal current_state, next_state: states;
attribute enum_encoding: string; --optional
attribute enum_encoding of state: type is "gray";
begin

--State Register:
SR_PROC : process (clk, rst)
Begin
    --State Register Logic…
end process;

--Combinational for Next State and Logic for Output:
CLNSO_PROC : process (all)
Begin
    --Next State and Output Logic…
end process;

--Optional output register:
OR_PROC : process (clk, rst)
Begin
    --Optional output register logic…
end process;

end architecture;
------------------------------------------------------------
```

```
--State Register:
SR_PROC : process (clk, rst)
begin
    if rst='1' then --see Note 2 above on boolean tests
        current_state <= A;
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;
```

```
--Combinational for Next State and Logic for Output:
CLNSO_PROC : process (all)
begin
    case current_state is
        when A =>
            output1 <= value1 ;
            output2 <= value2 ;
            ...
            if ( condition1 ) then
                next_state <= B;
            elsif ( condition2 ) then
                next_state <= ...;
            else
                next_state <= A;
            end if;
        when B =>
            ...
        when C =>
            ...
        end case;
end process;
```

```
--Optional output register:
OR_PROC : process (clk, rst)
begin
    if rst='1' then --rst generally optional here
        new_output1 <= ...;
        ...
    elsif rising_edge(clk) then
        new_output1 <= output1;
        ...
    end if;
end process;
```

Template – Mealy FSM

```
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
------------------------------------------------------------

entity mealy_fsm is
    generic (
        param1: std_logic_vector(...) := value1 ;
        param2: integer := value2 ;
        ...);
port (
    clk, rst: in std_logic;
    input1, input2, ...  : in std_logic_vector(...);
    output1, output2, ...: out std_logic_vector(...);
end entity;
------------------------------------------------------------

architecture mealy_fsm_arch of mealy_fsm is
type states is (A, B, C, ...);
signal current_state, next_state: states;
attribute enum_encoding: string; --optional
attribute enum_encoding of state: type is "gray";
begin

--State Register:
SR_PROC : process (clk, rst)
Begin
    --State Register Logic…
end process;

--Combinational for Next State and Logic for Output:
CLNSO_PROC : process (all)
Begin
    --Next State and Output Logic…
end process;

--Optional output register:
OR_PROC : process (clk, rst)
Begin
    --Optional output register logic…
end process;

end architecture;
------------------------------------------------------------
```

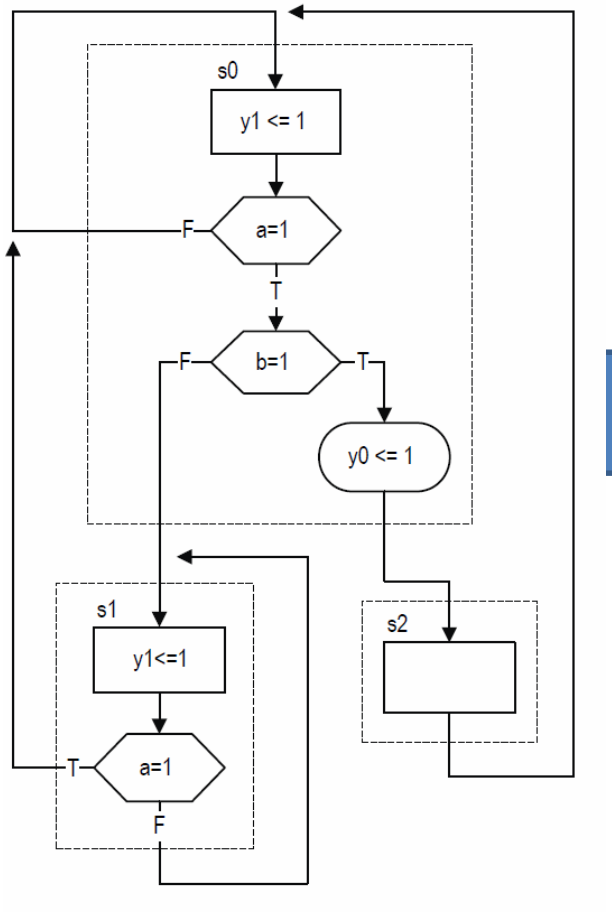
```
--State Register:
SR_PROC : process (clk, rst)
begin
    if rst='1' then --see Note 2 above on boolean tests
        current_state <= A;
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;
```

```
--Combinational for Next State and Logic for Output:
CLNSO_PROC : process (all)
begin
    case current_state is
        when A =>
            if ( condition1 ) then
                output1 <= value1 ;
                output2 <= value2 ;
                ...
                next_state <= B;
            elsif ( condition2 ) then
                output1 <= value3 ;
                output2 <= value4 ;
                next_state < = ...;
            else
                output1 <= default_value1 ;
                output2 <= default_value2 ;
                ...
               next_state <= A;
            end if;
        when B =>
            ...
     end case;
end process;
```

```
--Optional output register:
OR_PROC : process (clk, rst)
begin
    if rst='1' then --rst generally optional here
        new_output1 <= ...;
        ...
    elsif rising_edge(clk) then
        new_output1 <= output1;
        ...
    end if;
end process;
```

Exemple ASM en VHDL – Multiples Segments

ASM

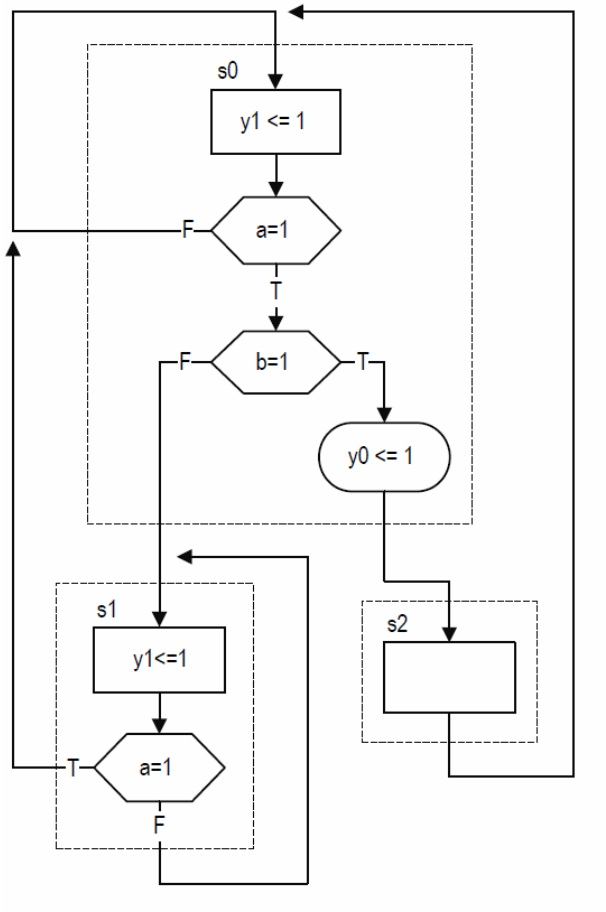


VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity fsm_eg is
   port(
      clk, reset : in  std_logic;
      a, b       : in  std_logic;
      y0, y1     : out std_logic
   );
end fsm_eg;

architecture mult_seg_arch of fsm_eg is
   type eg_state_type is (s0, s1, s2);
   signal state_reg, state_next : eg_state_type;

begin


   -- state register
   process(clk, reset)
   begin
      if (reset = '1') then
         state_reg <= s0;
      elsif (clk'event and clk = '1') then
         state_reg <= state_next;
      end if;
   end process;
```

Exemple ASM en VHDL – Multiples Segments

ASM

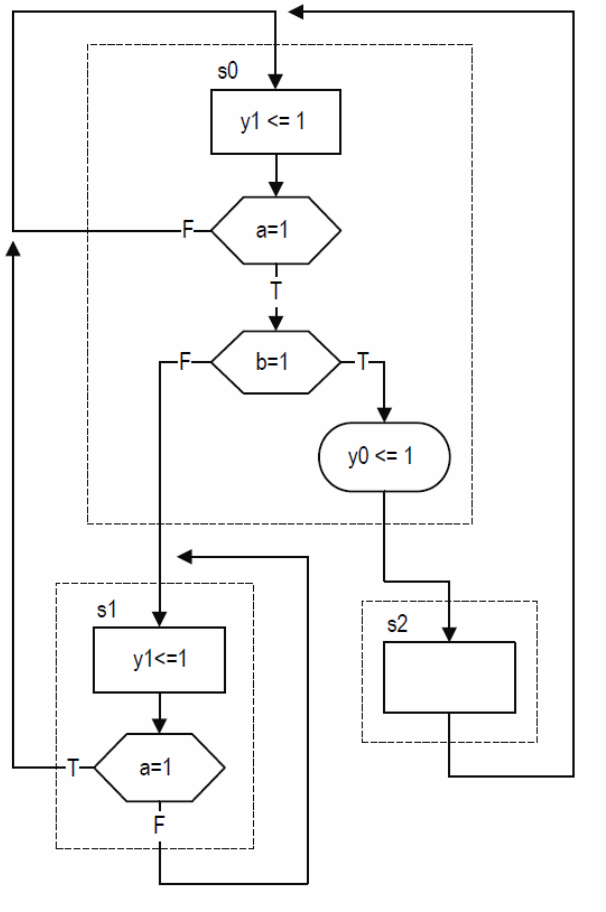


VHDL

```
-- next-state logic
process(state_reg, a, b)
begin
   case state_reg is
      when s0 =>
         if a = '1' then
            if b = '1' then
               state_next <= s2;
            else
               state_next <= s1;
            end if;
         else
            state_next <= s0;
         end if;
      when s1 =>
         if (a = '1') then
            state_next <= s0;
         else
            state_next <= s1;
         end if;
      when s2 =>
         state_next <= s0;
   end case;
end process;
```

Exemple ASM en VHDL – Multiples Segments

ASM

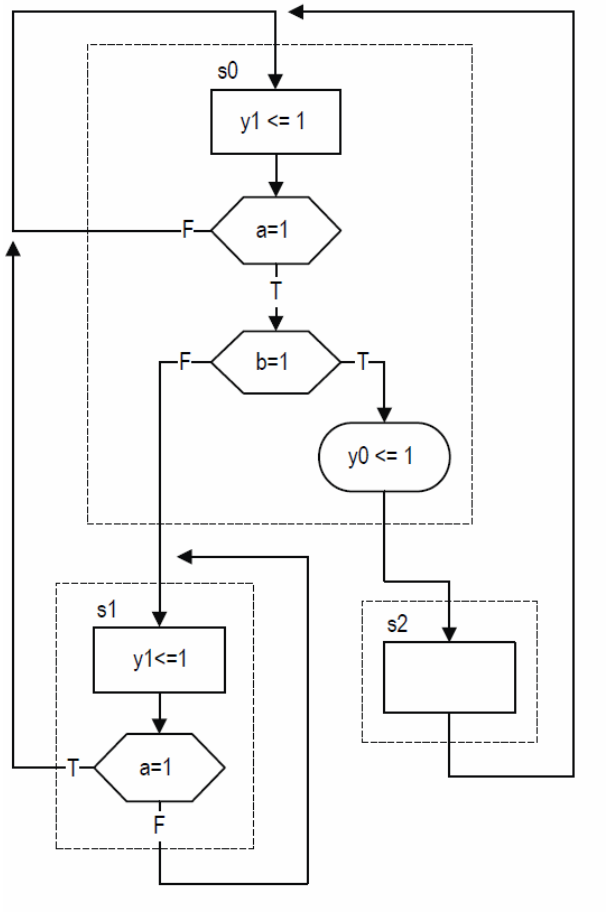


VHDL

```
-- Moore output logic
process(state_reg)
begin
   case state_reg is
      when s0 | s1 =>
         y1 <= '1';
      when s2 =>
         y1 <= '0';
   end case;
end process;

-- Mealy output logic
process(state_reg, a, b)
begin
   case state_reg is
      when s0 =>
         if (a='1') and (b='1') then
            y0 <= '1';
         else
            y0 <= '0';
         end if;
      when s1 | s2 =>
         y0 <= '0';
   end case;
end process;
end mult_seg_arch;
```

Exemple ASM en VHDL – Deux Segments

ASM

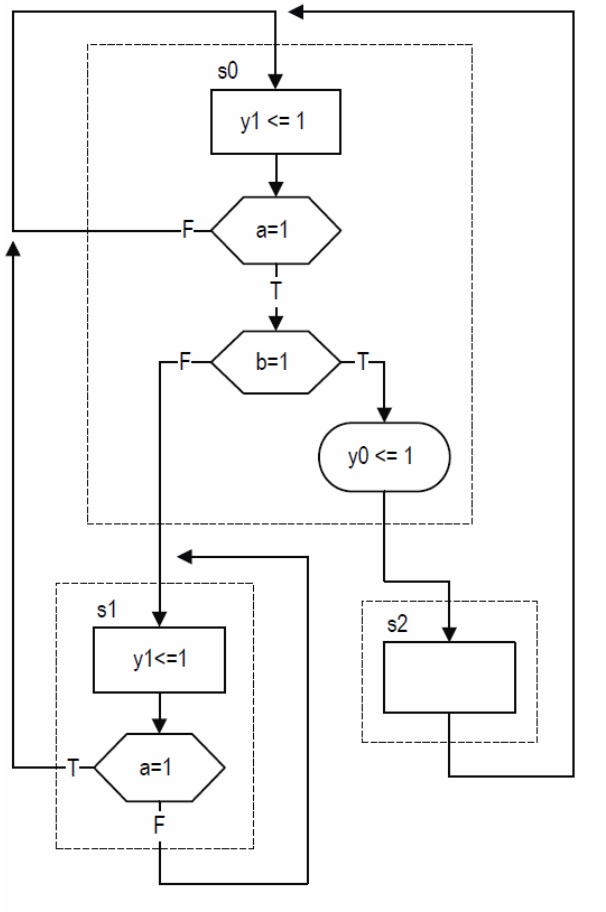


VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity fsm_eg is
   port(
      clk, reset : in  std_logic;
      a, b       : in  std_logic;
      y0, y1     : out std_logic
   );
end fsm_eg;

architecture mult_seg_arch of fsm_eg is
   type eg_state_type is (s0, s1, s2);
   signal state_reg, state_next : eg_state_type;

begin


   -- state register
   process(clk, reset)
   begin
      if (reset = '1') then
         state_reg <= s0;
      elsif (clk'event and clk = '1') then
         state_reg <= state_next;
      end if;
   end process;
```

Exemple ASM en VHDL – Deux Segments

ASM



VHDL

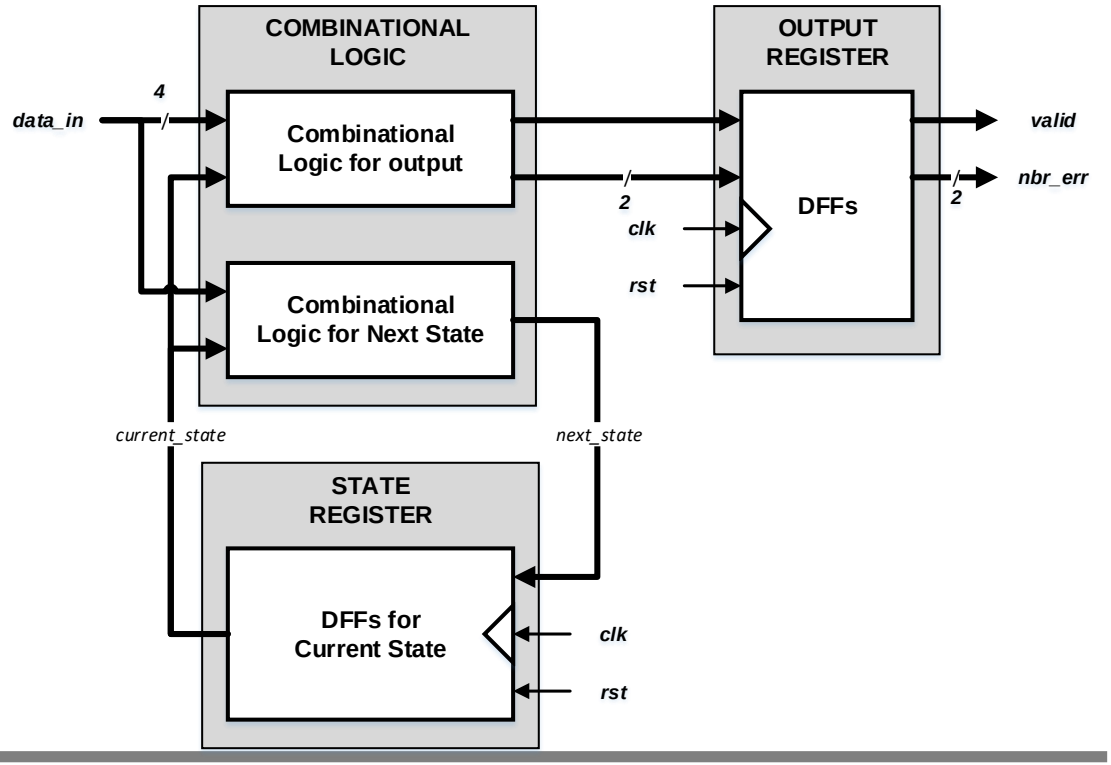
```
-- next-state/output logic
process(state_reg, a, b)
begin
   state_next <= state_reg;  -- default the same state
   y0         <= '0';        -- default 0
   y1         <= '0';        -- default 0
   case state_reg is
      when s0 =>
         y1 <= '1';
         if a = '1' then
            if b = '1' then
               state_next <= s2;
               y0         <= '1';
            else
               state_next <= s1;
            end if;
         -- no else branch
         end if;
      when s1 =>
         y1 <= '1';
         if (a = '1') then
            state_next <= s0;
         -- no else branch
         end if;
      when s2 =>
         state_next <= s0;
   end case;
end process;
```

Exercice – Détection de Séquence

- **Mealy FSM**
  1. **Entrées :** *clk, rst, data_in<4>*
  2. **Sorties :** *valid, nbr_err*
  3. **Séquence :** 0xA, 0xB, 0xC, 0xD
  4. **Proposer une FSM incluant la définition des états et les conditions de transition**
  5. **Propose un ASM**
  6. **Implémenter ce module**

- **STATE SIGNALS**
  1. *current_state*
  2. *next_state*

- **PROCESS**
  1. *SR_PROC*
  2. *CLNSO_PROC*
  3. *OR_PROC*



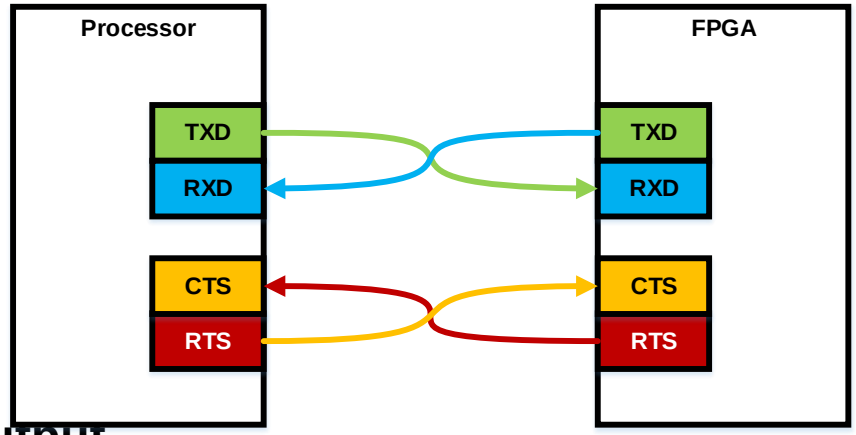
Partie #2 : ISE/VHDL
Communication Série SPI/UART

I. **Protocole UART**
   1. **Chronogramme UART**
   2. **FSM UART**
II. **Protocole SPI**
   1. **Architecture Maître/Esclave**
   2. **Mode de fonctionnement**
   3. **Chronogramme SPI**
   4. **FSM SPI**

Bus de Communication UART

Bus de communication UART

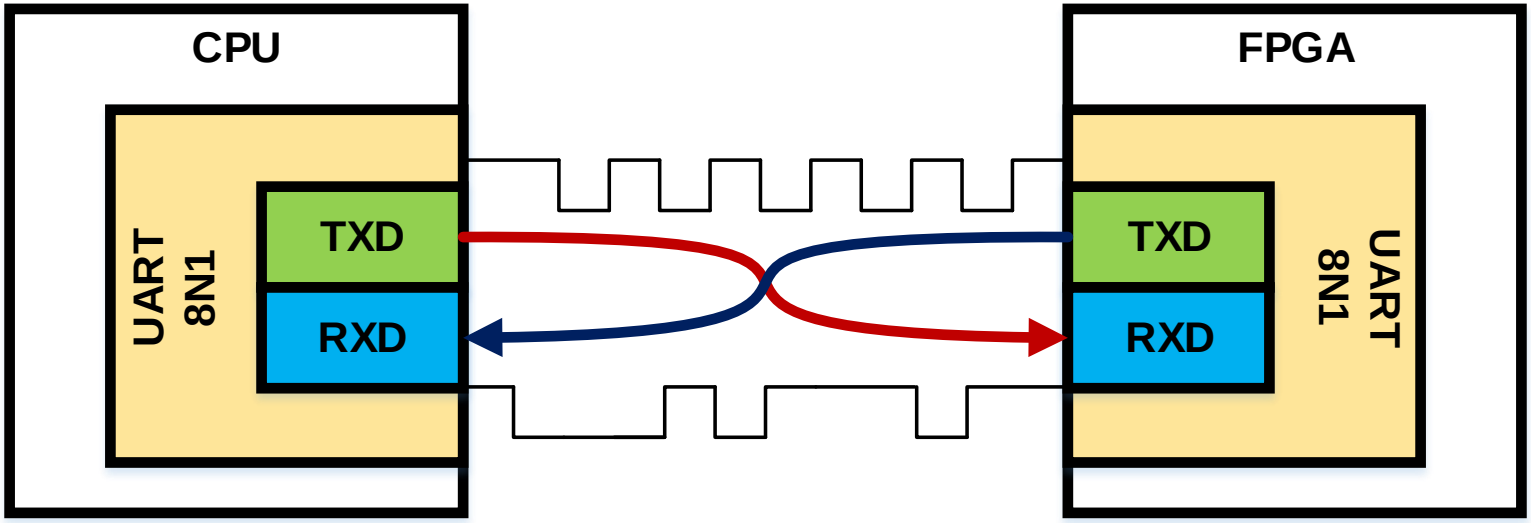
- Bus de communication série
- Communication asynchrone
- Communication full duplex
- Écriture et Lecture en Parallèle



- RXD : Received Data – Serial Data Output
- TXD : Transmitted Data – Serial Data Input
- CTS : Clear To Send – active low
- RTS : Request To Send – active low
- DTR : Data Terminal Ready
- DSR : Data Set Ready
- Baud Rate : 300, 1200, 2400, 4800, 9600, 19.2k, 38.4k, 57.6k, 115.2k
- Principales utilisations :
  - Communication vers un périphérique dédié :
  - Communication entre microprocesseurs
  - Communication entre ordinateur et modem

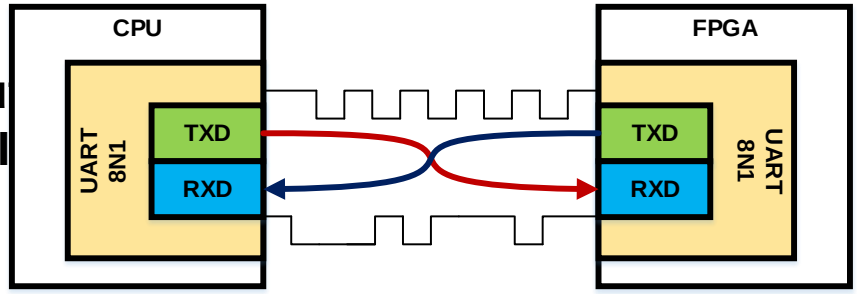
UART – 8N1

- UART – 8N1
- RXD : Received Data – Serial Data Output
- TXD : Transmitted Data – Serial Data Input
- Baud Rate : 9600, 115.2k
- RX: Oversampling 16
- 8 Bits Data
- No Parity
- 1 Stop Bit

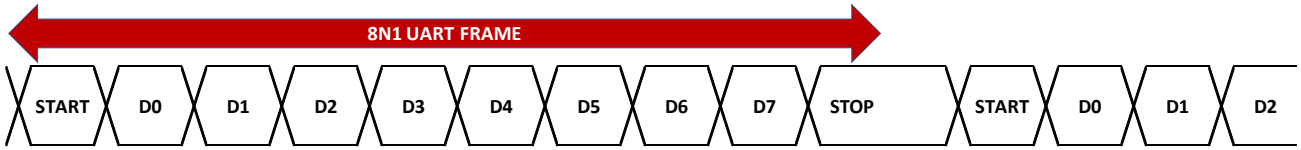


UART – 8N1

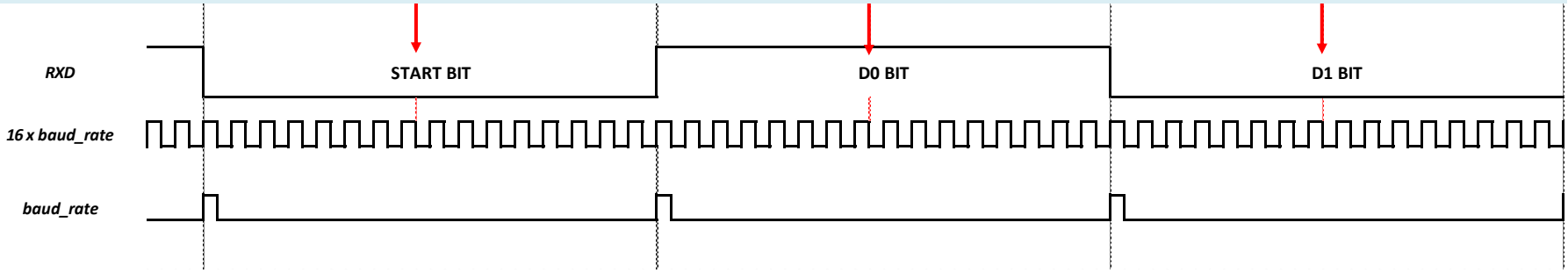
- UART – 8N1
- RXD : Received Data – Serial Data Ou
- TXD : Transmitted Data – Serial Data I
- Baud Rate : 9600, 115.2k
- RX: Oversampling 16
- 8 Bits Data
- No Parity
- 1 Stop Bit



TRAME UART 8N1

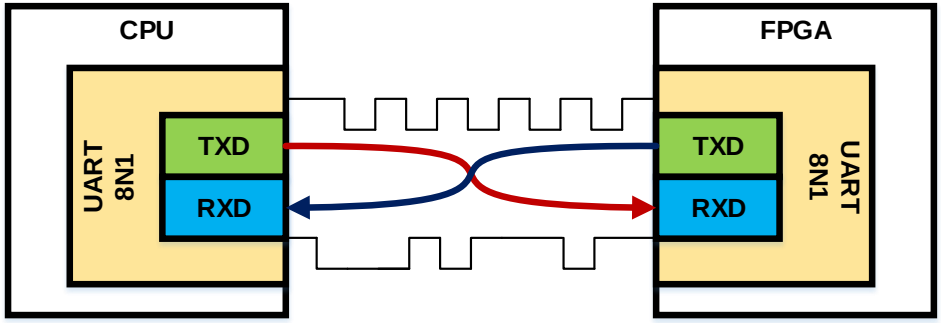


Réception Asynchrone
Sur-échantillonnage de facteur 16 du Baud Rate



UART – 8N1

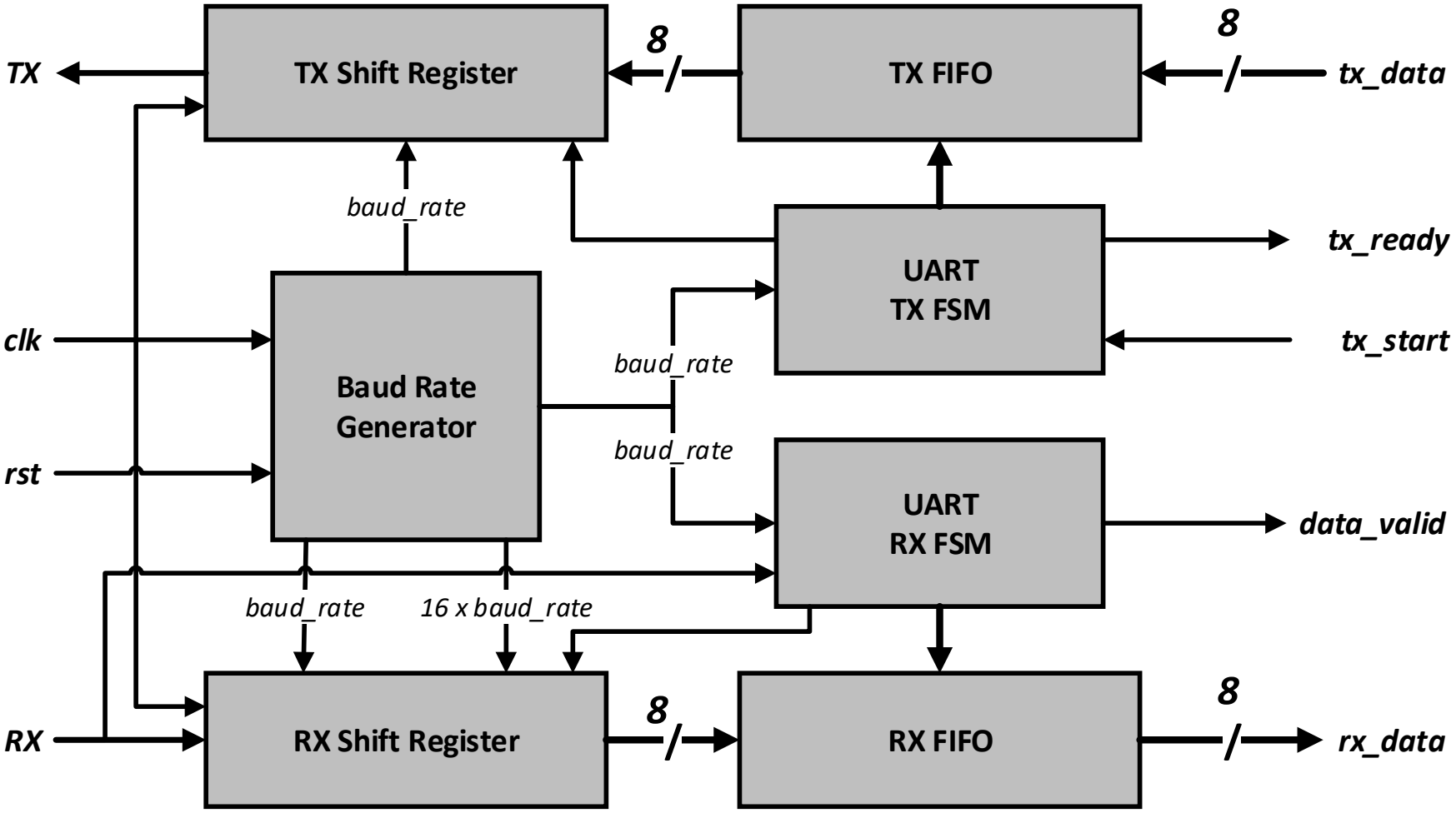
- UART – 8N1
- RXD : Received Data
- TXD : Transmitted Data
- Baud Rate : 9600, 115.2k
- RX: Oversampling 16
- 8 Bits Data
- No Parity
- 1 Stop Bit



Generic	Type	Default
System Clock (clk)	integer	50_000_000
Baud Rate	integer	9_600 ou 115_200
Over Sampling	integer	16

Baud Rate	OSR	DIVIDER 50 MHz/(OSR x Baud Rate)	DIVIDER 50 MHz/Baud Rate
9600	16	325	5208
115200	16	27	434

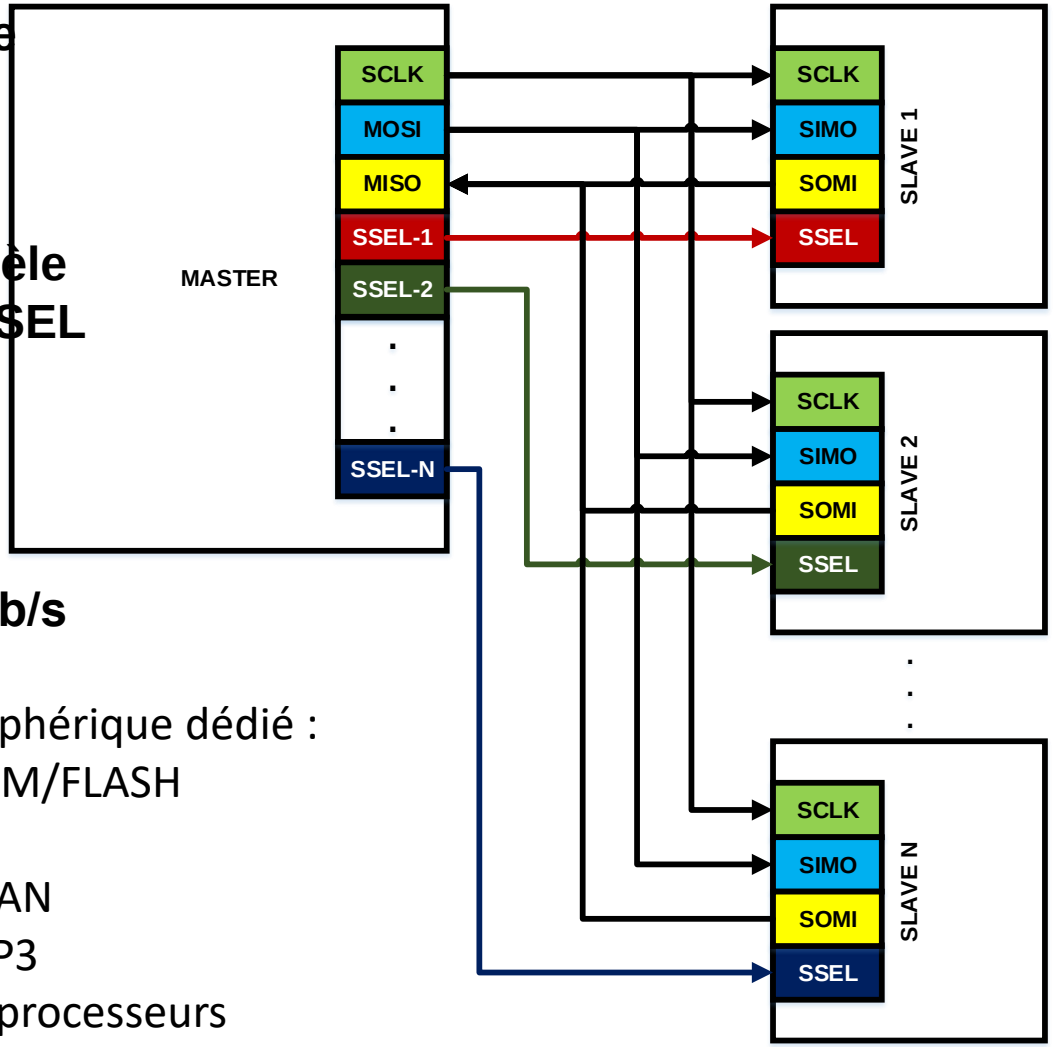
UART – 8N1


TX
TX Shift Register
8
TX FIFO
8
tx_data
baud_rate
tx_ready
UART
TX FSM
clk
Baud Rate
Generator
baud_rate
tx_start
rst
baud_rate
UART
RX FSM
data_valid
baud_rate
16 x baud_rate
RX
RX Shift Register
8
RX FIFO
8
rx_data

Bus de Communication SPI

Bus de communication SPI

- **Bus de communication série**
- **Architecture maître/esclave**
- **Communication synchrone**
- **Communication full duplex**
- **Écriture et Lecture en Parallèle**
- **Un esclave actif à la fois - SSEL**
- **SSEL : Slave Select**
- **SCLK : Clock Signal**
- **MOSI : Master Out Slave In**
- **MISO : Master In Slave Out**
- **Débit : 250Kb/s jusqu'à 10Mb/s**
- **Principales utilisations :**
  - Communication vers un périphérique dédié :
    - Mémoire : RAM/EEPROM/FLASH
    - Capteur, actionneur
    - Convertisseur CAN et CAN
    - Décodeur/Encodeur MP3
  - Communication entre microprocesseurs

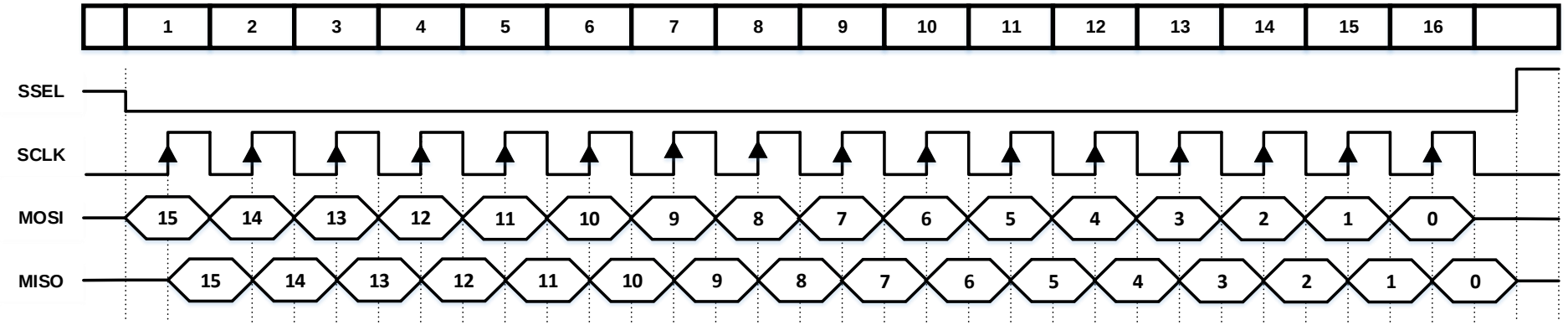


Bus de communication SPI

- **Paramètre du Bus SPI**
  1. **La fréquence d'horloge – fixée par le maîtr**
  2. **La polarité de l'horloge – CPOL (Clock Pol**
  3. **La phase de l'horloge – CPHA (Clock Phas**

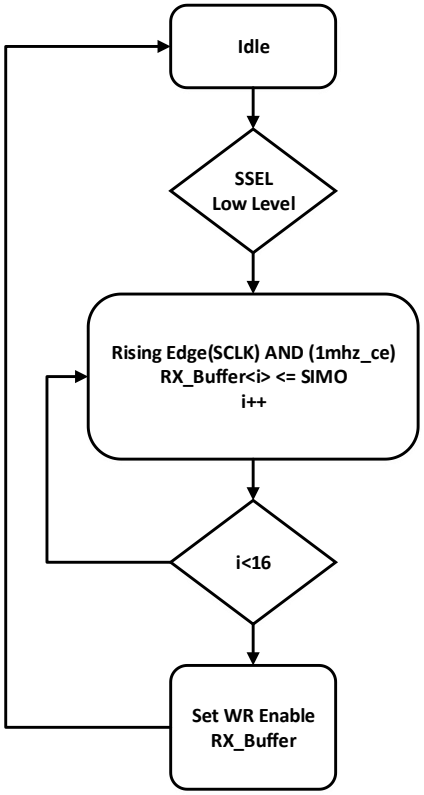
Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

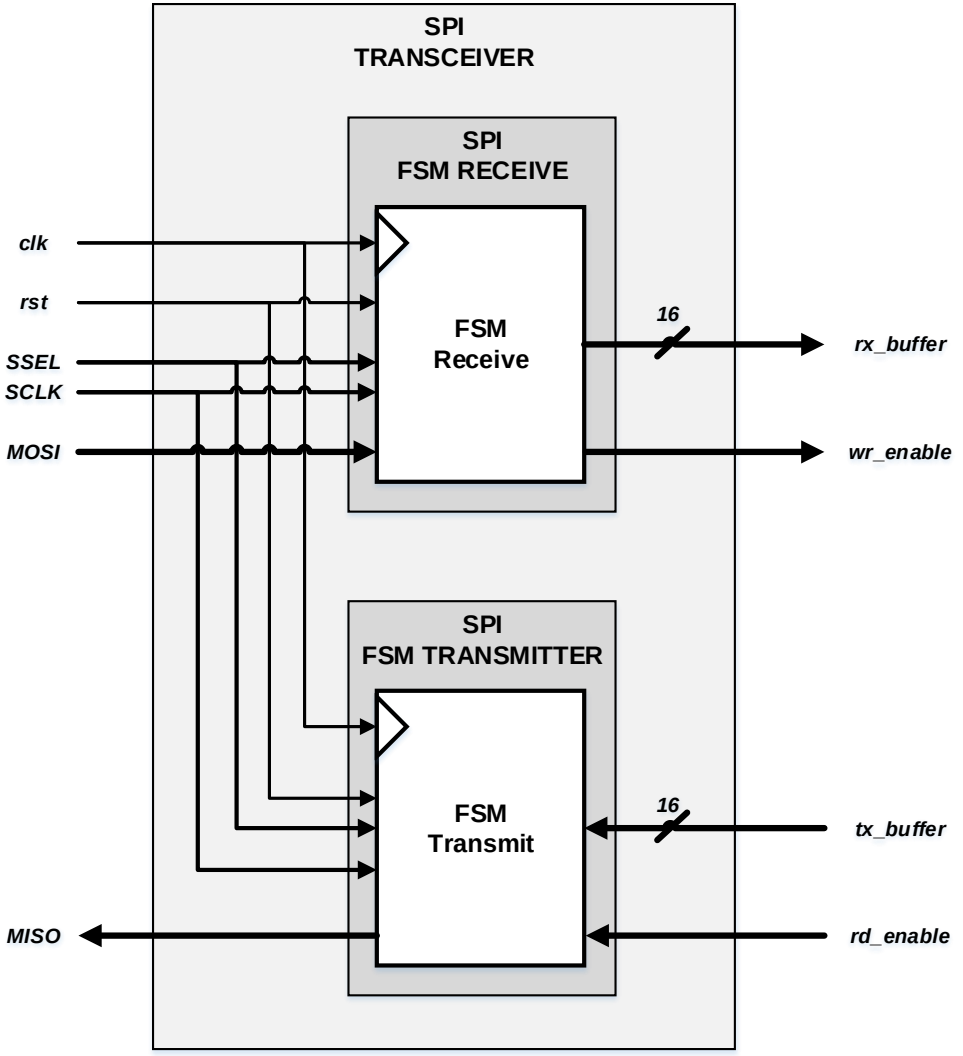
MODE 0 : CPOL = 0 & CPHA = 0



SPI Slave Receiver

- **Récepteur**
  1. **Entrées/Sorties**
  2. **FSM**
  3. **ASM**





Partie #3 : ISE/VHDL

Protocole de communication, configuration et commande moteur

I. **Commandes de configuration**
   1. **Marche/Arrêt**
   2. **Sens de rotation**
   3. **Vitesse**

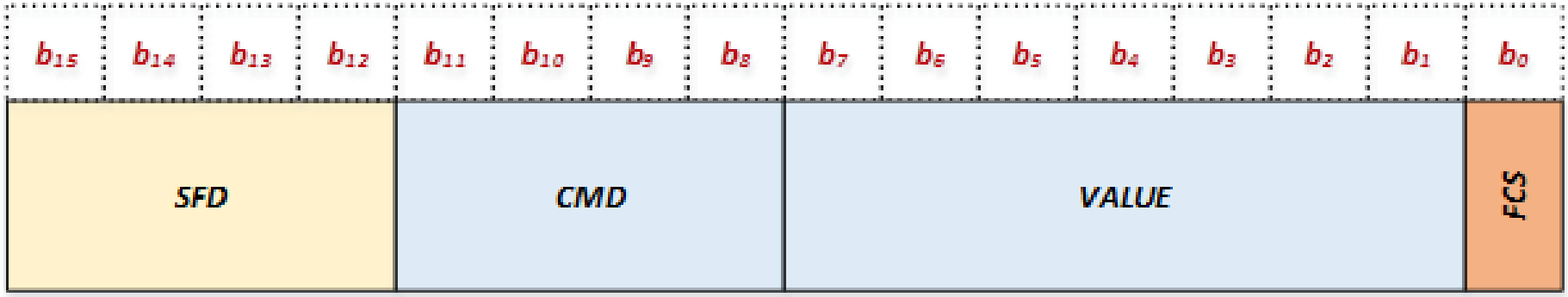
II. **Format du paquet**
   1. **Entête**
   2. **Commande**

III. **Diagramme de séquence**

IV. **FSM/ASM**

Format du paquet de commande

- **Paquet de 2 octets – 16 bits**
- **Entête**
- **Commande + Valeur**
- **Parité**

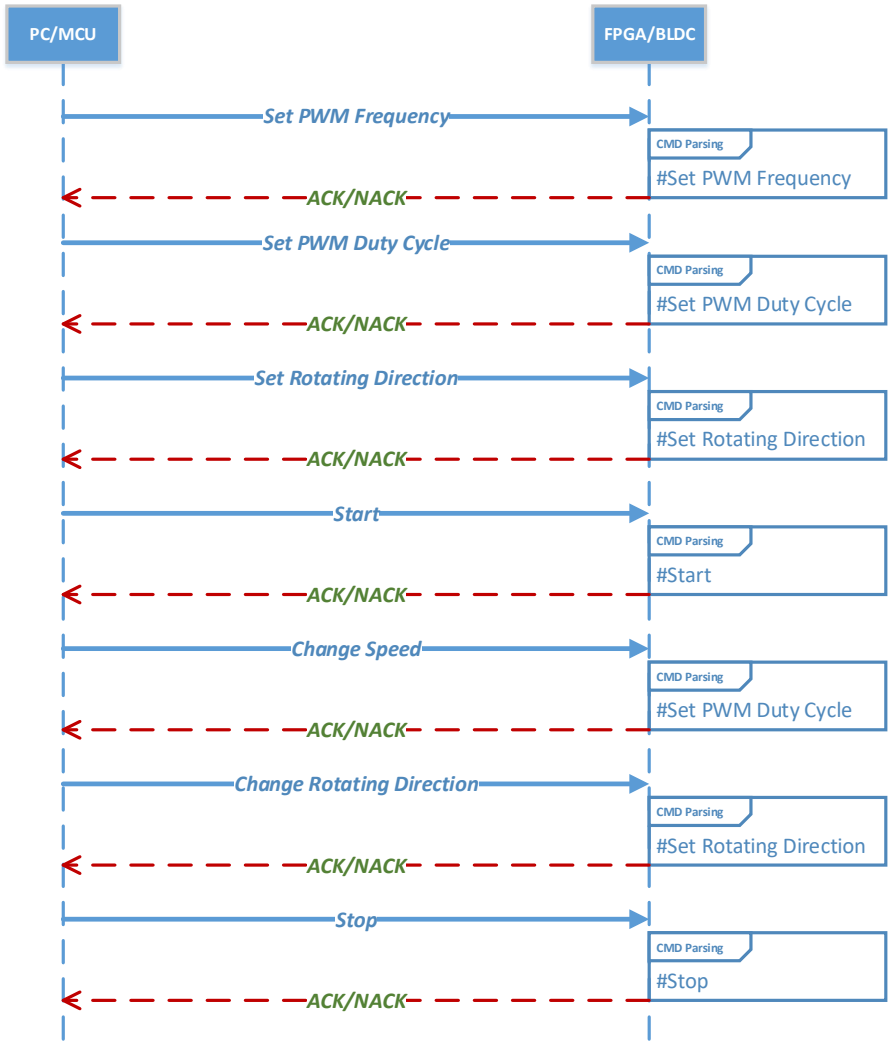


- **SFD** (Start of Frame Delimiter) : c'est un indicateur de début de trame sur 4 bits
- **CMD** : c'est la commande à envoyer au contrôleur de configuration sur FPGA – 4 bits
- **VALUE** : c'est la valeur de la commande à envoyer au contrôleur de configuration sur FPGA – 7 bits
- **FCS** (Frame Check Sequence) : c'est un bit de parité calculé sur le payload CMD/VALUE ($b_{11} - b_1$)

Configuration et commandes moteur

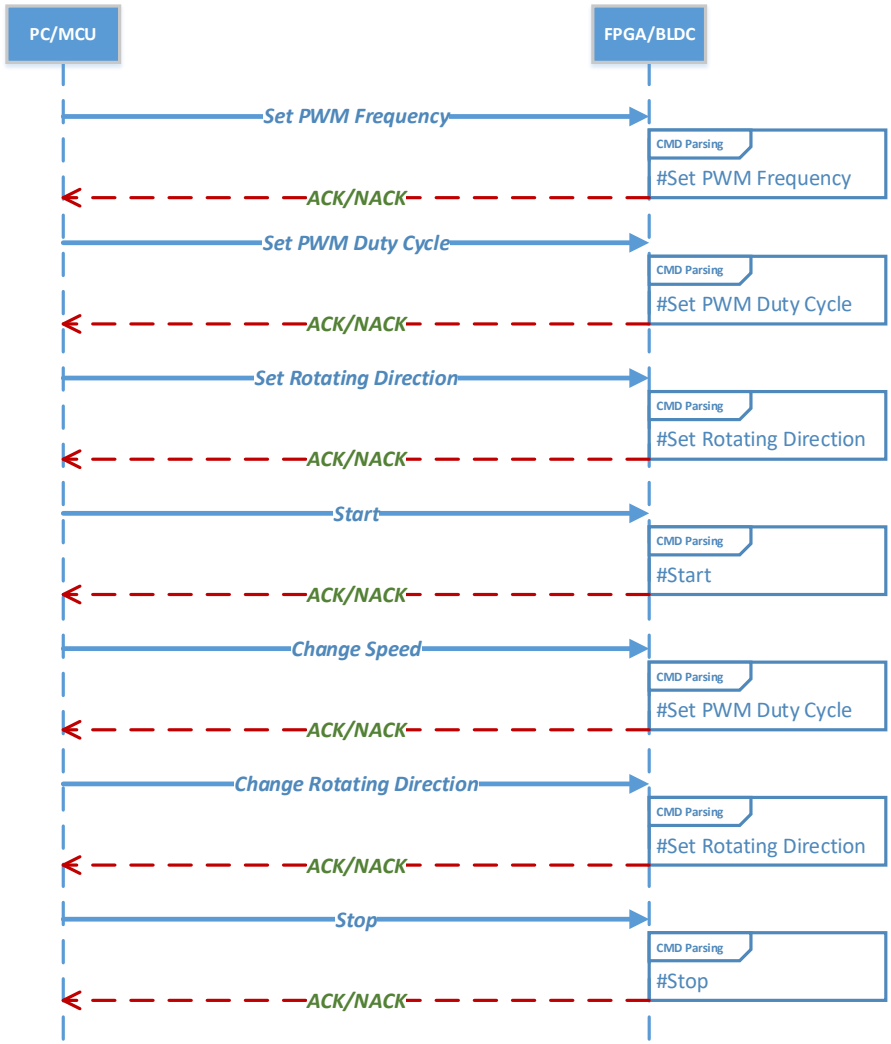
Valeur de CMD b_{11}-b_8	Moteur	Type de commande
0000	1	Réservée
0001	1	Marche/Arrêt
0010	1	Sens de rotation (marche avant / marche arrière)
0011	1	Réservée
0100	1	Configuration de la fréquence du PWM
0101	1	Réservée
0110	1	Réservée
0111	1	Configuration de la vitesse ou le rapport cyclique du PWM
1000	2	Réservée
1001	2	Marche/Arrêt
1010	2	Sens de rotation (marche avant / marche arrière)
1011	2	Réservée
1100	2	Configuration de la fréquence du PWM
1101	2	Réservée
1110	2	Réservée
1111	2	Configuration de la vitesse ou le rapport cyclique du PWM

Diagramme de séquence



- Communication SPI
- Protocole de commande
- Parsing/Décodage du Paquet
- Validation de la commande
  - Entête
  - Parité
  - Commande
  - Valeur de la commande
- Exécution de la commande
- Mécanisme d'accusé de réception
- Gestion des exceptions

Diagramme de séquence



>> Setting PWM Frequency of Motor 0 to 5 ...

Payload : 0xF402

>> Setting PWM Duty Cycle of Motor 0 to 4 ...

Payload : 0xF701

>> Rotating Motor 0 in Clockwise Direction ...

Payload : 0xF202

>> Rotating Motor 0 in Counter Clockwise Direction ...

Payload : 0xF201

>> Starting Motor 0 ...

Payload : 0xF102

>> Stopping Motor 0 ...

Payload : 0xF101

>> Setting PWM Frequency of Motor 1 to 5 ...

Payload : 0xFC03

>> Setting PWM Duty Cycle of Motor 1 to 4 ...

Payload : 0xFF00

>> Rotating Motor 1 in Clockwise Direction ...

Payload : 0xFA03

>> Rotating Motor 1 in Counter Clockwise Direction ...

Payload : 0xFA00

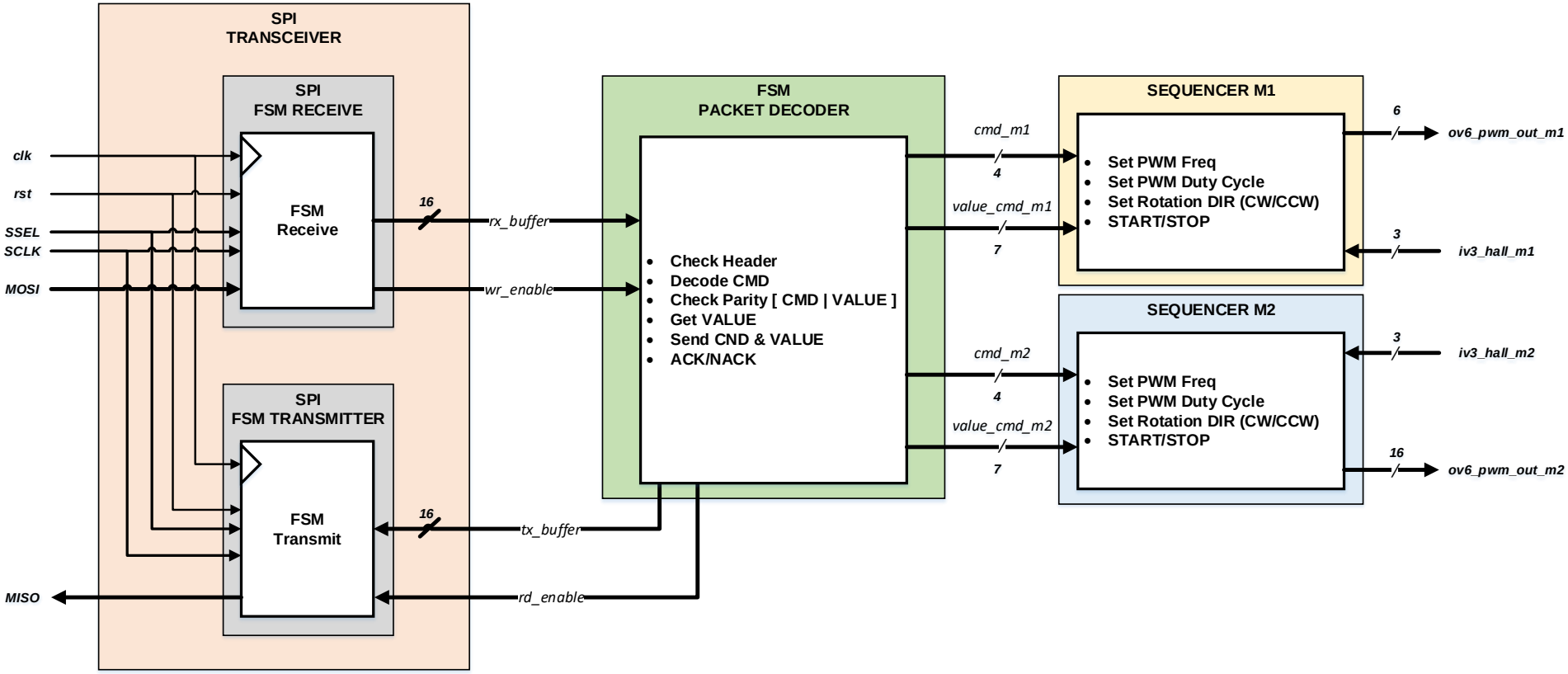
>> Starting Motor 1 ...

Payload : 0xF903

>> Stopping Motor 1 ...

Payload : 0xF900

SPI <-> Décodeur de Paquet <-> Séquenceurs Moteurs



Questions & Discussion