

Rapport TP 03

Samory DIABY

Table des matières

1	Objectif	2
2	Notes	2
3	Réalisations	2
3.1	Nouveau compteur	2
3.1.1	Schéma RTL du nouveau compteur	2
3.1.2	Code VHDL du nouveau compteur	3
3.1.3	Testbench du nouveau compteur	5
3.2	Clignotement des LEDs	7
3.2.1	Schémas RTL	7
3.2.2	Entrées / sorties du système	8
3.2.3	Analyse de rapport de synthèse	9
3.2.4	Analyse de rapport de timing	9
3.3	Démo	10

1 Objectif

L'objectif de ce TP est de réaliser une architecture permettant de faire clignoter deux LEDs RGB en rouge, vert et bleu. Le pilotage des LEDs se fera à l'aide de machines à états.

2 Notes

Pour ce tp, j'ai assumé que les compteurs étaient codés sur 28 bits.

3 Réalisations

3.1 Nouveau compteur

3.1.1 Schéma RTL du nouveau compteur

Pour réaliser le circuit permettant de faire clignoter les 2 leds RGB de la carte j'ai d'abord dessiné le schéma RTL ci-dessous qui permet de compter le nombre de fois où la sortie du compteur **counter_unit** passe à '1'. Ce compteur (**counter_unit**) passe à '1' en fonction d'un paramètre générique donné en entrée à notre circuit, et va définir le temps des états allumé ou éteint des leds.

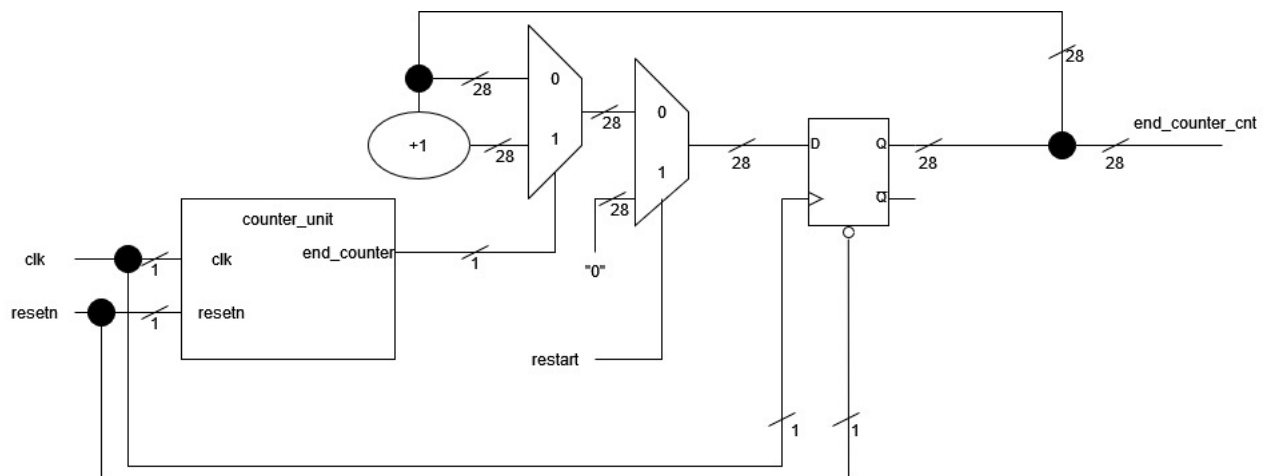


FIGURE 1 – Schéma RTL du compteur de "end_counter" (Q2)

3.1.2 Code VHDL du nouveau compteur

On retrouve ci-dessous le **code VHDL** associé au schéma précédent :

```
1 entity tp_fsm is
2   generic (
3       --vous pouvez ajouter des parameres generics ici
4       limit : unsigned(27 downto 0) := to_unsigned(8, 28) -- use in
          counter_unit
5   );
6   port (
7       clk          : in std_logic;
8       resetn       : in std_logic;
9       restart      : in std_logic;
10      end_counter_cnt : out std_logic_vector(27 downto 0)
11  );
12 end tp_fsm;
13
14 architecture behavioral of tp_fsm is
15     .
16     .
17     .
18
19     -- counter
20     -- intern_end_counter_cnt -> Q output of the flip flop in RTL
21     signal intern_end_counter_cnt : std_logic_vector(27 downto 0) := (
22         others => '0');
23
24     -- muxes cmd signals
25     signal cmd_cnt_incr : std_logic := '0'; -- will allow to choose to
26         increment the counter or not
27     signal cmd_cnt_restart : std_logic := '0'; -- will allow to choose to
28         increment the counter or not
29
30     -- counter_unit declaration
31     component counter_unit
32     generic(
33         limit : unsigned(27 downto 0)
34     );
35     port (
36         clk          : in std_logic;
37         resetn       : in std_logic;
38         end_counter  : out std_logic
39     );
40 end component;
41
42 begin
43
44     LED_COUNTER : counter_unit
45     generic map (
46         limit => limit -- set this value to change the time between LED
47             blinks
48     )
49     port map (
50         clk => clk,
```

```

47         resetn => resetn,
48         end_counter => cmd_cnt_incr
49     );
50
51
52 process(clk,resetn)
53 begin
54     if(resetn='1') then
55
56         current_state <= idle;
57
58         -- reset counter
59
60
61     elsif(rising_edge(clk)) then
62
63         current_state <= next_state;
64
65         --a completer avec votre compteur de cycles
66         -- counter incr mux
67         if cmd_cnt_incr = '1' then
68             intern_end_counter_cnt <= std_logic_vector(unsigned(
69                 intern_end_counter_cnt) + 1);
70         -- else -> keep the prevoius value of the signal
71         end if;
72
73         -- counter restart mux
74         if cmd_cnt_restart = '1' then
75             intern_end_counter_cnt <= (others => '0');
76         -- else -> keep the prevoius value of the signal
77         end if;
78     end if;
79 end process;
80
81 .
82 .
83 .
84
85 -- combinatory logic
86 cmd_cnt_restart <= restart;
87 -- cmd_cnt_incr -> set by the counter_unit
88
89 -- output
90 end_counter_cnt <= intern_end_counter_cnt;
91
92 end behavioral;

```

3.1.3 Testbench du nouveau compteur

On retrouve ci-dessous le **testbench** associé au code précédent :

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity tb_tp_fsm is
6 end tb_tp_fsm;
7
8 architecture behavioral of tb_tp_fsm is
9
10     signal resetn          : std_logic := '0';
11     signal clk              : std_logic := '0';
12     signal restart         : std_logic := '0';
13     signal end_counter_cnt : std_logic_vector(27 downto 0) := (others =>
14         '0');
15
16     -- Les constantes suivantes permette de definir la frequence de l'
17     horloge
18     constant hp            : time := 5 ns;  --demi periode de 5ns
19     constant period       : time := 2*hp;  --periode de 10ns, soit une
20     frequence de 100Hz
21
22     component tp_fsm
23     generic(
24         limit : unsigned(27 downto 0)
25     );
26     port (
27         clk          : in std_logic;
28         resetn       : in std_logic;
29         restart       : in std_logic;
30         end_counter_cnt : out std_logic_vector(27 downto 0)
31     );
32 end component;
33
34 begin
35     dut: tp_fsm
36     generic map (
37         limit => to_unsigned(4, 28)
38     )
39     port map (
40         clk => clk,
41         resetn => resetn,
42         restart => restart,
43         end_counter_cnt => end_counter_cnt
44     );
45
46     --Simulation du signal d'horloge en continue
47     process
48     begin
49         wait for hp;
50         clk <= not clk;
```

```
49  end process;
50
51  process
52  begin
53
54      resetn <= '1';
55      wait for period*5;
56      resetn <= '0';
57
58      --a completer
59      wait;
60
61  end process;
62
63
64 end behavioral;
```

3.2.1 Schémas RTL

Pour pouvoir faire clignoter les 2 LEDs RGB, j'utilise le résultat de comptage **intern_end_counter_cnt (iecc)** pour déterminer le nombre de clignotements. 1 Clignotement est équivalent à 1 cycle on/off des LEDs qui est représenté par 1 cycle tous les 2 iecc (1 demi-cycle à chaque incrément de iecc). Pour 1 état de LED, les 3 cycles de clignotements seront donc (premier cycle de clignotement en partant de 0) :

$$\text{cycle 1} \begin{cases} iecc = 0 & : LED \text{ off} \\ iecc = 1 & : LED \text{ on} \end{cases}$$

$$\text{cycle 2} \begin{cases} iecc = 2 & : LED \text{ off} \\ iecc = 3 & : LED \text{ on} \end{cases}$$

$$\text{cycle 3} \begin{cases} iecc = 4 & : LED \text{ off} \\ iecc = 5 & : LED \text{ on} \end{cases}$$

- Nombre impair \rightarrow on
- Nombre pair \rightarrow off

7

Le changement d'état dans la FSM se fait lorsque le compteur **iecc** a fini de compter. On compare iecc à 5 et on attend que **counter_unit** termine de compter son dernier cycle pour éviter de changer d'état trop tôt. Autrement dès le passage à 5 de iecc, l'état changerait et on réinitialiserait le compteur au front montant suivant soit 10ns après le passage à 5.

Voici le schéma de la FSM servant à contrôler l'état des LEDs :

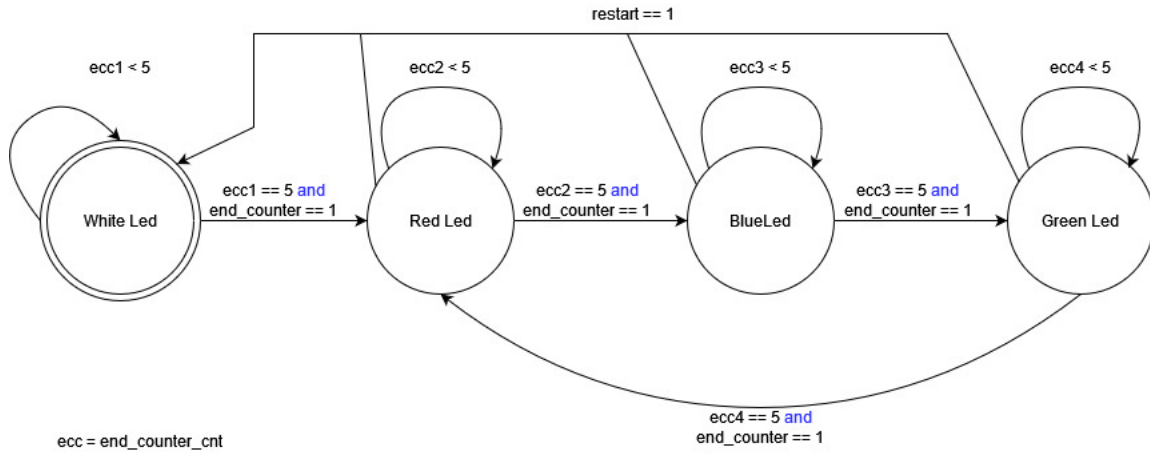


FIGURE 3 – Schéma RTL de la FSM de clignotement des LEDs

On peut voir sur le schéma RTL que l'on change d'état si :

- **iecc** à compté jusqu'à 5 et **counter_unit** vaut 1
- le bouton **restart** est pressé
- le bouton **resetn** est pressé

Dans le dernier cas, on reste dans l'état actuel.

3.2.2 Entrées / sorties du système

On a 9 entrées / sorties dans notre système qui sont :

- Entrées : clk, resetn, restart
- Sorties : **LED0** (led0_b, led0_g, led0_r) et **LED1** (led1_b, led1_g, led1_r)

3.2.3 Analyse de rapport de synthèse

En observant le rapport de synthèse, on retrouve notre machine à états :

State	New Encoding	Previous Encoding
white_led	00	00
red_led	01	01
blue_led	10	10
green_led	11	11

FIGURE 4 – FSM rapport de synthèse

On retrouve bien nos 4 états (LED blanche, rouge, verte et bleue) et il est indiqué que cette FSM est stocké dans un registre 2 bits nommé **FSM_sequential_current_state_reg**.

Dans le *schematic* on retrouve également la *counter_unit* qui est placé dans un module relié au reste du circuit. On a également le registre 28 bits **iecc** correspondant à notre compteur de *end_counter*.

3.2.4 Analyse de rapport de timing

On remarque sur le rapport de timing qu'il n'y a aucune violation (pas de violation de setup ni de hold) :

Design Timing Summary									
WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	
4.030	0.000	0	86	0.244	0.000	0	86	4.500	

FIGURE 5 – Rapport de timing

Chemin critique

D'après vivado, le chemin critique part du registre de comptage interne **iecc** et revient dans ce même registre. Cela correspondrait au circuit rouge sur le schéma ci-dessous :

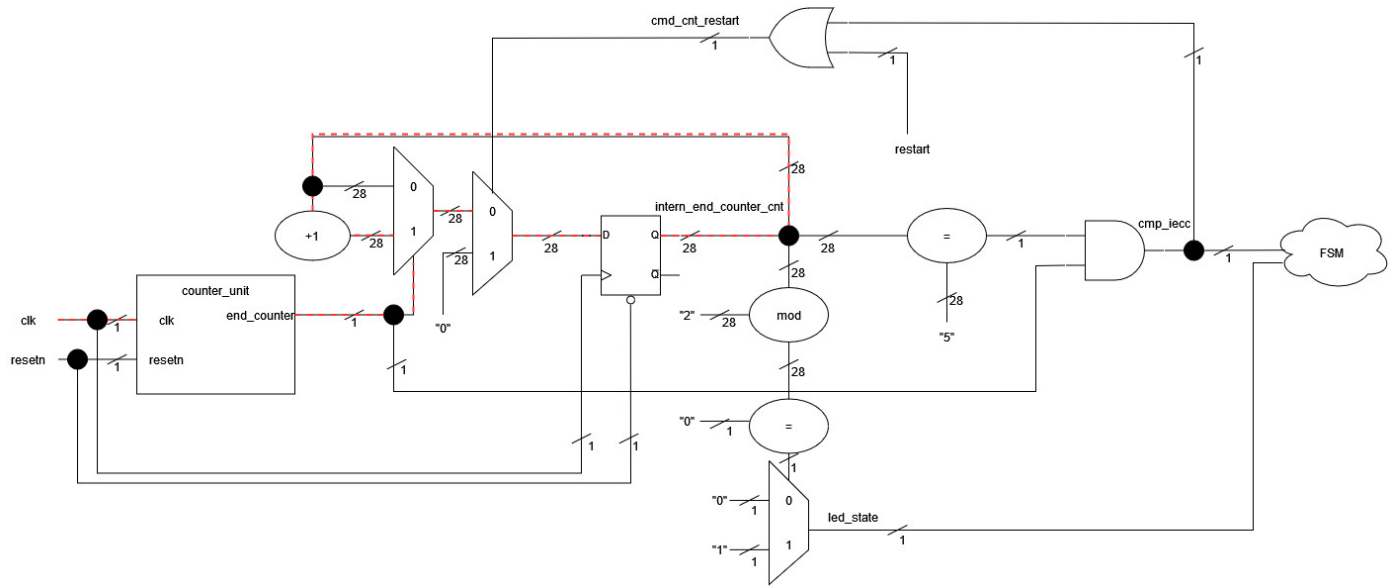


FIGURE 6 – Rapport de timing

3.3 Démo

On peut voir sur le chronogramme ci-dessous que le changement d'état de la FSM s'effectue bien lors de la remise à "0" du compteur et les LEDs pour les états "blanc" et "rouge" sont allumées correctement.

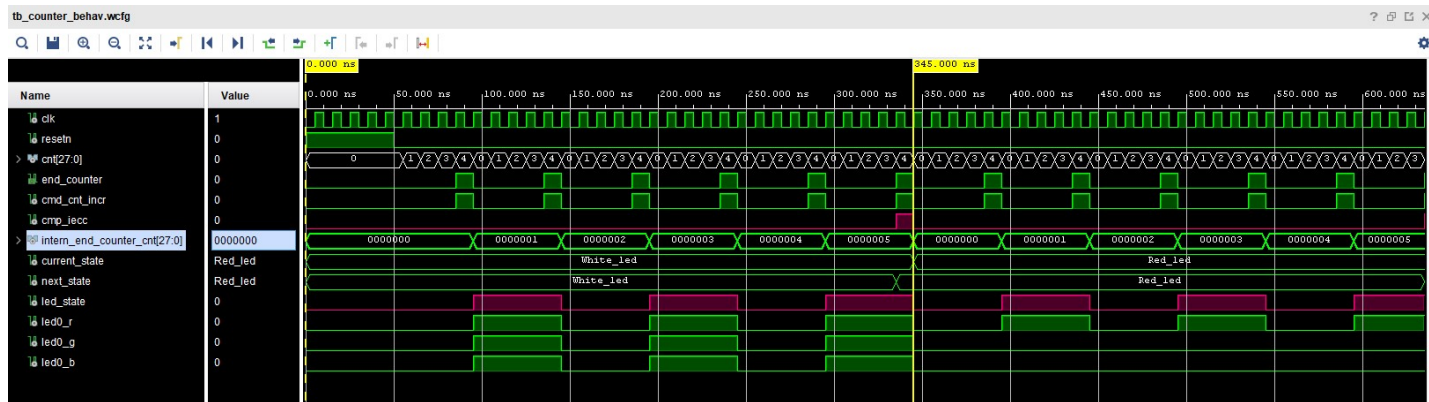


FIGURE 7 – Rapport de timing