

## Segment 2

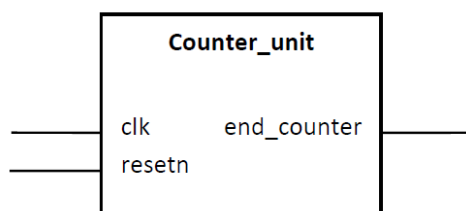
## TP3 : FSM

## Objectif

L'objectif de cet TP est faire clignoter une LED en utilisant un compteur de temporisation. Un compteur de temporisation permet de compter le nombre de coup d'horloge nécessaire pour attendre un temps voulu. En connaissant la fréquence de l'horloge il est possible de déterminer combien de périodes d'horloge il faut compter pour attendre 3 secondes par exemple.

## Questions

1. Dans un fichier *.vhd*, créez un module *Counter\_unit* à partir du compteur du TP1. Le module prendra en entrée un signal d'horloge et de resetn, et donnera en sortie le signal *end\_counter*. Utilisez un paramètre *generic()* pour définir le nombre de coup d'horloge à compter. Le code de *Counter\_unit* ne sera plus modifié ensuite.



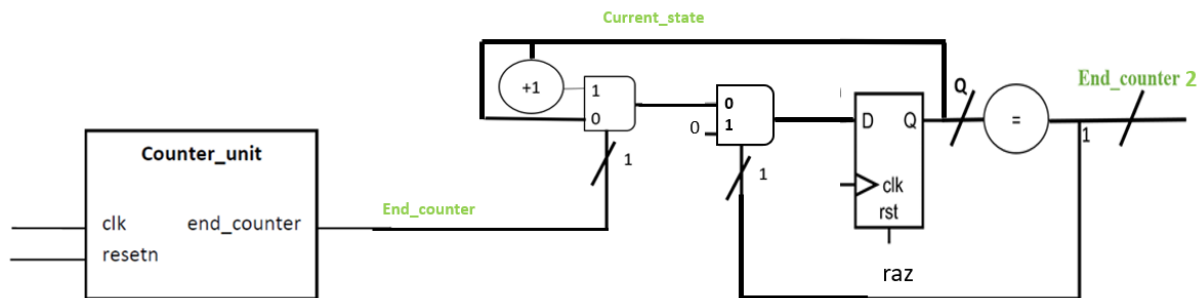
```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.numeric_std.all;
5
6  entity counter_unit is
7  generic (
8      max_count : integer := 10;
9      nb_bit : integer := 4
10 );
11 port (
12     clk          : in std_logic;
13     resetn       : in std_logic;
14
15     end_counter  : out std_logic
16 );
17 end counter_unit;
18
19 architecture behavioral of counter_unit is
20
21     signal Q : std_logic_vector((nb_bit-1) downto 0);
22     signal end_count : std_logic;
23
24 begin
25     --Partie sequentielle
26     process(clk,resetn)
27     begin
28         if(resetn = '1') then
29             Q <= (others => '0');
30
31         elsif(rising_edge(clk)) then
32
33             Q <= Q+ 1;
34             if(end_count= '1') then
35                 Q <= (others => '0');
36
37             end if;
38         end if;
39
40     end process;
41
42     --Partie combinatoire
43     end_count <= '1' when (Q =(max_count)-1)
44         else '0';
45
46     end_counter <= end_count;
47
48 end behavioral;

```

Dans ce module *Counter\_unit*, nous avons utilisé un paramètre générique *max\_count* pour définir le nombre à compter. Le compteur *counter\_unit* est une valeur non signée de 4 bits qui compte jusqu'à *max\_count* - 1. Lorsque le compteur atteint *max\_count* - 1, le signal *end\_counter* est mis à '1' pour indiquer que le comptage est terminé.

2. En schéma RTL, créez un compteur du signal *end\_counter*. Ce compteur doit permettre de déterminer le nombre de cycles allumé/éteint qui ont été effectués par la LED. Le compteur doit pouvoir être remis à 0, maintenir sa valeur actuelle ou s'incrémenter.



Le passage à 1 d'*end\_counter 2* avec le paramètre générique permet de définir le temps des états des leds(allumé/éteint). Le nombre de cycles allumé/éteint dans notre tp est de 6. On a trois étapes ou cycle d'allumer et trois d'états d'éteints.

3. Ecrivez un code VHDL décrivant ce compteur de cycle, vous utiliserez le module *Counter\_unit*.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith;

entity tp_fsm is
    generic (
        max_count : integer :=4;
        nb_bit : integer := 3;
        nbre_cycle : positive :=6 --cycle de clignotement allumé et éteint
    );
    port (
        clk          : in std_logic;
        resetrn      : in std_logic;
        --a completer
        --restart     : in std_logic;
        end_counter2 : out std_logic;
        led_out_R    : out std_logic;
        led_out_B    : out std_logic;
        led_out_V    : out std_logic
    );
end tp_fsm;

architecture behavioral of tp_fsm is

    signal raz : std_logic;
    signal end_count : std_logic;
    signal end_count2 : positive :=0;

    --Declaration de l'entite a tester
    component counter_unit
        generic (
            max_count : integer :=4;
            nb_bit : integer := 3
        );
        port (
            clk          : in std_logic;
            resetrn      : in std_logic;

```

```

        end_counter    : out std_logic
    );
end component;

begin

compteur : counter_unit
generic map (
    max_count => 4,
    nb_bit => 3
)
port map (
    clk => clk,
    resetn => resetn,
    end_counter => end_count
);

-- Process séquentielle

process(clk,resetn)
begin

if(resetn='1') then
    end_count2<= 0;

    -- current_state <= idle;

elseif(rising_edge(clk)) then
    if(raz='0') then
        if (end_count='1') then
            end_count2<=end_count2 +1;
        elseif(end_count='0') then
            end_count2<=end_count2;
        end if;
    else end_count2<=0;

    --current_state <= next_state;
    end if;
end if;
    --a completer avec votre compteur de cycles

    raz <= '1' when (end_count2 = ((nbre_cycle)-1) and end_count = '1')
    else '0';
    end_counter2 <= raz;

end process;
end behavioral;

```

4. Tester votre architecture avec un testbench.

```

library ieee;
use ieee.std_logic_1164.all;

entity tb_tp_fsm is
end tb_tp_fsm;

architecture behavioral of tb_tp_fsm is

    signal resetn      : std_logic := '0';
    signal clk          : std_logic := '0';
    --a completer
    signal end_count2 : positive :=0;
    signal end_counter2 : std_logic:= '0';
    -- Les constantes suivantes permette de definir la frequence de l'horloge
    constant hp : time := 5 ns;      --demi periode de 5ns
    constant period : time := 2*hp;  --periode de 10ns, soit une frequence de 100Hz
    constant nbre_cycle : positive := 6 ;
    constant max_count : integer :=4;
    constant nb_bit : integer := 3;

    component tp_fsm
    port (
        clk          : in std_logic;
        resetn       : in std_logic;
        --a completer
        end_counter2  : out std_logic
    );
    end component;

begin
    dut: tp_fsm
    port map (
        clk => clk,
        resetn => resetn,
        --a completer
        end_counter2 => end_counter2
    );

    --Simulation du signal d'horloge en continue
    process

        --Simulation du signal d'horloge en continue
    process
    begin

        wait for hp;
        clk <= not clk;

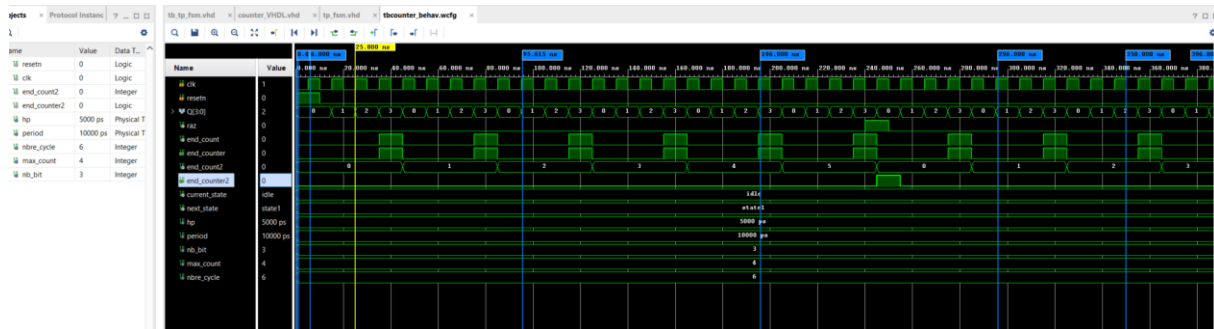
    end process;

    process
    begin

        resetn <= '1';
        wait for period*1;
        resetn <= '0';
        assert end_counter2='0'
        report "end_counter2 : test failed";
        wait for period*2;
        --a completer
        wait for period*nbre_cycle*max_count;
        assert end_counter2='1'
        report "end_counter2 : test failed";
        wait;
    end process;

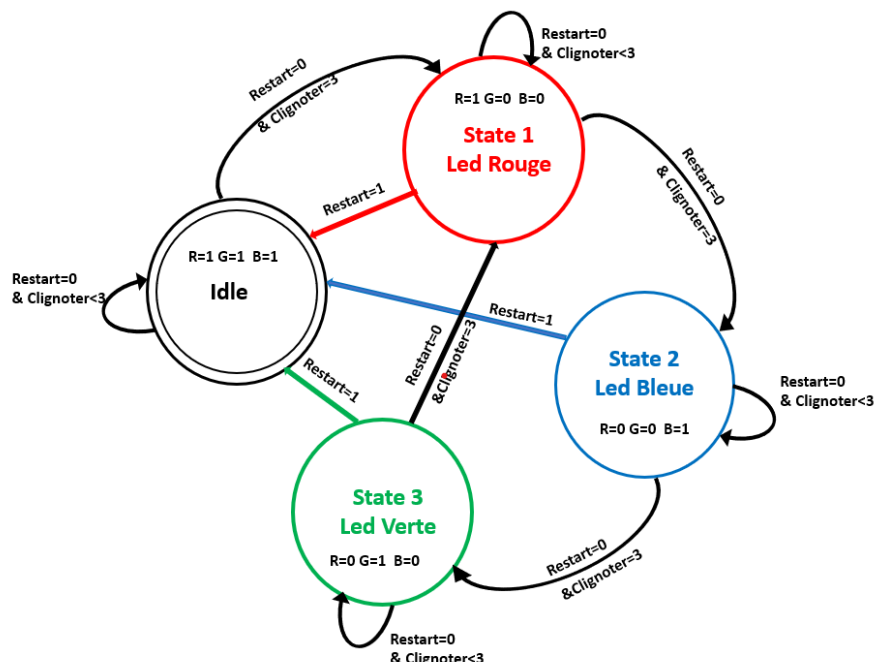
end behavioral;

```



On remarque que le comportement du compteur permet de déterminer le nombre de cycles allumé/éteint qui ont été effectués par la LED. Le compteur est ensuite remis à 0, pour maintenir sa valeur actuelle ou s'incrémenter. On compte 6 fronts montant d'end\_counter et end\_count2. End\_couter2 est alors à 1, puis remise à 0 pour un nouveau cycle.

5. Créez en RTL une machine à états (FSM) permettant de faire clignoter une LED RGB en rouge puis bleu et enfin en vert avant de recommencer le cycle (rouge, bleu, vert, ...). Dans chaque état la LED devra clignoter 3 fois. De plus, si le bouton restart est appuyé, on retourne dans l'état initial quel que soit l'état dans lequel on se situe. L'état initial est l'état dans lequel on se situe au démarrage, on passe à l'état rouge après 3 clignotements de la LED en blanc (rouge, vert et bleu actifs en même temps).



6. Listez les signaux d'entrée, de sortie et les signaux internes de votre architecture.

Les signaux d'entrées

-Clk : l'horloge

-Resetn : le reset

-Restart : la remise à l'état initial de notre FSM

Les signaux de sorties

-end\_counter2 : Sortie du compteur

-Led\_out : Leds RGB

Les signaux internes

current\_state : état dans lequel se trouve la led actuellement

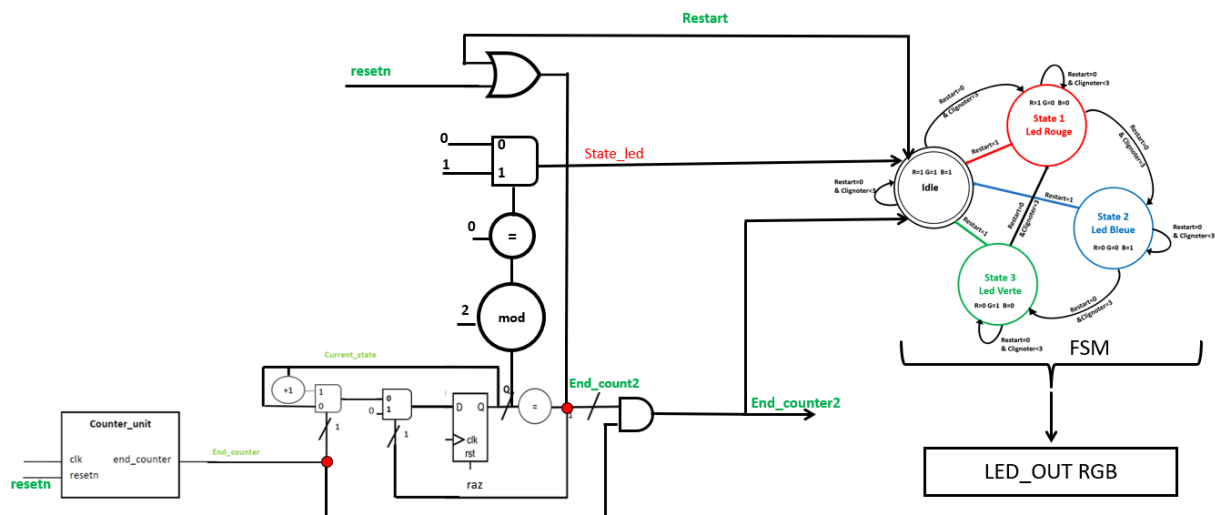
-next\_state : état dans lequel va se trouver la led au prochain coup d'horloge

-s\_led\_out : signal interne pour led\_out RGB

-state\_led : état de la led.

On garde tous les signaux internes déclarer pour le compteur unit et la remise à zéro.

7. Ajoutez à votre code VHDL les éléments que vous venez de créer.



Pour les sortie de leds en RGB, un mux est utilisé dans la fsm(current\_state ) avec le state\_led pour chaque couleur

Le schéma de l'ensemble, les signaux clock et resten sont reliés entre eux. Les points rouges sont les points de connexions du même signal. Sur ce schéma RTL, nous rajoutons à la FSM qui gère les états (state1, stade2, state3, et idle) de la led, le module du compteur, des opérateurs logiques et le restart pour un bon fonctionnement. Pour gérer le clignotement de la led(éteint/allumé), un opérateur modulo et un diviseur 2 est utilisé. En sortie d'end\_count2, nous avons 0 pour le pair ou 1 pour les impairs. Les leds sont allumées sur les impairs (state\_led). La FSM vient ensuite gérer les états : initial, led rouge, led bleue et led verte. Le compteur changerait d'état après 6 cycles au front montant.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4  use ieee.std_logic_arith;
5
6  entity tp_fsm is
7  generic (
8  -- max_count : integer :=2000000000;
9  -- nb_bit : integer := 28;
10     max_count : integer :=4;
11     nb_bit : integer := 3;
12     nbre_cycle : positive :=6 --cycle de clignotement allumé et éteint
13 );
14 port (
15     clk          : in std_logic;
16     resetn       : in std_logic;
17     --a completer
18     restart      : in std_logic;
19     end_counter2 : out std_logic;
20     led_out_R    : out std_logic;
21     led_out_B    : out std_logic;
22     led_out_V    : out std_logic
23 );
24 end tp_fsm;
25
26
27 architecture behavioral of tp_fsm is
28
29     type state is (idle, state1, state2, state3); --a modifier avec vos etats
30
31     signal current_state : state; --etat dans lequel on se trouve actuellement
32     signal next_state : state; --etat dans lequel on passera au prochain coup d'horloge
33     signal s_led_out_B : std_logic; -- signal de sortie de LED RGB
34     signal s_led_out_V : std_logic;
35     signal s_led_out_R : std_logic;
36     signal raz : std_logic; -- signal de remise à zéro
37     signal end_count : std_logic; -- signal de sortie de counter_unit
38     signal end_count2 : integer range 0 to 5:=0; -- signal de sortie de counter2
39     signal state_led : std_logic; -- signal de'état de led
40
41     --Declaration de l'entite a tester counter_unit
42     component counter_unit
43     generic (
44     -- max_count : integer :=2000000000;
45     -- nb_bit : integer := 28
46     max_count : integer :=4;
47     nb_bit : integer := 3
48 );
49     port (
50         clk          : in std_logic;
51         resetn       : in std_logic;
52
53         end_counter  : out std_logic
54 );
55 end component;
56
57 begin
58
59     compteur : counter_unit
60     generic map (
61     -- max_count =>2000000000,
62     -- nb_bit => 28
63     max_count =>4,
64     nb_bit => 3
65 )
66     port map (
67         clk => clk,
68         resetn=>resetn,
69         end_counter => end_count
70 );
71
72
73     -- Process séquentielle du clk, resetn, restart
74
75     process(clk,resetn,restart)
76     begin
77
78     if(resetn='1' or restart='1') then
79         end_count2<= 0;
80         current_state <= idle;
81
82     elsif(rising_edge(clk)) then
83         current_state <= next_state;
84
85         if(raz='0') then

```

```

85         if(raz='0') then
86             if (end_count='1') then
87                 end_count2<=end_count2 +1;
88             elsif(end_count='0') then
89                 end_count2<=end_count2;
90             end if;
91         else
92             end_count2<=0;
93         end if;
94     end if;
95 end process;
96
97 --Partie combinatoire a completer avec votre compteur de cycles
98
99 --remise à zero à la fin du compteur de cycles
100 raz <= '1' when (end_count2 = ((nbre_cycle)-1) and end_count = '1')
101 else '0';
102 end_counter2 <= raz;
103
104 -- Vérification de l'état de end_counter2 pour determine si la led doit etre 'on' or 'off'
105 state_led <= '0' when( end_count2 mod 2 = 0 )else '1';
106
107 -- FSM
108 process(current_state, restart,state_led,end_counter2) --a completer avec les signaux
109 current_state, restart,state_led,end_counter2
110
111 begin
112     --signaux pilotes par la fsm les signaux sont affectés dans les différentes cas
113
114     case current_state is
115         when idle =>
116             s_led_out_R<=state_led;
117             s_led_out_B<=state_led;
118             s_led_out_V<=state_led;
119             if restart='0' then
120                 if(end_counter2='1') then
121                     next_state <= statel;
122                 else
123                     next_state <= idle;           --prochain etat
124                 end if;
125             else
126                 next_state<=idle;
127             end if;
128
129         when statel =>
130             s_led_out_R<=state_led;
131             s_led_out_B<='0';
132             s_led_out_V<='0';
133             if restart='0' then
134                 if(end_counter2='1') then
135                     next_state <= state2;           --prochain etat
136                 else
137                     next_state <= statel;
138                 end if;
139             else
140                 next_state <= idle;
141             end if;
142
143         when state2 =>
144             s_led_out_R<='0';
145             s_led_out_B<=state_led;
146             s_led_out_V<='0';
147             if restart='0' then
148                 if(end_counter2='1') then
149                     next_state <= state3;           --prochain etat
150                 else
151                     next_state <= current_state;
152                 end if;
153             else
154                 next_state <= idle;
155             end if;
156
157         when state3 =>
158             s_led_out_R<='0';
159             s_led_out_B<='0';
160             s_led_out_V<=state_led;
161             if restart='0' then
162                 if(end_counter2='1') then
163                     next_state <= state4;           --prochain etat
164                 else
165                     next_state <= current_state;
166                 end if;
167             else
168                 next_state <= idle;
169             end if;
170
171         when state4 =>
172             s_led_out_R<='0';
173             s_led_out_B<='0';
174             s_led_out_V<=state_led;
175             if restart='0' then
176                 if(end_counter2='1') then
177                     next_state <= state5;           --prochain etat
178                 else
179                     next_state <= current_state;
180                 end if;
181             else
182                 next_state <= idle;
183             end if;
184
185         when state5 =>
186             s_led_out_R<='0';
187             s_led_out_B<='0';
188             s_led_out_V<=state_led;
189             if restart='0' then
190                 if(end_counter2='1') then
191                     next_state <= state6;           --prochain etat
192                 else
193                     next_state <= current_state;
194                 end if;
195             else
196                 next_state <= idle;
197             end if;
198
199         when state6 =>
200             s_led_out_R<='0';
201             s_led_out_B<='0';
202             s_led_out_V<=state_led;
203             if restart='0' then
204                 if(end_counter2='1') then
205                     next_state <= state7;           --prochain etat
206                 else
207                     next_state <= current_state;
208                 end if;
209             else
210                 next_state <= idle;
211             end if;
212
213         when state7 =>
214             s_led_out_R<='0';
215             s_led_out_B<='0';
216             s_led_out_V<=state_led;
217             if restart='0' then
218                 if(end_counter2='1') then
219                     next_state <= state8;           --prochain etat
220                 else
221                     next_state <= current_state;
222                 end if;
223             else
224                 next_state <= idle;
225             end if;
226
227         when state8 =>
228             s_led_out_R<='0';
229             s_led_out_B<='0';
230             s_led_out_V<=state_led;
231             if restart='0' then
232                 if(end_counter2='1') then
233                     next_state <= state9;           --prochain etat
234                 else
235                     next_state <= current_state;
236                 end if;
237             else
238                 next_state <= idle;
239             end if;
240
241         when state9 =>
242             s_led_out_R<='0';
243             s_led_out_B<='0';
244             s_led_out_V<=state_led;
245             if restart='0' then
246                 if(end_counter2='1') then
247                     next_state <= state10;          --prochain etat
248                 else
249                     next_state <= current_state;
250                 end if;
251             else
252                 next_state <= idle;
253             end if;
254
255         when state10 =>
256             s_led_out_R<='0';
257             s_led_out_B<='0';
258             s_led_out_V<=state_led;
259             if restart='0' then
260                 if(end_counter2='1') then
261                     next_state <= state11;          --prochain etat
262                 else
263                     next_state <= current_state;
264                 end if;
265             else
266                 next_state <= idle;
267             end if;
268
269         when state11 =>
270             s_led_out_R<='0';
271             s_led_out_B<='0';
272             s_led_out_V<=state_led;
273             if restart='0' then
274                 if(end_counter2='1') then
275                     next_state <= state12;          --prochain etat
276                 else
277                     next_state <= current_state;
278                 end if;
279             else
280                 next_state <= idle;
281             end if;
282
283         when state12 =>
284             s_led_out_R<='0';
285             s_led_out_B<='0';
286             s_led_out_V<=state_led;
287             if restart='0' then
288                 if(end_counter2='1') then
289                     next_state <= state13;          --prochain etat
290                 else
291                     next_state <= current_state;
292                 end if;
293             else
294                 next_state <= idle;
295             end if;
296
297         when state13 =>
298             s_led_out_R<='0';
299             s_led_out_B<='0';
300             s_led_out_V<=state_led;
301             if restart='0' then
302                 if(end_counter2='1') then
303                     next_state <= state14;          --prochain etat
304                 else
305                     next_state <= current_state;
306                 end if;
307             else
308                 next_state <= idle;
309             end if;
310
311         when state14 =>
312             s_led_out_R<='0';
313             s_led_out_B<='0';
314             s_led_out_V<=state_led;
315             if restart='0' then
316                 if(end_counter2='1') then
317                     next_state <= state15;          --prochain etat
318                 else
319                     next_state <= current_state;
320                 end if;
321             else
322                 next_state <= idle;
323             end if;
324
325         when state15 =>
326             s_led_out_R<='0';
327             s_led_out_B<='0';
328             s_led_out_V<=state_led;
329             if restart='0' then
330                 if(end_counter2='1') then
331                     next_state <= state16;          --prochain etat
332                 else
333                     next_state <= current_state;
334                 end if;
335             else
336                 next_state <= idle;
337             end if;
338
339         when state16 =>
340             s_led_out_R<='0';
341             s_led_out_B<='0';
342             s_led_out_V<=state_led;
343             if restart='0' then
344                 if(end_counter2='1') then
345                     next_state <= state17;          --prochain etat
346                 else
347                     next_state <= current_state;
348                 end if;
349             else
350                 next_state <= idle;
351             end if;
352
353         when state17 =>
354             s_led_out_R<='0';
355             s_led_out_B<='0';
356             s_led_out_V<=state_led;
357             if restart='0' then
358                 if(end_counter2='1') then
359                     next_state <= state18;          --prochain etat
360                 else
361                     next_state <= current_state;
362                 end if;
363             else
364                 next_state <= idle;
365             end if;
366
367         when state18 =>
368             s_led_out_R<='0';
369             s_led_out_B<='0';
370             s_led_out_V<=state_led;
371             if restart='0' then
372                 if(end_counter2='1') then
373                     next_state <= state19;          --prochain etat
374                 else
375                     next_state <= current_state;
376                 end if;
377             else
378                 next_state <= idle;
379             end if;
380
381         when state19 =>
382             s_led_out_R<='0';
383             s_led_out_B<='0';
384             s_led_out_V<=state_led;
385             if restart='0' then
386                 if(end_counter2='1') then
387                     next_state <= state20;          --prochain etat
388                 else
389                     next_state <= current_state;
390                 end if;
391             else
392                 next_state <= idle;
393             end if;
394
395         when state20 =>
396             s_led_out_R<='0';
397             s_led_out_B<='0';
398             s_led_out_V<=state_led;
399             if restart='0' then
400                 if(end_counter2='1') then
401                     next_state <= state21;          --prochain etat
402                 else
403                     next_state <= current_state;
404                 end if;
405             else
406                 next_state <= idle;
407             end if;
408
409         when state21 =>
410             s_led_out_R<='0';
411             s_led_out_B<='0';
412             s_led_out_V<=state_led;
413             if restart='0' then
414                 if(end_counter2='1') then
415                     next_state <= state22;          --prochain etat
416                 else
417                     next_state <= current_state;
418                 end if;
419             else
420                 next_state <= idle;
421             end if;
422
423         when state22 =>
424             s_led_out_R<='0';
425             s_led_out_B<='0';
426             s_led_out_V<=state_led;
427             if restart='0' then
428                 if(end_counter2='1') then
429                     next_state <= state23;          --prochain etat
430                 else
431                     next_state <= current_state;
432                 end if;
433             else
434                 next_state <= idle;
435             end if;
436
437         when state23 =>
438             s_led_out_R<='0';
439             s_led_out_B<='0';
440             s_led_out_V<=state_led;
441             if restart='0' then
442                 if(end_counter2='1') then
443                     next_state <= state24;          --prochain etat
444                 else
445                     next_state <= current_state;
446                 end if;
447             else
448                 next_state <= idle;
449             end if;
450
451         when state24 =>
452             s_led_out_R<='0';
453             s_led_out_B<='0';
454             s_led_out_V<=state_led;
455             if restart='0' then
456                 if(end_counter2='1') then
457                     next_state <= state25;          --prochain etat
458                 else
459                     next_state <= current_state;
460                 end if;
461             else
462                 next_state <= idle;
463             end if;
464
465         when state25 =>
466             s_led_out_R<='0';
467             s_led_out_B<='0';
468             s_led_out_V<=state_led;
469             if restart='0' then
470                 if(end_counter2='1') then
471                     next_state <= state26;          --prochain etat
472                 else
473                     next_state <= current_state;
474                 end if;
475             else
476                 next_state <= idle;
477             end if;
478
479         when state26 =>
480             s_led_out_R<='0';
481             s_led_out_B<='0';
482             s_led_out_V<=state_led;
483             if restart='0' then
484                 if(end_counter2='1') then
485                     next_state <= state27;          --prochain etat
486                 else
487                     next_state <= current_state;
488                 end if;
489             else
490                 next_state <= idle;
491             end if;
492
493         when state27 =>
494             s_led_out_R<='0';
495             s_led_out_B<='0';
496             s_led_out_V<=state_led;
497             if restart='0' then
498                 if(end_counter2='1') then
499                     next_state <= state28;          --prochain etat
500                 else
501                     next_state <= current_state;
502                 end if;
503             else
504                 next_state <= idle;
505             end if;
506
507         when state28 =>
508             s_led_out_R<='0';
509             s_led_out_B<='0';
510             s_led_out_V<=state_led;
511             if restart='0' then
512                 if(end_counter2='1') then
513                     next_state <= state29;          --prochain etat
514                 else
515                     next_state <= current_state;
516                 end if;
517             else
518                 next_state <= idle;
519             end if;
520
521         when state29 =>
522             s_led_out_R<='0';
523             s_led_out_B<='0';
524             s_led_out_V<=state_led;
525             if restart='0' then
526                 if(end_counter2='1') then
527                     next_state <= state30;          --prochain etat
528                 else
529                     next_state <= current_state;
530                 end if;
531             else
532                 next_state <= idle;
533             end if;
534
535         when state30 =>
536             s_led_out_R<='0';
537             s_led_out_B<='0';
538             s_led_out_V<=state_led;
539             if restart='0' then
540                 if(end_counter2='1') then
541                     next_state <= state31;          --prochain etat
542                 else
543                     next_state <= current_state;
544                 end if;
545             else
546                 next_state <= idle;
547             end if;
548
549         when state31 =>
550             s_led_out_R<='0';
551             s_led_out_B<='0';
552             s_led_out_V<=state_led;
553             if restart='0' then
554                 if(end_counter2='1') then
555                     next_state <= state32;          --prochain etat
556                 else
557                     next_state <= current_state;
558                 end if;
559             else
560                 next_state <= idle;
561             end if;
562
563         when state32 =>
564             s_led_out_R<='0';
565             s_led_out_B<='0';
566             s_led_out_V<=state_led;
567             if restart='0' then
568                 if(end_counter2='1') then
569                     next_state <= state33;          --prochain etat
570                 else
571                     next_state <= current_state;
572                 end if;
573             else
574                 next_state <= idle;
575             end if;
576
577         when state33 =>
578             s_led_out_R<='0';
579             s_led_out_B<='0';
580             s_led_out_V<=state_led;
581             if restart='0' then
582                 if(end_counter2='1') then
583                     next_state <= state34;          --prochain etat
584                 else
585                     next_state <= current_state;
586                 end if;
587             else
588                 next_state <= idle;
589             end if;
590
591         when state34 =>
592             s_led_out_R<='0';
593             s_led_out_B<='0';
594             s_led_out_V<=state_led;
595             if restart='0' then
596                 if(end_counter2='1') then
597                     next_state <= state35;          --prochain etat
598                 else
599                     next_state <= current_state;
600                 end if;
601             else
602                 next_state <= idle;
603             end if;
604
605         when state35 =>
606             s_led_out_R<='0';
607             s_led_out_B<='0';
608             s_led_out_V<=state_led;
609             if restart='0' then
610                 if(end_counter2='1') then
611                     next_state <= state36;          --prochain etat
612                 else
613                     next_state <= current_state;
614                 end if;
615             else
616                 next_state <= idle;
617             end if;
618
619         when state36 =>
620             s_led_out_R<='0';
621             s_led_out_B<='0';
622             s_led_out_V<=state_led;
623             if restart='0' then
624                 if(end_counter2='1') then
625                     next_state <= state37;          --prochain etat
626                 else
627                     next_state <= current_state;
628                 end if;
629             else
630                 next_state <= idle;
631             end if;
632
633         when state37 =>
634             s_led_out_R<='0';
635             s_led_out_B<='0';
636             s_led_out_V<=state_led;
637             if restart='0' then
638                 if(end_counter2='1') then
639                     next_state <= state38;          --prochain etat
640                 else
641                     next_state <= current_state;
642                 end if;
643             else
644                 next_state <= idle;
645             end if;
646
647         when state38 =>
648             s_led_out_R<='0';
649             s_led_out_B<='0';
650             s_led_out_V<=state_led;
651             if restart='0' then
652                 if(end_counter2='1') then
653                     next_state <= state39;          --prochain etat
654                 else
655                     next_state <= current_state;
656                 end if;
657             else
658                 next_state <= idle;
659             end if;
660
661         when state39 =>
662             s_led_out_R<='0';
663             s_led_out_B<='0';
664             s_led_out_V<=state_led;
665             if restart='0' then
666                 if(end_counter2='1') then
667                     next_state <= state40;          --prochain etat
668                 else
669                     next_state <= current_state;
670                 end if;
671             else
672                 next_state <= idle;
673             end if;
674
675         when state40 =>
676             s_led_out_R<='0';
677             s_led_out_B<='0';
678             s_led_out_V<=state_led;
679             if restart='0' then
680                 if(end_counter2='1') then
681                     next_state <= state41;          --prochain etat
682                 else
683                     next_state <= current_state;
684                 end if;
685             else
686                 next_state <= idle;
687             end if;
688
689         when state41 =>
690             s_led_out_R<='0';
691             s_led_out_B<='0';
692             s_led_out_V<=state_led;
693             if restart='0' then
694                 if(end_counter2='1') then
695                     next_state <= state42;          --prochain etat
696                 else
697                     next_state <= current_state;
698                 end if;
699             else
700                 next_state <= idle;
701             end if;
702
703         when state42 =>
704             s_led_out_R<='0';
705             s_led_out_B<='0';
706             s_led_out_V<=state_led;
707             if restart='0' then
708                 if(end_counter2='1') then
709                     next_state <= state43;          --prochain etat
710                 else
711                     next_state <= current_state;
712                 end if;
713             else
714                 next_state <= idle;
715             end if;
716
717         when state43 =>
718             s_led_out_R<='0';
719             s_led_out_B<='0';
720             s_led_out_V<=state_led;
721             if restart='0' then
722                 if(end_counter2='1') then
723                     next_state <= state44;          --prochain etat
724                 else
725                     next_state <= current_state;
726                 end if;
727             else
728                 next_state <= idle;
729             end if;
730
731         when state44 =>
732             s_led_out_R<='0';
733             s_led_out_B<='0';
734             s_led_out_V<=state_led;
735             if restart='0' then
736                 if(end_counter2='1') then
737                     next_state <= state45;          --prochain etat
738                 else
739                     next_state <= current_state;
740                 end if;
741             else
742                 next_state <= idle;
743             end if;
744
745         when state45 =>
746             s_led_out_R<='0';
747             s_led_out_B<='0';
748             s_led_out_V<=state_led;
749             if restart='0' then
750                 if(end_counter2='1') then
751                     next_state <= state46;          --prochain etat
752                 else
753                     next_state <= current_state;
754                 end if;
755             else
756                 next_state <= idle;
757             end if;
758
759         when state46 =>
760             s_led_out_R<='0';
761             s_led_out_B<='0';
762             s_led_out_V<=state_led;
763             if restart='0' then
764                 if(end_counter2='1') then
765                     next_state <= state47;          --prochain etat
766                 else
767                     next_state <= current_state;
768                 end if;
769             else
770                 next_state <= idle;
771             end if;
772
773         when state47 =>
774             s_led_out_R<='0';
775             s_led_out_B<='0';
776             s_led_out_V<=state_led;
777             if restart='0' then
778                 if(end_counter2='1') then
779                     next_state <= state48;          --prochain etat
780                 else
781                     next_state <= current_state;
782                 end if;
783             else
784                 next_state <= idle;
785             end if;
786
787         when state48 =>
788             s_led_out_R<='0';
789             s_led_out_B<='0';
790             s_led_out_V<=state_led;
791             if restart='0' then
792                 if(end_counter2='1') then
793                     next_state <= state49;          --prochain etat
794                 else
795                     next_state <= current_state;
796                 end if;
797             else
798                 next_state <= idle;
799             end if;
800
801         when state49 =>
802             s_led_out_R<='0';
803             s_led_out_B<='0';
804             s_led_out_V<=state_led;
805             if restart='0' then
806                 if(end_counter2='1') then
807                     next_state <= state50;          --prochain etat
808                 else
809                     next_state <= current_state;
810                 end if;
811             else
812                 next_state <= idle;
813             end if;
814
815         when state50 =>
816             s_led_out_R<='0';
817             s_led_out_B<='0';
818             s_led_out_V<=state_led;
819             if restart='0' then
820                 if(end_counter2='1') then
821                     next_state <= state51;          --prochain etat
822                 else
823                     next_state <= current_state;
824                 end if;
825             else
826                 next_state <= idle;
827             end if;
828
829         when state51 =>
830             s_led_out_R<='0';
831             s_led_out_B<='0';
832             s_led_out_V<=state_led;
833             if restart='0' then
834                 if(end_counter2='1') then
835                     next_state <= state52;          --prochain etat
836                 else
837                     next_state <= current_state;
838                 end if;
839             else
840                 next_state <= idle;
841             end if;
842
843         when state52 =>
844             s_led_out_R<='0';
845             s_led_out_B<='0';
846             s_led_out_V<=state_led;
847             if restart='0' then
848                 if(end_counter2='1') then
849                     next_state <= state53;          --prochain etat
850                 else
851                     next_state <= current_state;
852                 end if;
853             else
854                 next_state <= idle;
855             end if;
856
857         when state53 =>
858             s_led_out_R<='0';
859             s_led_out_B<='0';
860             s_led_out_V<=state_led;
861             if restart='0' then
862                 if(end_counter2='1') then
863                     next_state <= state54;          --prochain etat
864                 else
865                     next_state <= current_state;
866                 end if;
867             else
868                 next_state <= idle;
869             end if;
870
871         when state54 =>
872             s_led_out_R<='0';
873             s_led_out_B<='0';
874             s_led_out_V<=state_led;
875             if restart='0' then
876                 if(end_counter2='1') then
877                     next_state <= state55;          --prochain etat
878                 else
879                     next_state <= current_state;
880                 end if;
881             else
882                 next_state <= idle;
883             end if;
884
885         when state55 =>
886             s_led_out_R<='0';
887             s_led_out_B<='0';
888             s_led_out_V<=state_led;
889             if restart='0' then
890                 if(end_counter2='1') then
891                     next_state <= state56;          --prochain etat
892                 else
893                     next_state <= current_state;
894                 end if;
895             else
896                 next_state <= idle;
897             end if;
898
899         when state56 =>
900             s_led_out_R<='0';
901             s_led_out_B<='0';
902             s_led_out_V<=state_led;
903             if restart='0' then
904                 if(end_counter2='1') then
905                     next_state <= state57;          --prochain etat
906                 else
907                     next_state <= current_state;
908                 end if;
909             else
910                 next_state <= idle;
911             end if;
912
913         when state57 =>
914             s_led_out_R<='0';
915             s_led_out_B<='0';
916             s_led_out_V<=state_led;
917             if restart='0' then
918                 if(end_counter2='1') then
919                     next_state <= state58;          --prochain etat
920                 else
921                     next_state <= current_state;
922                 end if;
923             else
924                 next_state <= idle;
925             end if;
926
927         when state58 =>
928             s_led_out_R<='0';
929             s_led_out_B<='0';
930             s_led_out_V<=state_led;
931             if restart='0' then
932                 if(end_counter2='1') then
933                     next_state <= state59;          --prochain etat
934                 else
935                     next_state <= current_state;
936                 end if;
937             else
938                 next_state <= idle;
939             end if;
940
941         when state59 =>
942             s_led_out_R<='0';
943             s_led_out_B<='0';
944             s_led_out_V<=state_led;
945             if restart='0' then
946                 if(end_counter2='1') then
947                     next_state <= state60;          --prochain etat
948                 else
949                     next_state <= current_state;
950                 end if;
951             else
952                 next_state <= idle;
953             end if;
954
955         when state60 =>
956             s_led_out_R<='0';
957             s_led_out_B<='0';
958             s_led_out_V<=state_led;
959             if restart='0' then
960                 if(end_counter2='1') then
961                     next_state <= state61;          --prochain etat
962                 else
963                     next_state <= current_state;
964                 end if;
965             else
966                 next_state <= idle;
967             end if;
968
969         when state61 =>
970             s_led_out_R<='0';
971             s_led_out_B<='0';
972             s_led_out_V<=state_led;
973             if restart='0' then
974                 if(end_counter2='1') then
975                     next_state <= state62;          --prochain etat
976                 else
977                     next_state <= current_state;
978                 end if;
979             else
980                 next_state <= idle;
981             end if;
982
983         when state62 =>
984             s_led_out_R<='0';
985             s_led_out_B<='0';
986             s_led_out_V<=state_led;
987             if restart='0' then
988                 if(end_counter2='1') then
989                     next_state <= state63;          --prochain etat
990                 else
991                     next_state <= current_state;
992                 end if;
993             else
994                 next_state <= idle;
995             end if;
996
997         when state63 =>
998             s_led_out_R<='0';
999             s_led_out_B<='0';
1000            s_led_out_V<=state_led;
1001            if restart='0' then
1002                if(end_counter2='1') then
1003                    next_state <= state64;          --prochain etat
1004                else
1005                    next_state <= current_state;
1006                end if;
1007            else
1008                next_state <= idle;
1009            end if;
1010
1011         when state64 =>
1012             s_led_out_R<='0';
1013             s_led_out_B<='0';
1014             s_led_out_V<=state_led;
1015             if restart='0' then
1016                 if(end_counter2='1') then
1017                     next_state <= state65;          --prochain etat
1018                 else
1019                     next_state <= current_state;
1020                 end if;
1021             else
1022                 next_state <= idle;
1023             end if;
1024
1025         when state65 =>
1026             s_led_out_R<='0';
1027             s_led_out_B<='0';
1028             s_led_out_V<=state_led;
1029             if restart='0' then
1030                 if(end_counter2='1') then
1031                     next_state <= state66;          --prochain etat
1032                 else
1033                     next_state <= current_state;
1034                 end if;
1035             else
1036                 next_state <= idle;
1037             end if;
1038
1039         when state66 =>
1040             s_led_out_R<='0';
1041             s_led_out_B<='0';
1042             s_led_out_V<=state_led;
1043             if restart='0' then
1044                 if(end_counter2='1') then
1045                     next_state <= state67;          --prochain etat
1046                 else
1047                     next_state <= current_state;
1048                 end if;
1049             else
1050                 next_state <= idle;
1051             end if;
1052
1053         when state67 =>
1054             s_led_out_R<='0';
1055             s_led_out_B<='0';
1056             s_led_out_V<=state_led;
1057             if restart='0' then
1058                 if(end_counter2='1') then
1059                     next_state <= state68;          --prochain etat
1060                 else
1061                     next_state <= current_state;
1062                 end if;
1063             else
1064                 next_state <= idle;
1065             end if;
1066
1067         when state68 =>
1068             s_led_out_R<='0';
1069             s_led_out_B<='0';
1070             s_led_out_V<=state_led;
1071             if restart='0' then
1072                 if(end_counter2='1') then
1073                     next_state <= state69;          --prochain etat
1074                 else
1075                     next_state <= current_state;
1076                 end if;
1077             else
1078                 next_state <= idle;
1079             end if;
1080
1081         when state69 =>
1082             s_led_out_R<='0';
1083             s_led_out_B<='0';
1084             s_led_out_V<=state_led;
1085             if restart='0' then
1086                 if(end_counter2='1') then
1087                     next_state <= state70;          --prochain etat
1088                 else
1089                     next_state <= current_state;
1090                 end if;
1091             else
1092                 next_state <= idle;
1093             end if;
1094
1095         when state70 =>
1096             s_led_out_R<='0';
1097             s_led_out_B<='0';
1098             s_led_out_V<=state_led;
1099             if restart='0' then
1100                 if(end_counter2='1') then
1101                     next_state <= state71;          --prochain etat
1102                 else
1103                     next_state <= current_state;
1104                 end if;
1105             else
1106                 next_state <= idle;
1107             end if;
1108
1109         when state71 =>
1110             s_led_out_R<='0';
1111             s_led_out_B<='0';
1112             s_led_out_V<=state_led;
1113             if restart='0' then
1114                 if(end_counter2='1') then
1115                     next_state <= state72;          --prochain etat
1116                 else
1117                     next_state <= current_state;
1118                 end if;
1119             else
1120                 next_state <= idle;
1121             end if;
1122
1123         when state72 =>
1124             s_led_out_R<='0';
1125             s_led_out_B<='0';
1126             s_led_out_V<=state_led;
1127             if restart='0' then
1128                 if(end_counter2='1') then
1129                     next_state <= state73;          --prochain etat
1130                 else
1131                     next_state <= current_state;
1132                 end if;
1133             else
1134                 next_state <= idle;
1135             end if;
1136
1137         when state73 =>
1138             s_led_out_R<='0';
1139             s_led_out_B<='0';
1140             s_led_out_V<=state_led;
1141             if restart='0' then
1142                 if(end_counter2='1') then
1143                     next_state <= state74;          --prochain etat
1144                 else
1145                     next_state <= current_state;
1146                 end if;
1147             else
1148                 next_state <= idle;
1149             end if;
1150
1151         when state74 =>
1152             s_led_out_R<='0';
```



```

171
172     when state3 =>
173         s_led_out_R<='0';
174         s_led_out_B<='0';
175         s_led_out_V<=state_led;
176         if restart='0' then
177
178             if(end_counter2='1') then
179                 next_state <= state1;           --prochain etat
180
181             else
182
183                 next_state <= current_state;
184                 end if;
185             else
186                 next_state <= idle;
187                 end if;
188
189             --signaux pilotes par la fsm
190             when others =>
191
192                 next_state <= idle;
193
194             end case;
195
196     end process;
197
198     --Partie combinatoire a completer avec les leds affectations de signaux
199     led_out_R<=s_led_out_R;
200     led_out_B<=s_led_out_B;
201     led_out_V<=s_led_out_V;
202
203 end behavioral;

```

8. Ecrivez un testbench pour tester votre architecture. Vérifiez à la simulation que vous obtenez le résultat attendu.

```

207
208 library ieee;
209 use ieee.std_logic_1164.all;
210 use ieee.std_logic_unsigned.all;
211 use ieee.std_logic_arith;
212
213
214 entity tb_tp_fsm is
215     end tb_tp_fsm;
216
217 architecture behavioral of tb_tp_fsm is
218
219     --signaux de clk, resetn et restart
220     signal resetn      : std_logic := '0';
221     signal clk         : std_logic := '0';
222     signal restart     : std_logic := '0';
223
224     --Signaux de sorties
225     signal end_count2 : integer range 0 to 5:=0;
226     signal end_counter2 : std_logic:= '0';
227     signal s_led_out_B : std_logic;
228     signal s_led_out_V : std_logic;
229     signal s_led_out_R : std_logic;
230
231     -- Les constantes suivantes permet de definir la frequence de l'horloge
232     constant hp : time := 5 ns;           --demi periode de 5ns
233     constant period : time := 2*hp;      --periode de 10ns, soit une frequence de 100Hz
234     constant nbre_cycle : positive := 6 ;
235     constant max_count : integer :=200000000;
236     constant nb_bit : integer := 28;
237     constant max_count : integer :=4;
238     constant nb_bit : integer := 3;
239
240     --Déclaration du composant fsm
241     component tp_fsm
242     port (
243         clk          : in std_logic;
244         resetn       : in std_logic;
245         restart      : in std_logic;
246         --a completer
247         end_counter2 : out std_logic;
248         led_out_V:    out std_logic;
249         led_out_R:    out std_logic;
250         led_out_B:    out std_logic
251     );
252 end component;

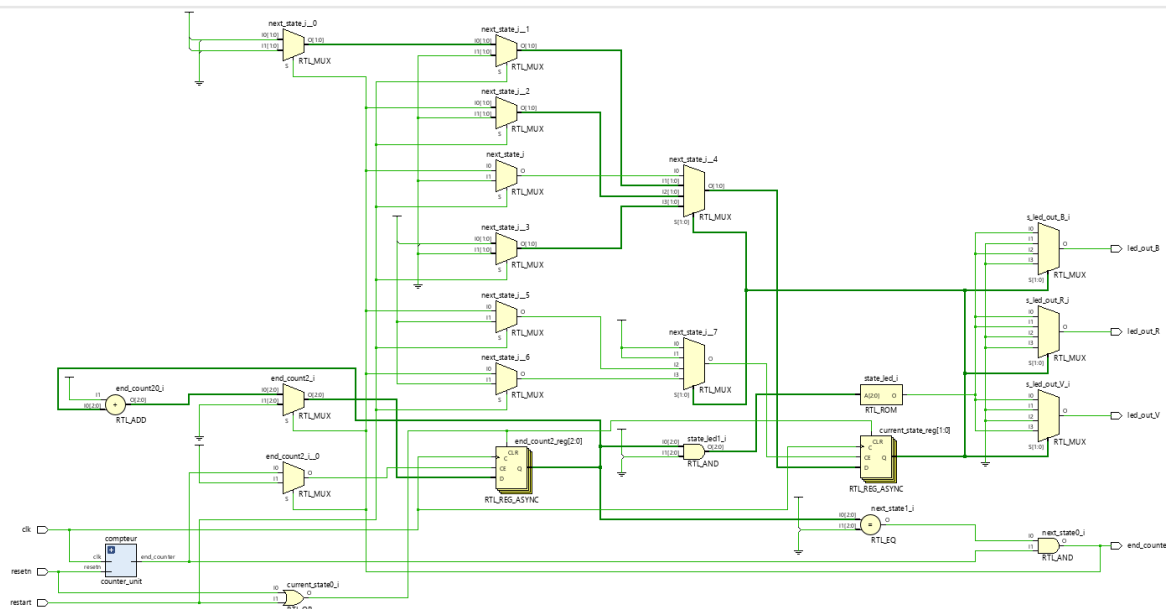
```

```

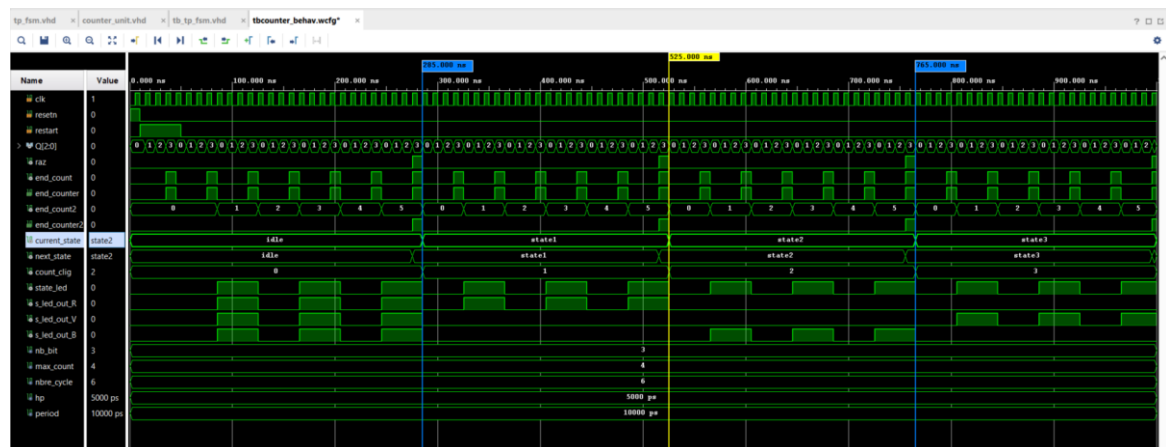
254     begin
255     dut: tp_fsm
256     port map (
257         clk => clk,
258         resetn => resetn,
259         restart => restart,
260         --a completer
261         end_counter2 => end_counter2,
262         led_out_B => s_led_out_B ,
263         led_out_V => s_led_out_V ,
264         led_out_R => s_led_out_R
265     );
266
267     --Simulation du signal d'horloge en continue
268     process
269     begin
270
271         wait for hp;
272         clk <= not clk;
273
274     end process;
275
276     --Simulation du signal resetn en continue avec affichage de end_counter2 qui fonctionne après
    le nombre de cycle
277     process
278     begin
279
280         resetn <= '1';
281
282         --wait for hp;
283         wait for period*1;
284         resetn <= '0';
285         assert end_counter2='0'
286             report "end_counter2 : test failed";
287         wait for period*2;
288         --a completer
289         wait for period*nbre_cycle*max_count;
290         assert end_counter2='1'
291             report "end_counter2 : test failed";
292         wait;
293     end process;
294     --simulation de leds et clignotement
295     process
296     begin
297         wait for period*1;
298         -- Restart
299         restart <= '1';
300         wait for 40 ns;
301         restart <= '0';
302         wait for period*nbre_cycle*max_count;
303
304         -- Verifions l'état initial idle (LED blanche)
305         assert (s_led_out_R = '1' and s_led_out_V = '1' and s_led_out_B = '1')
306             report "Initial state mismatch" severity error;
307
308         -- Wait for 3 clignotements en blanc (INITIAL state : idle)
309         wait for period*nbre_cycle*max_count;
310
311         -- Verifions RED state1
312         assert (s_led_out_R = '1' and s_led_out_V = '0' and s_led_out_B = '0')
313             report "RED state mismatch" severity error;
314
315         -- Wait for 3 clignotements en rouge (RED state)
316         wait for period*nbre_cycle*max_count;
317
318         -- Verifions BLUE state2
319         assert (s_led_out_R = '0' and s_led_out_V = '0' and s_led_out_B = '1')
320             report "BLUE state mismatch" severity error;
321
322         -- Wait for 3 clignotements en bleu (BLUE state)
323         wait for period*nbre_cycle*max_count;
324
325         -- Verifions GREEN state3
326         assert (s_led_out_R = '0' and s_led_out_V = '1' and s_led_out_B = '0')
327             report "GREEN state mismatch" severity error;
328
329         -- Wait for 3 clignotements en vert (GREEN state)
330         wait for period*nbre_cycle*max_count;
331
332         --Pour finir la simulation
333         wait;
334     end process;
335 end behavioral;

```

## Le RTL de l'ensemble



Les résultats de simulations sur le chronogramme montrent que cela fonctionne bien.



- Exécutez la synthèse et relevez les ressources utilisées (y compris la FSM). Sur la schématique, identifiez où se situe votre compteur de cycle.

Nous retrouvons dans la synthèse les états de nos leds. Ils sont stockés dans un registre de 2 bits.

INFO: [Synth 8-802] inferred FSM for state register 'current\_state\_reg' in module 'tp\_fsm'

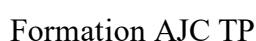
State	New Encoding	Previous Encoding
idle	00	00
state1	01	01
state2	10	10
state3	11	11

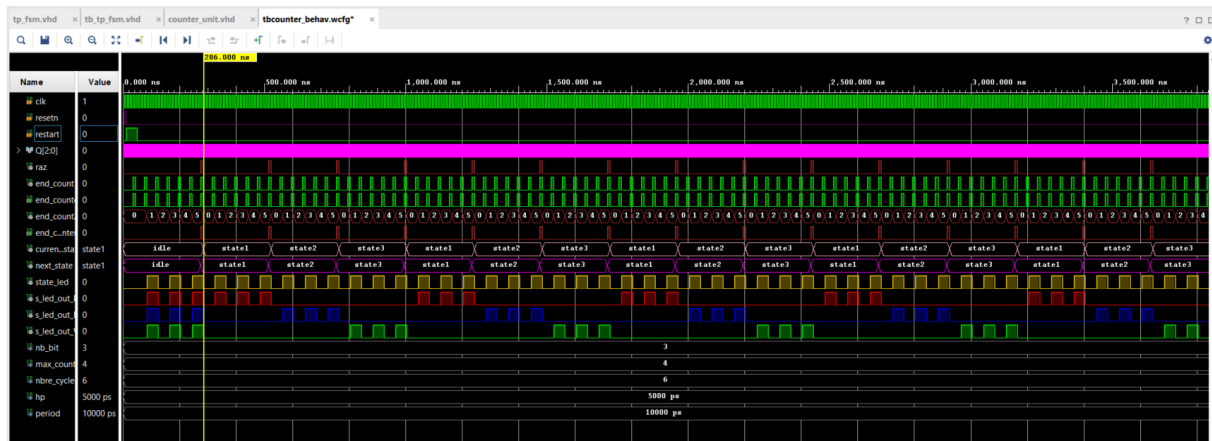
INFO: [Synth 8-3354] encoded FSM with state register 'current\_state\_reg' using encoding 'sequential' in module 'tp\_fsm'

La description du RTL dans la synthèse.

Lorsque le générique est de 200000000, on adonc 33 registres (deux pour la machine d'état, 3 pour le counter2 et 28 pour le counter\_unit).

Dans la synthèse : voici le compteur de cycle. On retrouve autour de lui, les registres qui gère le clignotement et la FSM.





10. Modifiez le fichier de contraintes pour connecter vos entrées / sorties du système avec les broches de la carte. Réglez l'horloge pour que sa fréquence soit à 100MHz.

```
1  ## This file is a general .xdc for the Core S7-07S Rev. B
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project
5
6  # PL System Clock
7  set_property -dict { PACKAGE_PIN H16 IOSTANDARD LVCMOS33 } [get_ports clk]
8  create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]
9
10 # RGB LEDs
11 set_property -dict { PACKAGE_PIN L15 IOSTANDARD LVCMOS33 } [get_ports {led_out_R}]
12 set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports {led_out_B}]
13 set_property -dict { PACKAGE_PIN N15 IOSTANDARD LVCMOS33 } [get_ports {led_out_V}]
14 #set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports {led1_b }]; #IO_0_35 Sch=led1_b
15 set_property -dict { PACKAGE_PIN L14 IOSTANDARD LVCMOS33 } [get_ports { end_counter2 }]; #IO_L22P_T3_AD7P_35 Sch=led1_g
16 #set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { led1_r }]; #IO_L23N_T3_35 Sch=led1_r
17
18 # Buttons
19 set_property -dict { PACKAGE_PIN D20 IOSTANDARD LVCMOS33 } [get_ports { resetn }]; #IO_L4N_T0_35 Sch=btn[0]
20 set_property -dict { PACKAGE_PIN D19 IOSTANDARD LVCMOS33 } [get_ports { restart}]
21
```

La fréquence est réglée à 100MHZ. La schématique est représentée ci-dessous.

11. Lancez l'implémentation puis étudiez le rapport de timing (vérifiez les violations de set up et de hold et identifiez le chemin critique).

Le Clock

Clock Summary			
Clock	Waveform(ns)	Period(ns)	Frequency(MHz)
dbg_hub/inst/BSCANID_u_xsdm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK	{0.000 16.500}	33.000	30.303
sys_clk_pin	{0.000 5.000}	10.000	100.000

On vérifie bien que la période est à 10ns et la fréquence est de 100MHz

Les valeurs dans le THS et TNS sont à 0, il n'y a pas de violation du set up et du hold. Pas de métastabilité.

Design Timing Summary											
WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WFS(ns)	TFWS(ns)	TFWS Failing Endpoints	TFWS Total Endpoints
4.026	0.000	0	3529	0.023	0.000	0	3513	3.750	0.000	0	2200

All user specified timing constraints are met.

Le chemin critique est :

Max Delay Paths

Formation AJC TP

*Slack (MET) :* 25.927ns (*required time - arrival time*)

Source: `dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[0]/C`  
*(rising edge-triggered cell FDRE clocked by*  
`dbg_hub/inst/BSCANID.u_xsdbm_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK`  
*{rise@0.000ns fall@16.500ns period=33.000ns})*

*Destination:*  
*dbg\_hub/inst/BSCANID.u\_xsdbm\_id/SWITCH\_N\_EXT\_BSCAN.bscan\_switch/portno\_temp\_reg[3]/D*  
*(rising edge-triggered cell FDRE clocked by*  
*dbg\_hub/inst/BSCANID.u\_xsdbm\_id/SWITCH\_N\_EXT\_BSCAN.bscan\_inst/SERIES7\_BSCAN.bscan\_inst/TCK*  
*{rise@0.000ns fall@16.500ns period=33.000ns})*

### Max Delay Paths

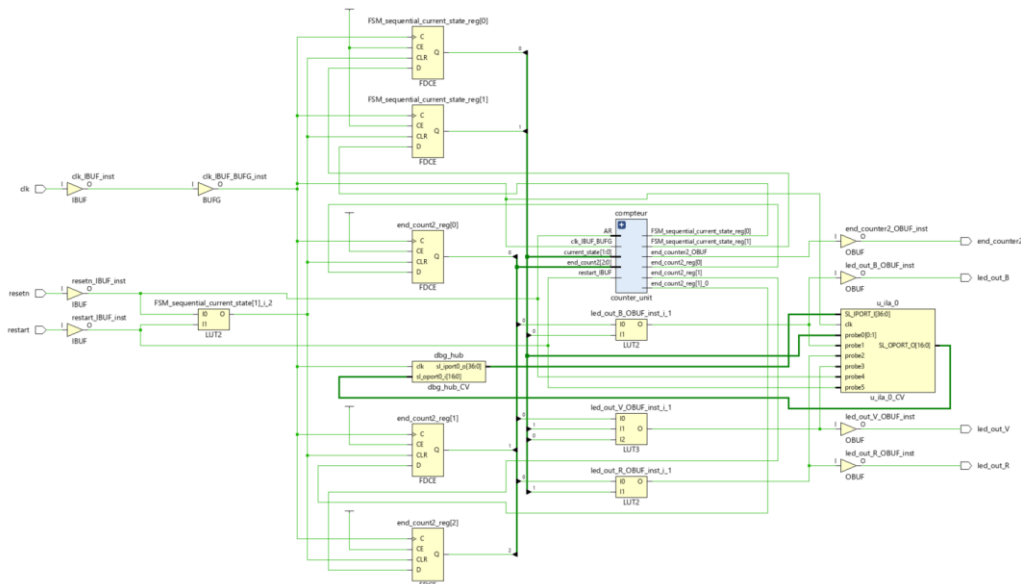
```
Slack (MEF) : 25.927ns (required time - arrival time)
Source: dbg_hub/inst/BSCANID.U_xdmem_id/SWITCH_N_EXT_BSCAN.bscan_switch/state_reg[0]/C
(rising edge triggered cell FDRE clocked by dbg_hub/inst/BSCANID.U_xdmem_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK (rise=0.000ns fall@16.500ns period=33.000ns))
Destination: dbg_hub/inst/BSCANID.U_xdmem_id/SWITCH_N_EXT_BSCAN.bscan_switch/portno_temp_reg[3]/D
(rising edge-triggered cell FDRE clocked by dbg_hub/inst/BSCANID.U_xdmem_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK (rise=80.000ns fall@16.500ns period=33.000ns))
Path Group: dbg_hub/inst/BSCANID.U_xdmem_id/SWITCH_N_EXT_BSCAN.bscan_inst/SERIES7_BSCAN.bscan_inst/TCK
Dark Power: Ratio (Max at Slow Process Corner)
```

12. Générez le bitstream pour vérifier le système sur carte.

Les signaux leds out, et current state sont observés dans le trigger. Hw ila 1

Pour rappel, la principale fonction d'une ILA est de surveiller et d'analyser les signaux numériques à l'intérieur d'un circuit intégré ou d'un FPGA (Field-Programmable Gate Array). Elle permet de détecter les erreurs, de valider le fonctionnement des circuits et de comprendre le comportement des signaux lors de l'exécution d'un programme ou d'une séquence d'opérations.

Nous démarrons l'enregistrement de l'ILA sur le front montant et descendant du port de sortie du `current_state`. On procède à un changement de valeur sur le signal de déclenchement entraînera l'ILA pour commencer à enregistrer les signaux sondés (Led\_out RGB). Ceci est fait dans le déclencheur (trigger setup).



**ILA Core Properties**

hw\_ila\_1

Name: hw\_ila\_1  
Cell: u\_ila\_0  
Device: xc7z010\_1  
HW core: core\_2  
Capture sample count: 512 of 1024  
Core status: Waiting For Trigger

**Settings - hw\_ila\_1**
Trigger mode: BASIC\_ONLY

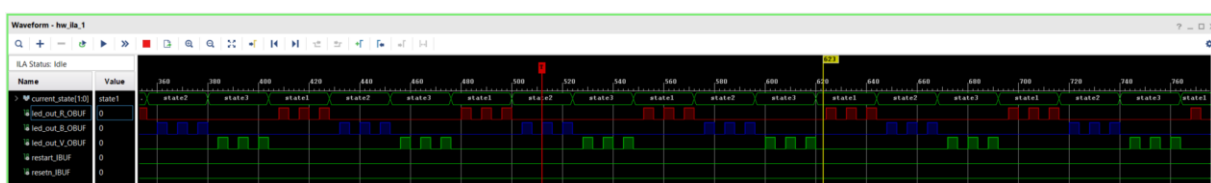
**Capture Mode Settings**
Capture mode: ALWAYS  
Number of windows: 1 [1 - 1024]  
Window data depth: 1024 [1 - 1024]  
Trigger position in window: 512 [0 - 1023]

**General Settings**
Refresh rate: 10000 ms

**Trigger Setup - hw\_ila\_1**

Name	Operator	Radix	Value	Port	Comparator Usage
led_out_B_OBUF	==	[B]	B	probe1[0]	1 of 1
current_state[1:0]	==	[H]	X	probe0[1:0]	
led_out_R_OBUF	==	[B]	B	probe2[0]	1 of 1
led_out_V_OBUF	==	[B]	B	probe3[0]	1 of 1

Nous voyons la ligne verticale rouge (marqueur) sur le front montant de notre signal de déclenchement (port trigger de led), et il est en position 512. Nous pouvons également vérifier que le signal compte se comporte correctement et change de couleur suivant les changements d'état.



Lorsque que le bouton restart est appuyé, on retourne dans l'état initial quel que soit l'état dans lequel on se situe.

