Faculté des Sciences & Techniques — Université de Limoges

Master 2nde année

TMC

TP 1

Installation et utilisation des outils de simulation et de synthèse

Simulation

1 – a. Vous installerez les outils suivants :

```
xterm
$ sudo apt install iverilog
$ sudo apt install gtkwave
```

Vous disposerez:

- ▷ d'un simulateur de circuit décrit en verilog avec « *icarus verilog* » ;
- ▷ d'un visualiseur de signaux digitaux avec « *gtkwave* » ;

Soit le code suivant dans le fichier « `simple_circuit.v` »:

```
`timescale 1ns / 1ps

module example_design (
      input clock, input reset,
      input entree,
      output reg sortie);

always @(posedge clock) begin
  if (reset)
    sortie <= 0;
  else
    sortie <= entree;
end
endmodule

module example_tb ();
  // Clock and reset signals
  reg clk;
  reg reset;
  // Design Inputs and outputs
  reg  in_d;
  wire out_q;

  // DUT instantiation
  example_design dut (
    .clock  (clk),
    .reset  (reset),
    .entree (in_d),
    .sortie (out_q)
  );

  // generate the clock
  initial begin
    clk = 1'b1;
    forever #5 clk = ~clk;
  end

  // Generate the reset
  initial begin
    reset = 1'b1;
    // wait for the next rising edge
    // of the clock
    repeat(1)@(posedge clk);
    reset = 1'b0;
  end
```

```
// Test stimulus
initial begin
  // Use the monitor task to display the FPGA IO
  $monitor("time=%3d, in_d=%b, out_q=%2b",
        $time, in_d, out_q);
  $dumpfile("simple_circuit.vcd");
  $dumpvars(0, example_tb);

  // Generate each input with a 2 clock cycles,
  // i.e. 20ns
  in_d <= 1'b0;
  repeat(2)@(posedge clk);
  in_d <= 1'b1;
  repeat(2)@(posedge clk);
  in_d <= 1'b0;
  repeat(2)@(posedge clk);
  in_d <= 1'b1;
  #30 $finish;
end
endmodule
```

Récupération des sources

Vous pouvez récupérer le contenu de ces exercices sur :
`https://git.p-fb.net/PeFClic/fpga_exos_flipflops.git`

b. Vous lancerez la simulation et observerez la sortie avec :

```
xterm
iverilog -o simple_circuit -s example_tb simple_circuit.v
./simple_circuit
gtkwave simple_circuit.vcd --script=gtkwave.tcl
```

c. Est-ce conforme à votre idée ?

d. Quelle sorte de circuit logique avez vous construit ?

2 – Soit le code suivant dans le fichier « `simple_circuit.v` »:

```
`timescale 1ns / 1ps

module example_design (
   input clock,
    input reset,
    input entree,
    output reg sortie);
   always @(posedge clock) begin
      if (reset)
        sortie <= 0;
      else
      sortie <= entree ;
   end
endmodule

module example_tb ();
  // Clock and reset signals
  reg clk;
  reg reset;
  // Design Inputs and outputs
  reg in_d;
  wire out_q;

  // DUT instantiation
  example_design dut (
    .clock  (clk),
    .reset  (reset),
    .entree (in_d),
    .sortie (out_q)
  );

  // generate the clock
  initial begin
    clk = 1'b1;
    forever #5 clk = ~clk;
  end

  // Generate the reset
  initial begin
    reset = 1'b1;
    repeat (1) @(posedge clk);
    reset = 1'b0;
  end
```

```
// Test stimulus
initial begin
  // Use the monitor task to display the FPGA IO
  $monitor("time=%3d, in_d=%b, q=%2b",
            $time, in_d, out_q);

  $dumpfile("simple_circuit.vcd");
  $dumpvars(0, example_tb);
  in_d = 1'b0;
  #22
  in_d = 1'b1;
  #22
  in_d = 1'b0;
  #23
  in_d = 1'b1;
  #30 $finish;
end
endmodule
```

a. Que pouvez vous observer ?

b. Quelles sont les différences avec l'exercice 1 ?
Expliquez ces différences.

Synthèse d'un circuit et programmation FPGA

3 – a. Vous installerez les outils de synthèse suivant
(ou méthode alternative à `https://github.com/YosysHQ/oss-cad-suite-build`) :

xterm
```
$ sudo apt install yosys
$ sudo apt install fpga-icestorm
$ sudo apt install nextpnr-ice40-qt
```

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

Soit le code suivant dans le fichier « `led_allumee.v` » :

```
module circuit (output led1);
  assign led1 = 1;
endmodule
```

Soit le contenu du fichier « `blackice-ii.pcf` » :

```
set_io led1 71
```

Et le contenu du « `Makefile` » :

```
VERILOG_SRC=led_allumee.v
PROJECT=$(basename $(VERILOG_SRC))
ARCH=hx8k
CLOCK=16
PACKAGE=tq144:4k
PCF=blackice-ii.pcf
PORT=/dev/ttyACM0

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

all: $(PROJECT).bin

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

b. Vous exécuterez :

xterm
```
$ make show
```

c. Que pouvez vous observer ?
Expliquez ce que vous voyez.

d. Après avoir connecté la carte « *blackice II* » sur votre machine, en utilisant le port USB de programmation (celui plus « *central* »), vous ferez :

xterm
```
$ make prog
```

Que pouvez vous observer ?

Voici maintenant le code du fichier « `led_bouton.v` » :

```
module circuit (input B1, output led1);
  assign led1 = B1;
endmodule
```

4 – a. Que pouvez vous observez en :
- ◇ l'observant dans « `nextpnr` » ?
- ◇ le programmant dans le FPGA ?

b. Réalisez le circuit inverse.

c. Quelle sorte de circuit logique avez vous réalisé ?

d. Réalisez un « *testbench* » avec le simulateur « *iverilog* » et étudiez la trace obtenue avec « `gtkwave` ».