

## Clignotement cyclique de LEDs et suivant un motif

### ■ ■ ■ Clignotement cycliquement de LEDs individuellement

#### 1 – Utilisation d'un compteur pour faire clignoter une LED :

Le contenu du fichier «blink.v» :

```

`ifndef TAILLE_COMPTEUR
`define TAILLE_COMPTEUR 27
`endif

module blink(input clock, input reset, output led);

reg [`TAILLE_COMPTEUR-1:0] counter = `TAILLE_COMPTEUR'h0;

always @(posedge clock or posedge reset)
    if (reset)
        counter <= 0;
    else
        counter <= counter + 1'b1;

assign led = counter[`TAILLE_COMPTEUR-1];

endmodule

```

La version synthétisable dans le fichier «synthese.v» :

```

`include "blink.v"

module top_level(input clock, input greset, output led1);

wire reset;
assign reset = ~greset;

blink b(.clock(clock), .reset(reset), .led(led1));

endmodule

```

La version simulable dans le fichier «tb.v» :

```

`define TAILLE_COMPTEUR 2
`include "blink.v"

`timescale 1ns / 1ps

module tb ();
reg clock;
reg reset;
wire led;

blink dut(.clock(clock), .reset(reset), .led(led));

initial begin
    clock = 1;
    E forever #5 clock <= ~clock;
end

initial begin
    reset = 0;
    #5 reset = 1;
    #5 reset = 0;
end

initial
begin
    $monitor("time=%3d, sortie=%b", $time, led);
    $dumpfile("blink.vcd");
    $dumpvars(0, tb);
    #60 $finish ;
end
endmodule

```

Le contenu du fichier « Makefile » :

```
VERILOG_SRC=synthese.v
PROJECT=$(basename $(VERILOG_SRC))
ARCH=hx8k
CLOCK=90
PACKAGE=tq144:4k
PCF=blackice-ii.pcf
PORT=/dev/ttyACM0

$(PROJECT).json: $(VERILOG_SRC)
    yosys -ql $(PROJECT).log -p 'synth_ice40 -top top_level -json $@' $^

$(PROJECT).asc: $(PCF) $(PROJECT).json
    nextpnr-ice40 --freq $(CLOCK) --$(ARCH) --package $(PACKAGE) --asc $@ --pcf
$(PCF) --json $(PROJECT).json

$(PROJECT).bin: $(PROJECT).asc
    icetime -d $(ARCH) -c $(CLOCK) -mtr $(PROJECT).rpt $(PROJECT).asc
    icepack $^ $@

prog: $(PROJECT).bin
    stty 115200 -F $(PORT) raw; cat $(PROJECT).bin > $(PORT)

show: $(PROJECT).json
    nextpnr-ice40 --gui --freq $(CLOCK) --$(ARCH) --package $(PACKAGE) --pcf $(PCF)
--json $(PROJECT).json
    # nextpnr-ice40 --routed-svg toto.svg --freq $(CLOCK) --$(ARCH) --package $(PA
CKAGE) --pcf $(PCF) --json $(PROJECT).json

all: $(PROJECT).bin

clean:
    rm -f $(PROJECT).json $(PROJECT).asc $(PROJECT).bin
```

Vous pouvez récupérer ces fichiers par git :

```
❏ — xterm —
$ git clone https://git.p-fb.net/PeFClic/fpga_blink.git
```

### Questions :

- Vous regarderez le résultat de la simulation et le résultat du déploiement sur le FPGA.
- À quoi sert le `define TAILLE_COMPTEUR` ?  
Quelle est la valeur maximale du compteur ?
- Faites clignoter l'ensemble des 4 LEDs disponibles suivant des fréquences différentes.  
Pour compléter le fichier « blackice-ii.pcf » :

```
set_io led1 71
set_io led2 67
set_io led3 68
set_io led4 70
```

## ■ ■ ■ Clignotement de LEDs suivant un motif

### 2 – On va animer les LEDs suivant une séquence de motifs choisis.

La synthèse sera faite avec le contenu du fichier « circuit.v » :

```
// Version avec FSM
// `include "led_sequencer.v"
// Version avec memoire simulee
// `include "led_sequencer_memory.v"
// Version avec memoire BRAM
`include "led_sequencer_memory_bram.v"

`include "clock_divider.v"

module circuit (
    input clock,
    input reset,
    output [3:0] LEDS,
    output sortie_clock,
    input bouton
);

    wire clock_interne;
    assign sortie_clock = clock_interne;

    clock_divider cd (
        .clock(clock),
        .slow_clock(clock_interne),
        .reset(reset)
    );

    led_sequencer ls (
        .slow_clock(clock_interne),
        .clock(clock),
        .reset(reset),
        .LEDS(LEDS)
    );
endmodule
```

Le contenu du fichier « clock\_divider.v » :

```
// HORLOGE_ENTREE 100_000_000
// HORLOGE_SORTIE 4
// VALEURCOMPTEUR = HORLOGE_ENTREE/HORLOGE_SORTIE = 25000000

`ifndef VALEURCOMPTEUR
    `define HORLOGE_ENTREE 100_000_000
    `define HORLOGE_SORTIE 16
    `define VALEURCOMPTEUR (`HORLOGE_ENTREE/`HORLOGE_SORTIE)
`endif

module clock_divider(input clock, output reg slow_clock, input reset);
    parameter COMPTEUR = `VALEURCOMPTEUR;
    reg [$clog2(COMPTEUR+1):0] compteur = 0;

    always @(posedge clock) begin
        if (reset) begin
            compteur <= 0;
            slow_clock <= 0;
        end
        else begin
            if (compteur == COMPTEUR) begin
                compteur <= 0;
                // version en clock derivee => generer une horloge
                // slow_clock <= ~ slow_clock;
                // version en clock enabled => generer une impulsion
                slow_clock <= 1;
            end
            else begin
                compteur <= compteur + 1;
                // version en clock enabled => generer une impulsion
                slow_clock <= 0;
            end
        end
    end
end
endmodule
```

La version avec un FSM, dans le fichier « led\_sequencer.v »:

```
`define NOMBRE_ETATS 5

module led_sequencer(
    input clock,
    input slow_clock,
    input reset,
    output reg [3:0] LEDS);

    // Les etats de l'automate
    /* reg [$clog2(`NOMBRE_ETATS)-1:0] etat; */
    /* reg [3:0] etat = DEBUT; */
    reg [3:0] etat; // = DEBUT;

    parameter DEBUT    = 4'd 0;
    parameter ETAPE1   = 4'd 1;
    parameter ETAPE2   = 4'd 2;
    parameter ETAPE3   = 4'd 3;
    parameter ETAPE4   = 4'd 4;

    // les configurations des LEDs
    parameter CONFIG_LED_0 = 4'b 0000;
    parameter CONFIG_LED_1 = 4'b 1000;
    parameter CONFIG_LED_2 = 4'b 0100;
    parameter CONFIG_LED_3 = 4'b 0010;
    parameter CONFIG_LED_4 = 4'b 0001;

    // piloter le circuit par la slow_clock
    always @(posedge slow_clock) begin
        // piloter le circuit par l'horloge globale => version clock enabled
        // always @(posedge clock) begin
            if (reset) begin
                LEDS <= 4'b 1111;
                etat <= DEBUT;
            end
        else begin
            // pour la version "clock enabled"
            // if (slow_clock) begin
            case (etat)
                DEBUT: begin
                    LEDS <= CONFIG_LED_0;
                    etat <= ETAPE1;
                end
                ETAPE1: begin
                    LEDS <= CONFIG_LED_1;
                    etat <= ETAPE2;
                end
                ETAPE2: begin
                    LEDS <= CONFIG_LED_2;
                    etat <= ETAPE3;
                end
                ETAPE3: begin
                    LEDS <= CONFIG_LED_3;
                    etat <= ETAPE4;
                end
                ETAPE4: begin
                    LEDS <= CONFIG_LED_4;
                    etat <= DEBUT;
                end
                default:
                    etat <= DEBUT;
            endcase
            // pour la version "clock enabled"
        // end
    end
endmodule
```

La version utilisant de la RAM avec le contenu du fichier « *led\_sequencer\_memory.v* » :

```
module led_sequencer(
    input clock,
    input slow_clock,
    input reset,
    output reg [3:0] LEDS);

    reg [3:0] config_leds [9:0];
    reg [3:0] adresse = 0;

    // les configurations des LEDs

    always @(posedge slow_clock) begin
        if (reset) begin
            adresse <= 0;
            config_leds[0] <= 4'b 0000;
            config_leds[1] <= 4'b 1000;
            config_leds[2] <= 4'b 0100;
            config_leds[3] <= 4'b 0010;
            config_leds[4] <= 4'b 0001;
            config_leds[5] <= 4'b 0010;
            config_leds[6] <= 4'b 0100;
            config_leds[7] <= 4'b 1000;
        end
        else begin
            if (adresse == 7)
                adresse <= 0;
            else
                adresse <= adresse + 1;
            LEDS <= config_leds[adresse];
        end
    end
endmodule
```

*Ici, on parcourt simplement la mémoire pour appliquer chaque motif sur les LEDs.*

La version utilisant de la **BRAM** dans le fichier « *led\_sequencer\_memory\_bram.v* » :

```
module memoire #(
    parameter WORD = 8,
    parameter TAILLE = 10
)
(input clock,
    input w_en,
    input r_en,
    input [TAILLE_ADRESSE - 1:0] w_addr,
    input [TAILLE_ADRESSE - 1:0] r_addr,
    input [WORD - 1:0] w_data,
    output reg [WORD - 1:0] r_data
);
    localparam TAILLE_ADRESSE = $clog2(TAILLE);
    reg [WORD - 1:0] mem [0:TAILLE - 1];
    always @(posedge clock) begin
        if (w_en == 1'b1) begin
            mem[w_addr] <= w_data;
        end
        if (r_en == 1'b1) begin
            r_data <= mem[r_addr];
        end
    end
endmodule

module led_sequencer(
    input clock,
    input slow_clock,
    input reset,
    output reg [3:0] LEDS);

    reg w_en = 0;
    reg r_en = 1;
    reg [3:0] r_addr;
    reg [3:0] w_addr = 0;
    reg [7:0] w_data;
    wire [7:0] r_data;

    memoire #(.WORD(8), .TAILLE(10))
    config_leds
    (
        .clock(clock),
        .w_en(w_en),
```

```

.r_en(r_en),
.w_addr(w_addr),
.r_addr(r_addr),
.w_data(w_data),
.r_data(r_data)
);

reg [3:0] adresse_lecture = 0;

// charger la memoire
localparam WRITE_0 = 0;
localparam WRITE_1 = 1;
localparam WRITE_2 = 2;
localparam WRITE_3 = 3;
localparam WRITE_4 = 4;
localparam WRITE_5 = 5;
localparam WRITE_6 = 6;
localparam WRITE_7 = 7;
localparam FIN = 8;

reg [3:0] etat = WRITE_0;
// les configurations des LEDs

always @(posedge clock or posedge reset) begin
  if (reset) begin
    w_en <= 0;
    etat <= WRITE_0;
  end
  else
    if (slow_clock) begin
      case(etat)
        WRITE_0: begin
          w_en <= 1;
          w_addr <= 0;
          w_data <= 8'b 00000000;
          etat <= WRITE_1;
        end
        WRITE_1: begin
          w_en <= 1;
          w_addr <= 1;
          w_data <= 8'b 00001000;
          etat <= WRITE_2;
        end
        WRITE_2: begin
          w_en <= 1;
          w_addr <= 2;
          w_data <= 8'b 00000100;
          etat <= WRITE_3;
        end
        WRITE_3: begin
          w_en <= 1;
          w_addr <= 3;
          w_data <= 8'b 00000010;
          etat <= WRITE_4;
        end
        WRITE_4: begin
          w_en <= 1;
          w_addr <= 4;
          w_data <= 8'b 00000001;
          etat <= WRITE_5;
        end
        WRITE_5: begin
          w_en <= 1;
          w_addr <= 5;
          w_data <= 8'b 00000010;
          etat <= WRITE_6;
        end
        WRITE_6: begin
          w_en <= 1;
          w_addr <= 6;
          w_data <= 8'b 00000100;
          etat <= WRITE_7;
        end
        WRITE_7: begin
          w_en <= 1;
          w_addr <= 7;
          w_data <= 8'b 00001000;
          etat <= FIN;
        end
        FIN: begin
          w_en <= 0;
        end
      endcase
    end
  end
end

```

```

        endcase
    end
end

always @(posedge clock) begin
    if (reset) begin
        adresse_lecture <= 0;
        r_en <= 0;
    end
    else
        if (slow_clock && (etat == FIN)) begin
            if (adresse_lecture == 8) begin
                adresse_lecture <= 0;
            end
            else begin
                r_en <= 1;
                r_addr <= adresse_lecture;
                LEDS <= r_data[3:0];
                adresse_lecture <= adresse_lecture + 1;
            end
        end
    end
end
endmodule

```

*Vous noterez qu'il faut un FSM pour initialiser le contenu de la mémoire initialement.*

### Questions :

- Comparez la synthèse des différentes solutions en terme d'occupation des ressources du FPGA par les différentes solutions.  
Quelles sont les différentes maximales atteignables par les différentes solutions ?
- Comment ces différentes solutions peuvent être étendues (ajouter plus de motifs pour de nouvelles animations) ?  
Quelle est la solution la plus pratique ?
- Dans le fichier «blackice-ii.pcf» : comment sont gérées les LEDs ?
- Quelles sont les versions les plus faciles à simuler ?

```

xterm
$ git clone https://git.p-fb.net/PeFClic/fpga_led_sequencer.git

```