Synthèse et mise en œuvre des systèmes

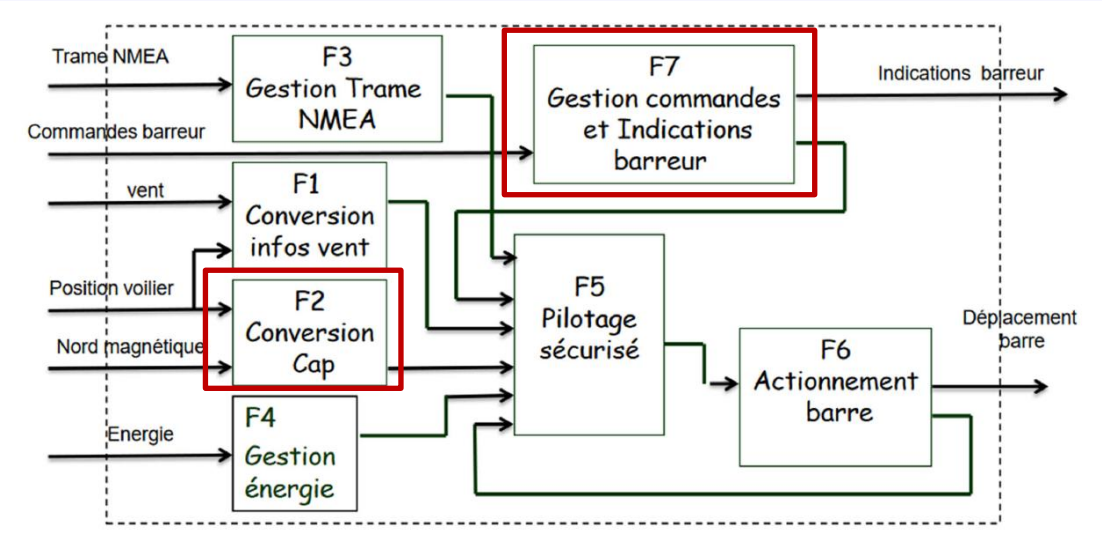
BE- VHDL M2 SME

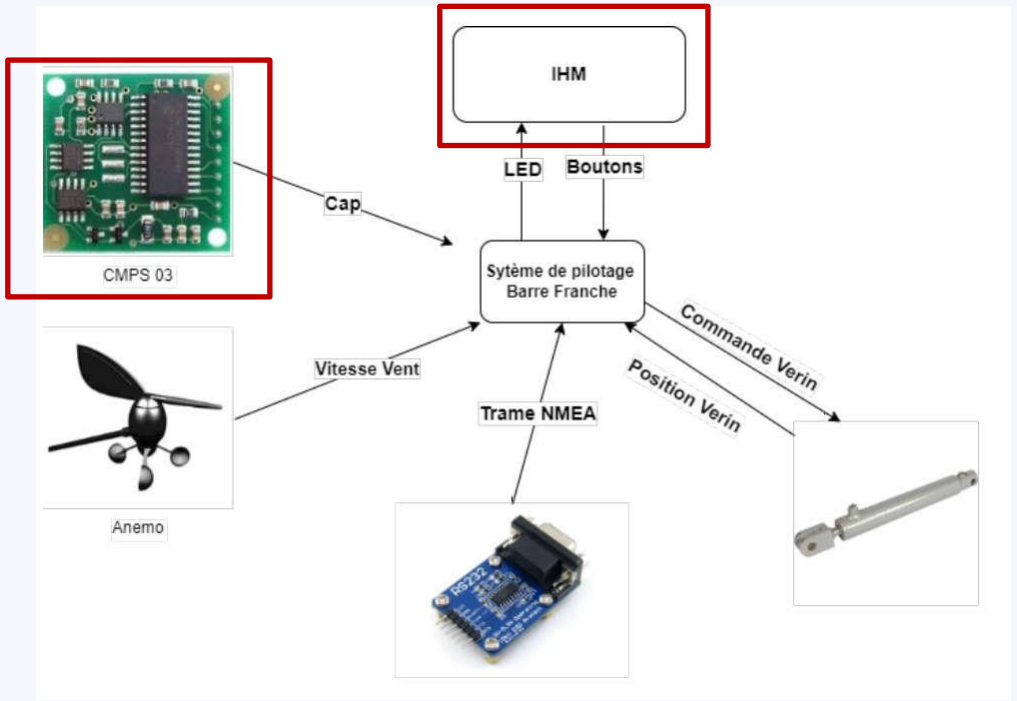
Université Paul Sabatier Toulouse III
HADJ HASSEN Yosri & HAMICI Mohamed Anis

Sommaire

I- Introduction

développer un pilote de barre franche sous la forme d'une puce programmable SOPC (System On Programmable Chip)

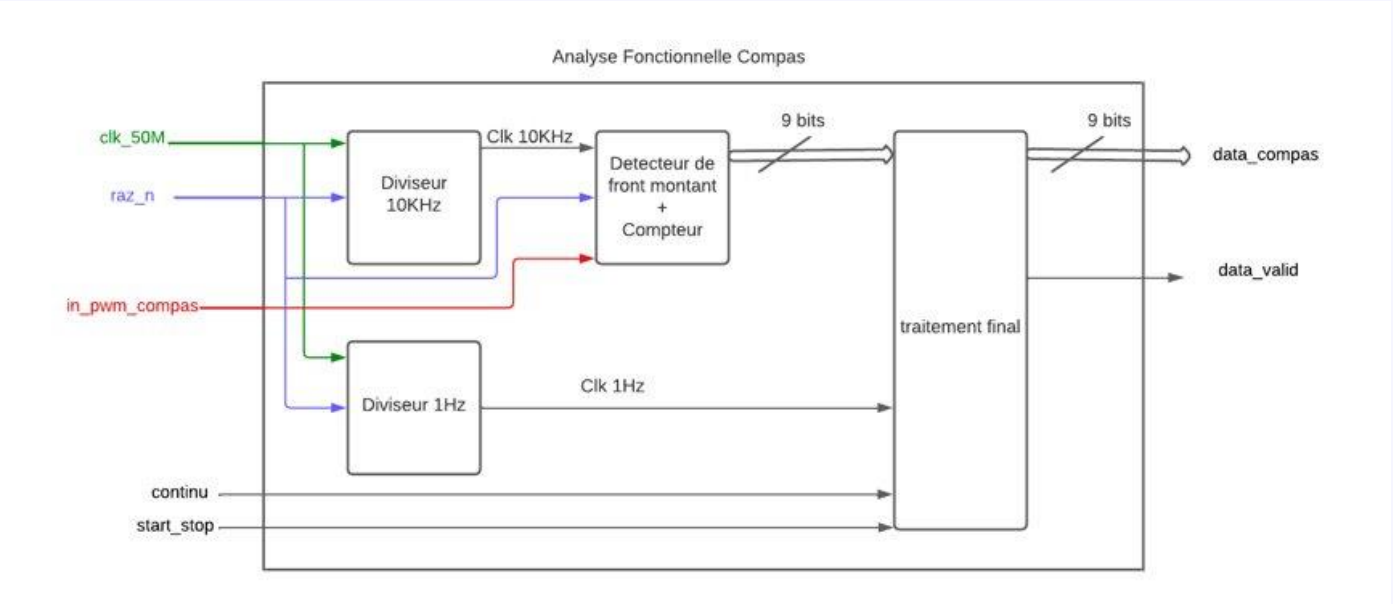


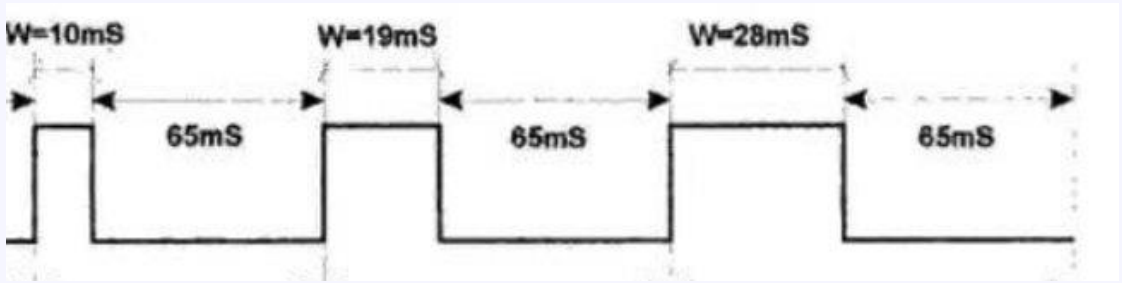


II- Fonction: Gestion du cap

1- Analyse fonctionnelle
2- Implémentation et simulation
3- Intégration SOPC

1- Analyse fonctionnelle





1- Analyse fonctionnelle

Diviseurs de fréquence :

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity Diviseur_10Khz is
port ( clk,reset: in std_logic;
clock_out: out std_logic);
end Diviseur_10Khz;

architecture bhv of Diviseur_10Khz is

signal count: integer:=25000;
signal tmp : std_logic := '0';

begin

process(clk,reset)
begin
if(reset='0') then
count<=1;
tmp<='0';
elsif(clk'event and clk='1') then
count <=count+1;
if (count = 2500) then
tmp <= NOT tmp;
count <= 1;
end if;
end if;
clock_out <= tmp;
end process;

end bhv;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity Diviseur_1hz is
port ( clk,reset: in std_logic;
clock_out: out std_logic);
end Diviseur_1hz;

architecture bhv of Diviseur_1hz is

signal count: integer:=25000000;
signal tmp : std_logic := '0';

begin

process(clk,reset)
begin
if(reset='0') then
count<=1;
tmp<='0';
elsif(clk'event and clk='1') then
count <=count+1;
if (count = 25000000) then
tmp <= NOT tmp;
count <= 1;
end if;
end if;
clock_out <= tmp;
end process;

end bhv;
```

Détecteur de front et compteur :

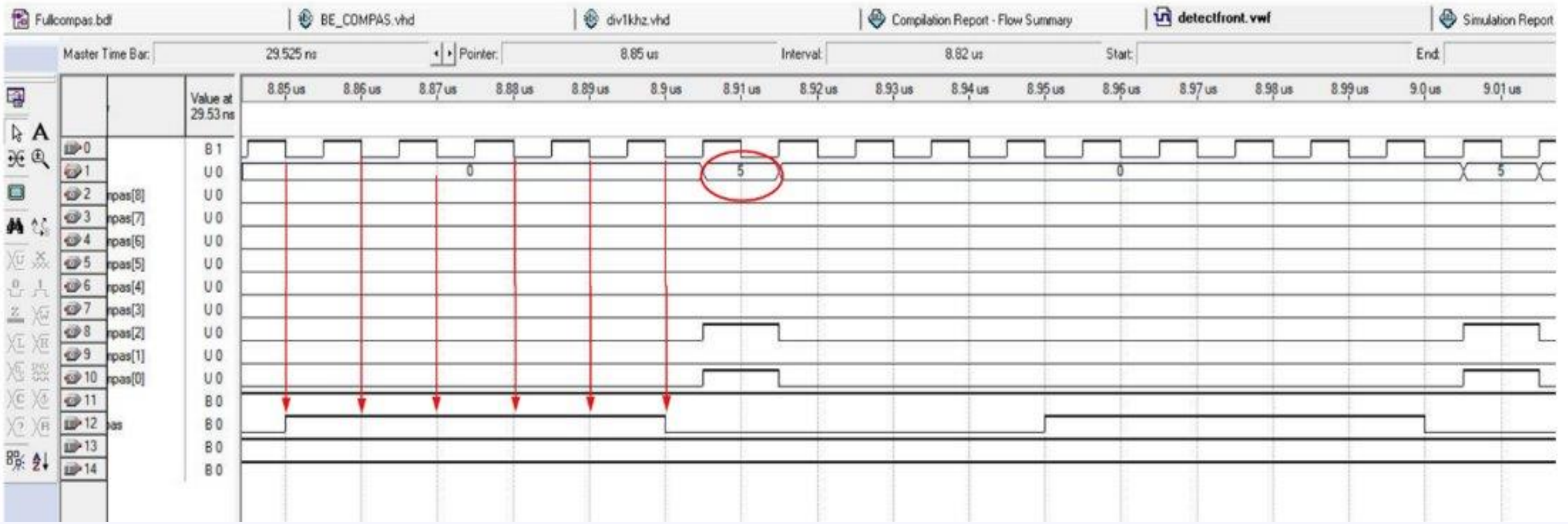
```
begin
  process(clk_50M, raz_n)
  begin
    if raz_n = '0' then
      pulse_width_counter <= 0;
      data_compas_internal <= (others => '0');
    elsif rising_edge(clk_50M) then
      if in_pwm_compas = '1' then
        pulse_width_counter <= pulse_width_counter + 1;
      elsif in_pwm_compas = '0' then
        data_compas_internal <= std_logic_vector(to_unsigned(pulse_width_counter, 9));

        if data_compas_internal /= "000000000" then
        data_compas <= data_compas_internal;
        pulse_width_counter <= 0;
        end if;
      end if;
    end if;
  end process;



end Behavioral;
```

1- Analyse fonctionnelle

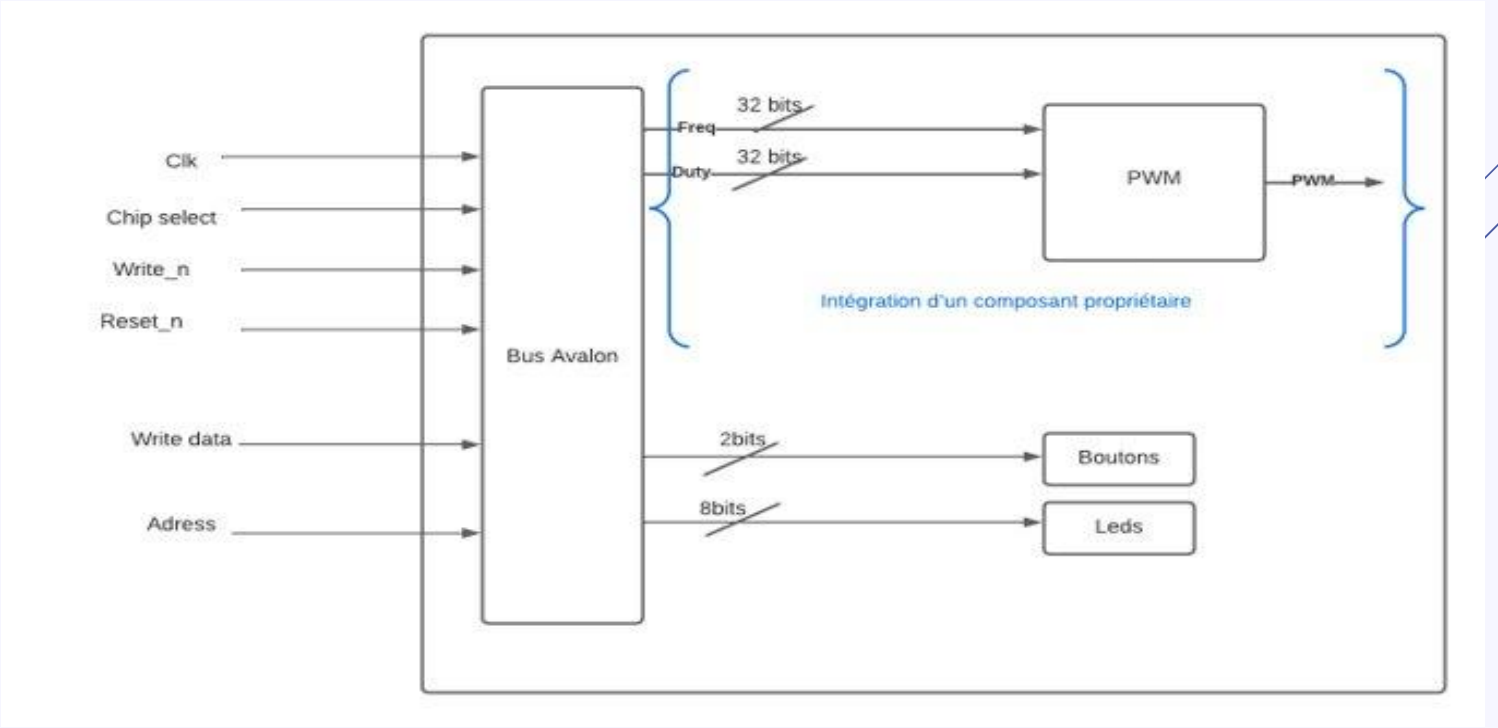
Traitement

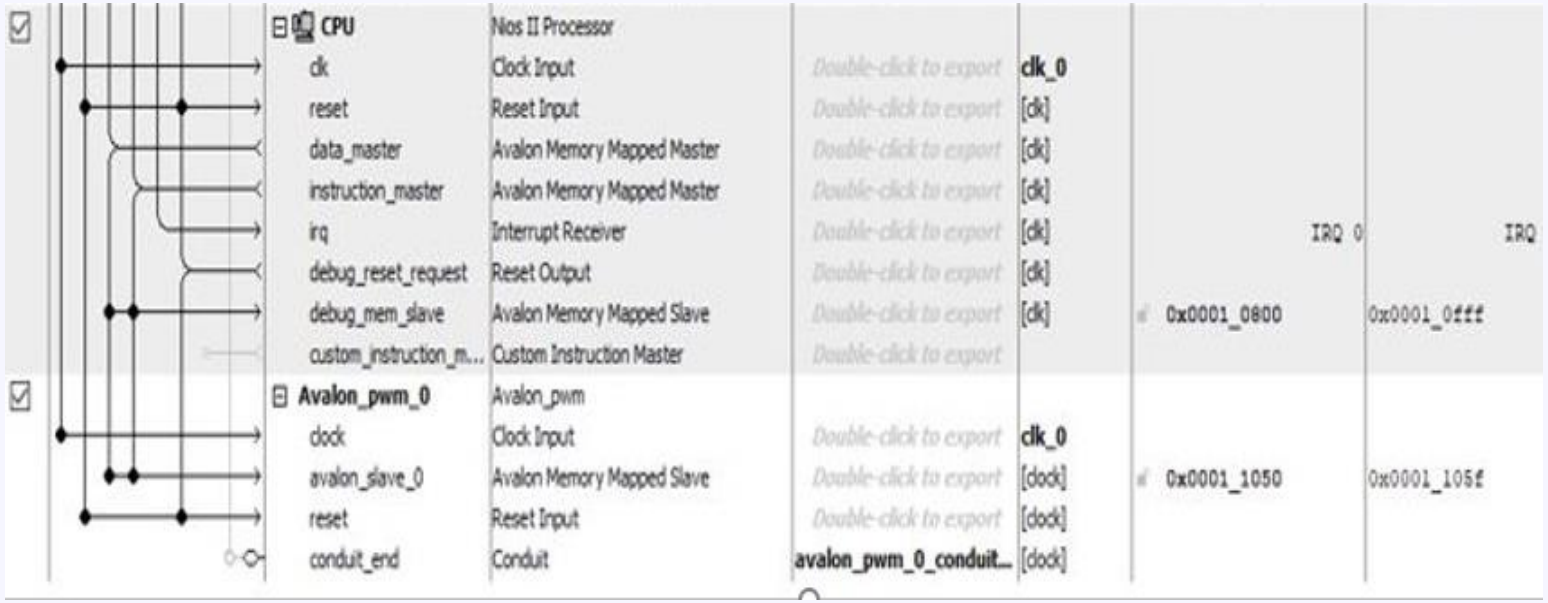
```
begin

  process(clk1hz, raz_n)   --continu a 1
  begin
    if raz_n = '0' then

      data_valid_internal <= '0';

      data_compas<="000000000";

     elsif rising_edge(clk1hz) then

    if continu = '1' then

    data_compas <= data_compas_in;
         data_valid_internal <= '1';
     elsif continu ='0' then            --mode monocoup
         if start_stop ='1' then
       if fin_mesure ='0' then
       data_valid_internal <= '1';
       data_compas  <= data_compas_in;
       fin_mesure <='1' ;
       end if;
         elsif start_stop ='0' then
         data_valid_internal <= '0';
         fin_mesure <='0' ;

         end if;
         end if;
       end if;


  end process;

data_valid <= data_valid_internal;

end Behavioral;
```

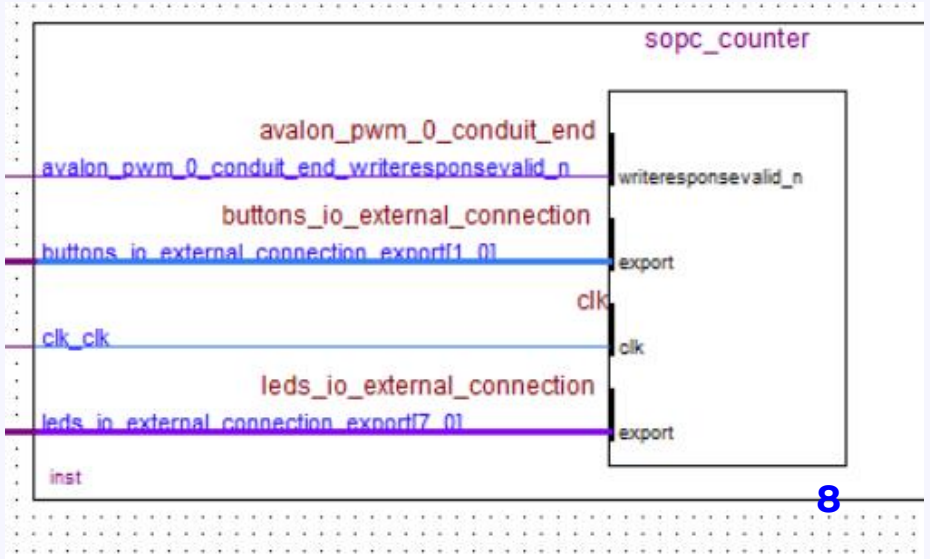
2- Implémentation et simulation



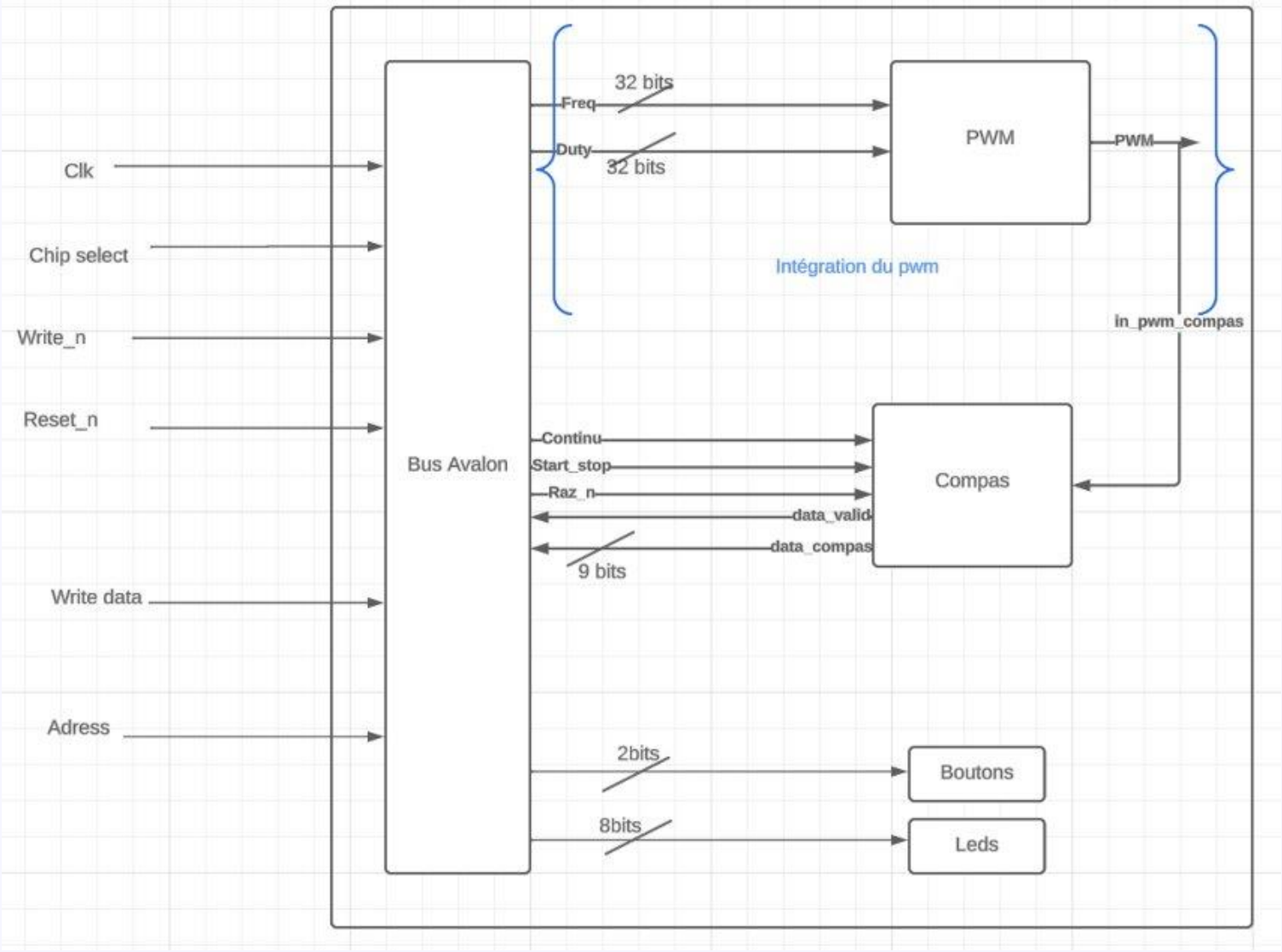
3- Intégration SOPC



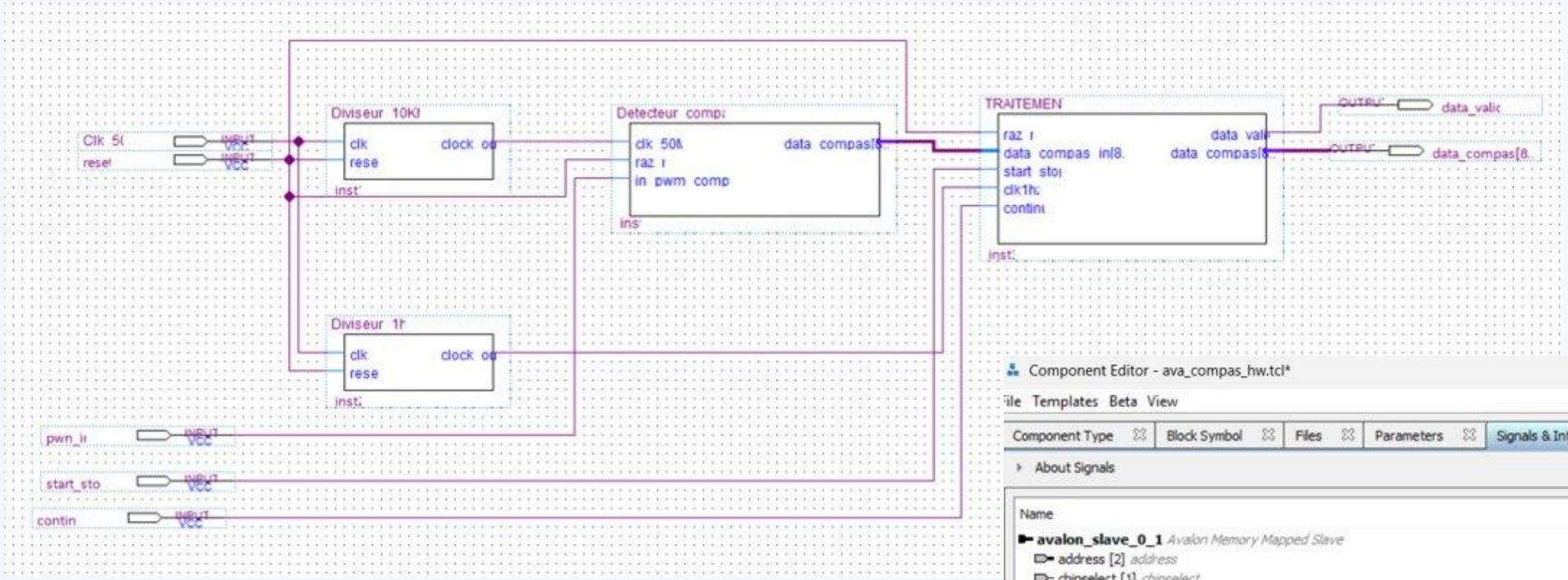


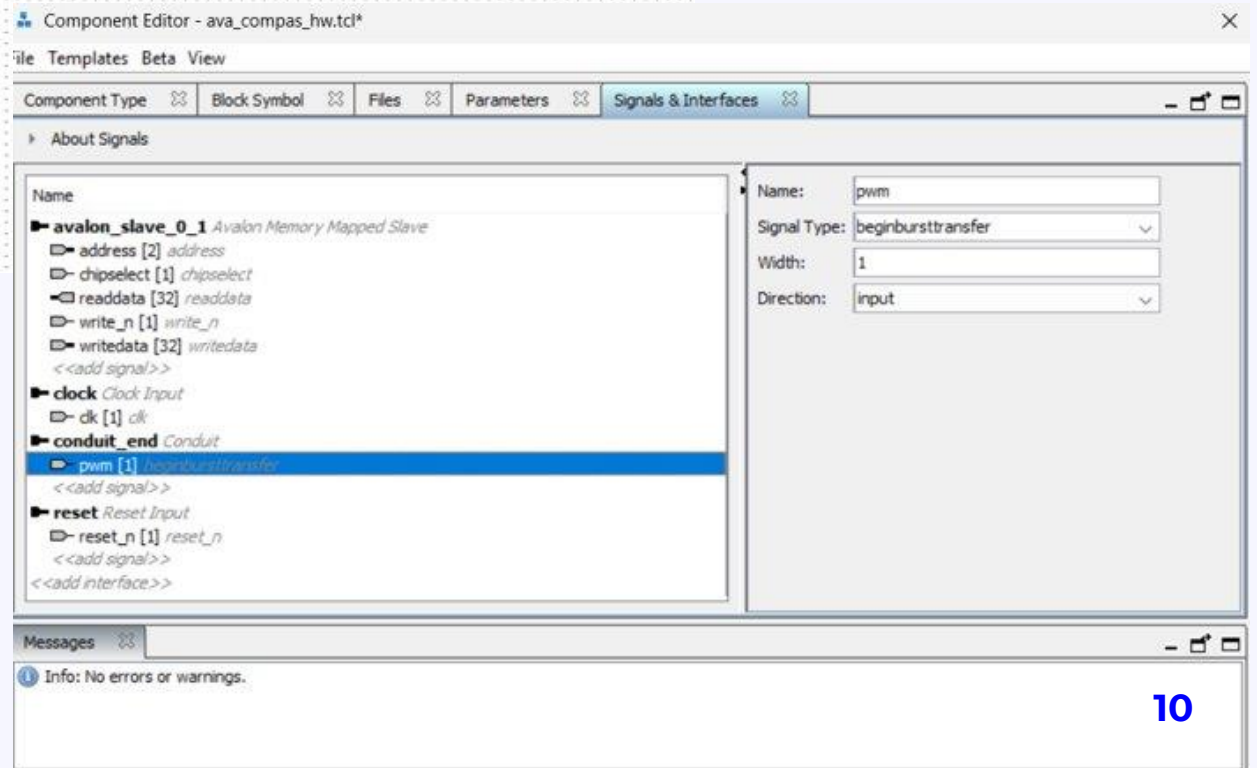


⦿ 3- Intégration SOPC (avec module compas)

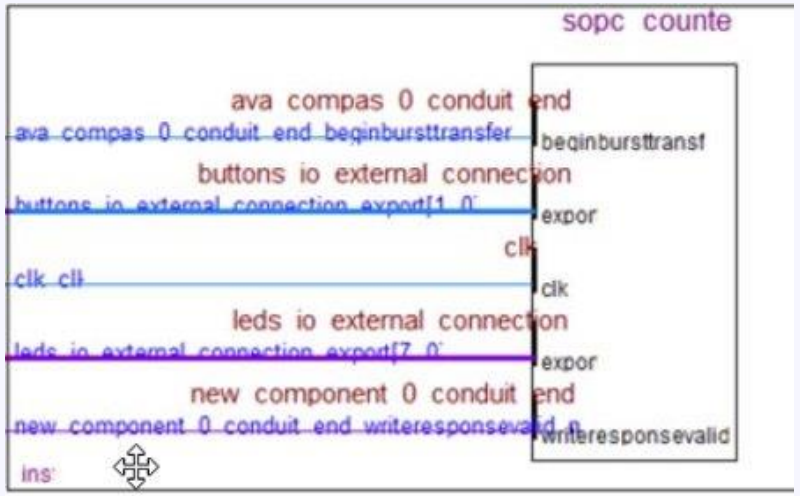


⦿ 3- Intégration SOPC (avec module compas)





3- Intégration SOPC (avec module compas)



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
		clk1	Clock Input	Double-click to export	clk_0			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0000_8000	0x0000_cfff	
		reset1	Reset Input	Double-click to export	[clk1]			
☑		**Buttons_IO**	PIO (Parallel I/O) Intel FPGA IP					
		clk	Clock Input	Double-click to export	**clk_0**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1050	0x0001_105f	
		external_connection	Conduit	**buttons_io_external_co...**				
☑		**LEDS_IO**	PIO (Parallel I/O) Intel FPGA IP					
		clk	Clock Input	Double-click to export	**clk_0**			
		reset	Reset Input	Double-click to export	[clk]			
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1040	0x0001_104f	
		external_connection	Conduit	**leds_io_external_conne...**				
☑		**jtag_uart_0**	JTAG UART Intel FPGA IP					
		clk	Clock Input	Double-click to export	**clk_0**			
		reset	Reset Input	Double-click to export	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1098	0x0001_109f	
		irq	Interrupt Sender	Double-click to export	[clk]			16
☑		**sysid_qsys_0**	System ID Peripheral Intel FPGA IP					
		clk	Clock Input	Double-click to export	**clk_0**			
		reset	Reset Input	Double-click to export	[clk]			
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1090	0x0001_1097	
☑		**CPU**	Nios II Processor					
		clk	Clock Input	Double-click to export	**clk_0**			
		reset	Reset Input	Double-click to export	[clk]			
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
		irq	Interrupt Receiver	Double-click to export	[clk]	IRQ 0	IRQ 31	
		debug_reset_request	Reset Output	Double-click to export	[clk]			
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_0800	0x0001_0fff	
		custom_instruction_m...	Custom Instruction Master	Double-click to export				
☑		**new_component_0**	new_component					
		clock	Clock Input	Double-click to export	**clk_0**			
		avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0001_1070	0x0001_107f	
		reset	Reset Input	Double-click to export	[clock]			
		conduit_end	Conduit	**new_component_0_con...**	[clock]			
☑		**ava_compas_0**	ava_compas					
		clock	Clock Input	Double-click to export	**clk_0**			
		reset	Reset Input	Double-click to export	[clock]			
		conduit_end	Conduit	**ava_compas_0_conduit...**	[clock]			
		avalon_slave_0_1	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0001_1060	0x0001_106f	

⦿ 3- Intégration SOPC (avec module compas)

Registres de communication:

```
--registres d'ecriture

process_write : process (clk, reset_n)
begin
if reset_n = '0' then
   reset <= '0';
   continu <= '0';
   start_stop <= '0';
   elsif clk'event and clk = '1' then
   if chipselect ='1' and write_n = '0' then
      if address = "00" then
         reset <= writedata(0);
         continu <= writedata(1);
         start_stop <= writedata(2);



      end if;
   end if;
end if;
end process;

-- registres de lecture



process_Read : process(address, start_stop, continu, reset,data_compas,data_valid)
BEGIN
if address = "00" then
 readdata <= X"0000000"&"0"&start_stop&continu&reset;
 else
 readdata <= X"00000"&"00"&data_valid&data_compas;

end if;
END PROCESS process_Read ;
```

⦿3- Intégration SOPC (avec module compas)

Implémentation NIOS :

```
100
101 #define freq (unsigned int *) NEW_COMPONENT_0_BASE
102 #define duty (unsigned int *) (NEW_COMPONENT_0_BASE + 4)
103 #define control (unsigned int *) (NEW_COMPONENT_0_BASE + 8)
104
105 #define boutons (volatile char *) BUTTONS_IO_BASE
106 #define leds (unsigned int*) LEDS_IO_BASE
107 unsigned int a,e,f;
108 #define config (volatile int *) AVA_COMPAS_0_BASE
109 #define data_compas (volatile int *) (AVA_COMPAS_0_BASE+4)
110
111 int main()
112 {
113   alt_putstr("Salut ext!\n");   // test si communication OK
114   *freq = 0x40D990; // diviser la clk de 50 M par 4250000 => periode de 85 ms
115   *duty = 0xF4240; // RC = 23% => duty de 20 ms
116   *control = 0x0003;
117   *config =0x0003;
118
119   while (1)
120       {
121
122       alt_putstr("Salut int!\n");
123
124       e=*data_compas & 0x0040;    //0000 0000 0100 0000
125       f=*data_compas & 0xFF80 ;   //1111 1111 1000 0000
126       printf("data_valid = %d ",e);
127       printf("num_out= %d\n",(f-1)); usleep(100000);
128
129       a = *boutons & 3;
130       printf("boutons = %d\n", a);
```

Problems Tasks Console Nios II Console Properties

compas_avalon Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtag_uart_0.jtag

```
boutons = 3
Salut int!
data_valid = 1 num_out= 190
boutons = 3
Salut int!
data_valid = 1 num_out= 190
boutons = 3
Salut int!
data_valid = 1 num_out= 190
boutons = 3
```

III-Fonction: Gestion commandes et indications barreur

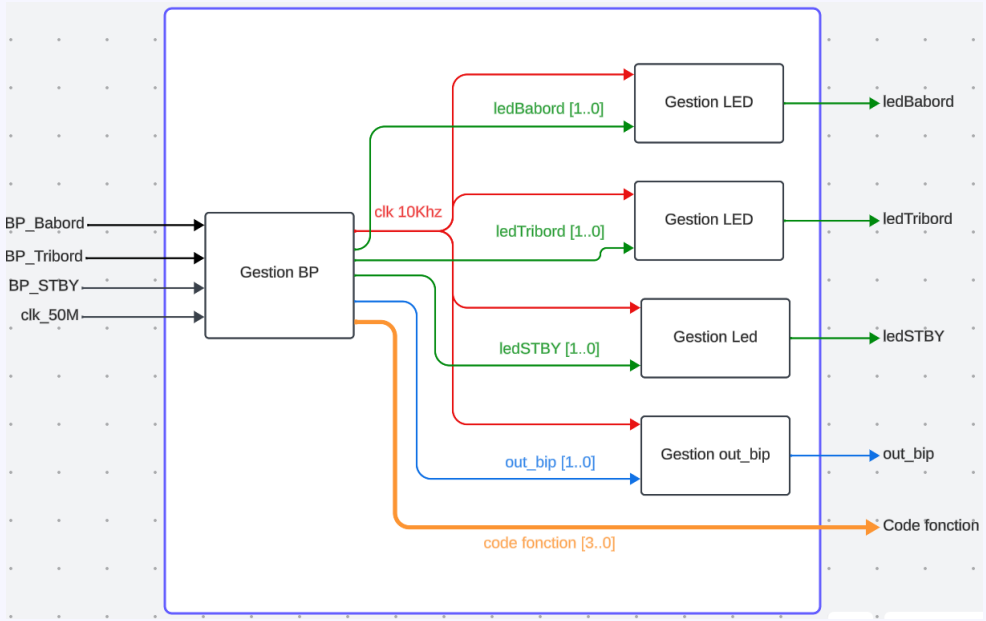
1- Analyse fonctionnelle

1- Analyse fonctionnelle

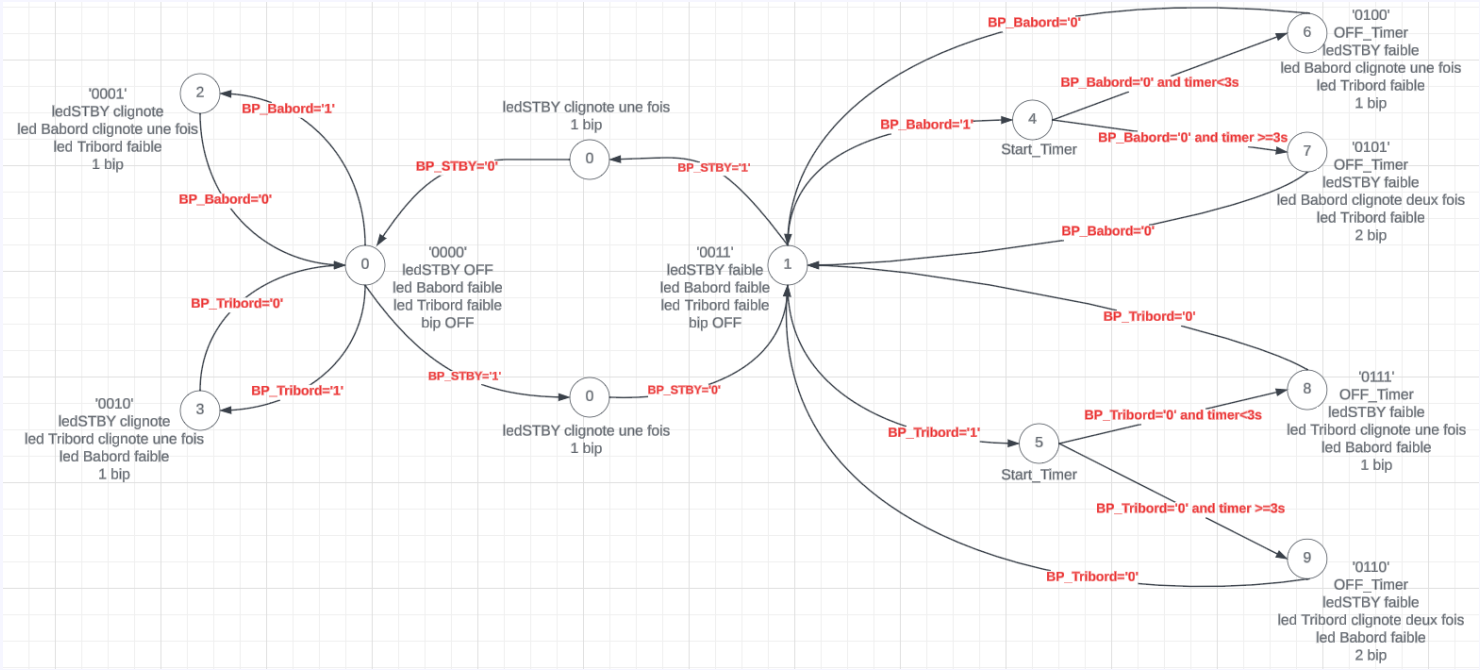
Code_fonction	Action
0000	Pas d'action, Pilote en veille
0001	Mode manuel action vérin babord
0010	Mode manuel action vérin tribord
0011	Mode pilote automatique/cap
0100	Incrément de 1° consigne de cap
0101	Incrément de 10° consigne de cap
0111	Décrément de 1° consigne de cap
0110	Décrément de 10° consigne de cap

1- Analyse fonctionnelle

Gestion des boutons poussoirs

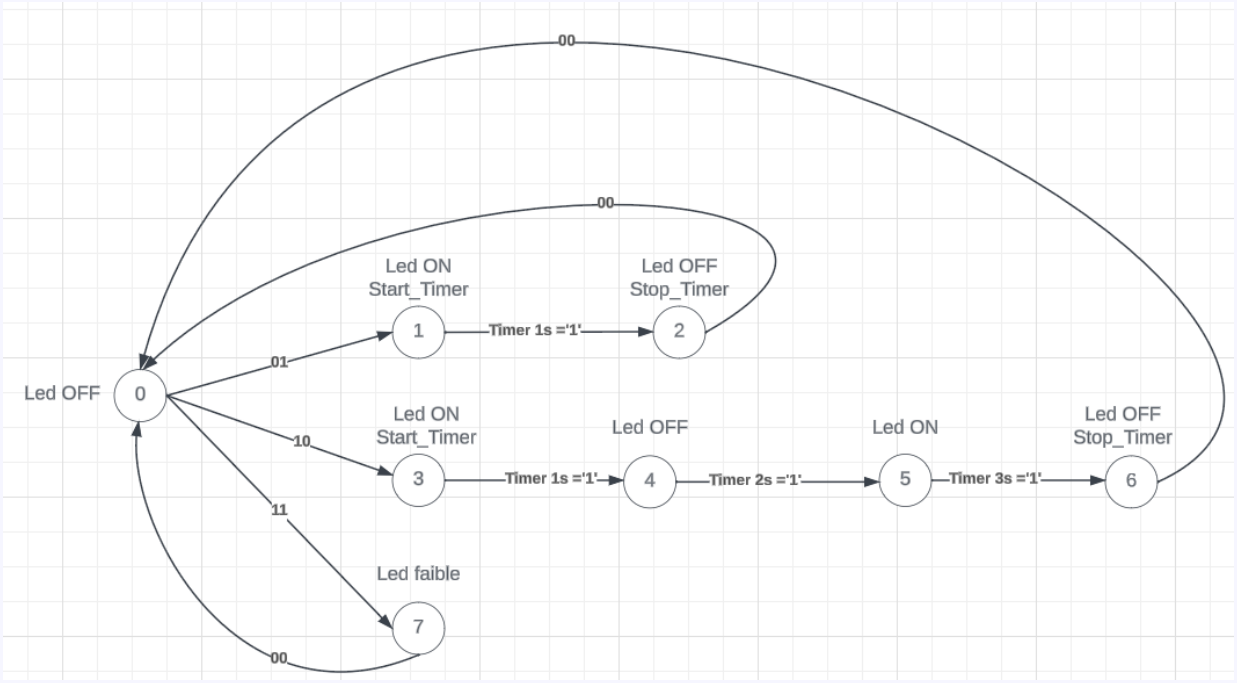


BP	Code LEDs	Code Bip
BP_STBY/Auto	00/11	00/01
BP_Tribord	01	01
BP_Babord	01	01
court sur BP_Babord	01	01
long sur BP_Babord	10	10
court sur BP_Tribord	01	01
long sur BP_Tribord	10	10



1- Analyse fonctionnelle

Gestion des LEDs

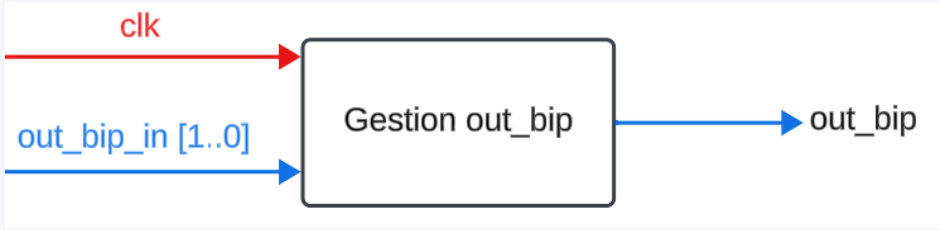


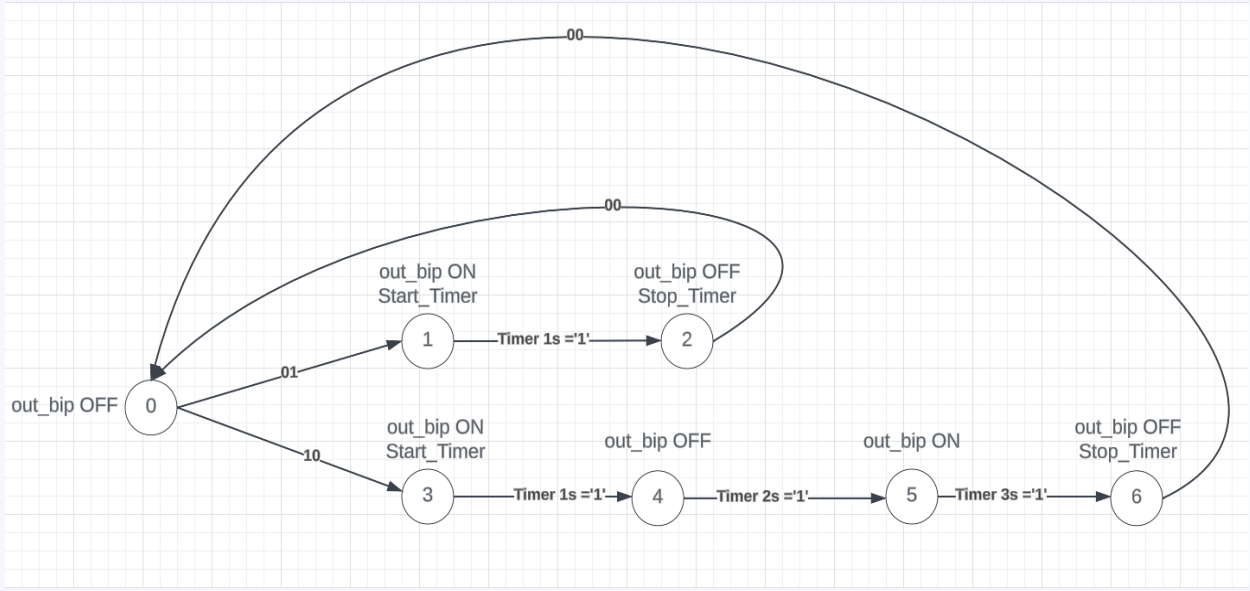
Les modes de fonctionnement

- 00 : LED éteinte
- 01 : LED clignote une fois pendant 1 seconde
- 10 : LED clignote deux fois, chaque clignotement durant 1 seconde
- 11 : LED allumée faiblement

1- Analyse fonctionnelle

Gestion buzzer





Les modes de fonctionnement

- 00 : Buzzer éteint
- 01 : Buzzer émet un bip unique, d'une durée de 1 seconde
- 10 : Buzzer émet deux bips, chacun durant 1 seconde

IV-Conclusion

Merci pour votre attention !