MICROPROCESSOR LAB EXPERIMENT 2

Deenabandhan N ee23b021
Sai Harshith Gajendra ee23b069
Krutarth Patel ee23b137

14 August 2024

Introduction

FPGA board : Edge Artix 7

This experiment involves

- Simulating a half-adder using Xilinx Vivado and implementing on the FPGA board.
- Extending the half-adder design to a full-adder, simulating it and implementing on the FPGA board.
- Designing a 4-bit ripple-carry adder and implementing on the FPGA board.

Xilinx Vivado

- We were introduced to Xilinx vivado , a software that is used to synthesize and analyse the hardware description designs.
- The procedures that we followed are ,
  - Create a project with source file as our Verilog code.
  - Add constraints to the ports that we have defined in the Verilog code.
  - Run the synthesis to check whether our Verilog code has any error in it.
  - We can open the Schematics to see the design that we have coded in Verilog.
  - After running synthesis , we can run simulation using our testbench and check for logical errors.
  - Once we confirm there is no logical/syntax error , we can run the implementation which basically implements our hardware description in the board which we have chosen while creating the project.
  - After the implementation is done , we can generate bitstream which is basically a file that contains the configuration information for an FPGA.
  - Once we successfully generate the bitstream , we can connect the target source and program it with the generated bitstream.
- In this report , we have included
  - Verilog code for module instantiation.
  - We have included both data flow as well as gate level modelling here.
  - The Schematics generated from Xilinx Vivado.
  - The Constraints file generated for FPGA.
  - The testbench and simulation generated
  - The analysis of reports obtained from Xilinx Vivado.

DFlipflop

Aim :

- The aim of this part of experiment is to stimulate a D-Flip flop using Verilog code.
- We have just stimulated in verilog and not implemented it in FPGA. This will be used in the following part of Johnson counter.
- We have used behavioural model to stimulate it and used positive edge of the clock to trigger the flip flop.
- We have tried out two different possibilities of reset button.
  - The reset button can be used as an asynchronous button with negative edge triggered i.e when reset goes from 1 to 0 irrespective of the clock position , it resets to zero.
  - The reset button can be used as a synchronous button which resets only when the clock is positive edge triggered.

Code for synchronous reset

```
module D_FF(input D,clk,rst,output Q,Qbar);
        reg Q,Qbar;
        always@(posedge clk)
        begin
                if(rst==1'b0)
                begin
                        Q=0;
                        Qbar=1;
                end
                else
                begin
                        Q<=D;
                        Qbar<=(~(D));
                end
        end
endmodule
```

Schematics for synchronous reset

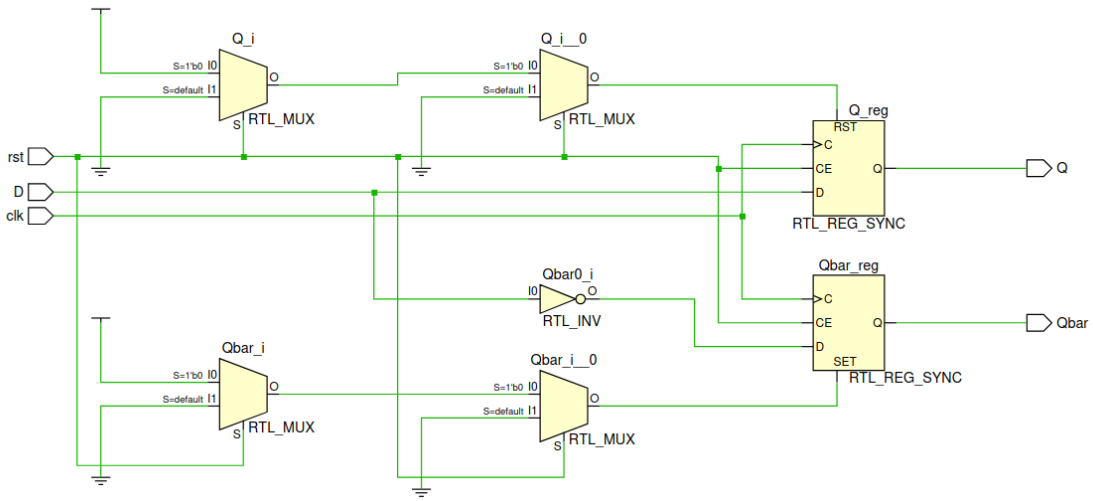


Figure 1: Schematics of the D-Flipflop with synchronous reset

Code for asynchronous reset

```
module D_FF(input D,clk,rst,output Q,Qbar);
        reg Q,Qbar;
        always@(posedge clk , negedge rst)
        begin
                if(rst==1'b0)
                begin
                        Q=0;
                        Qbar=1;
                end
                else
                begin
                        Q<=D;
                        Qbar<=(~(D));
                end
        end
endmodule
```

Schematics for Asynchronous reset

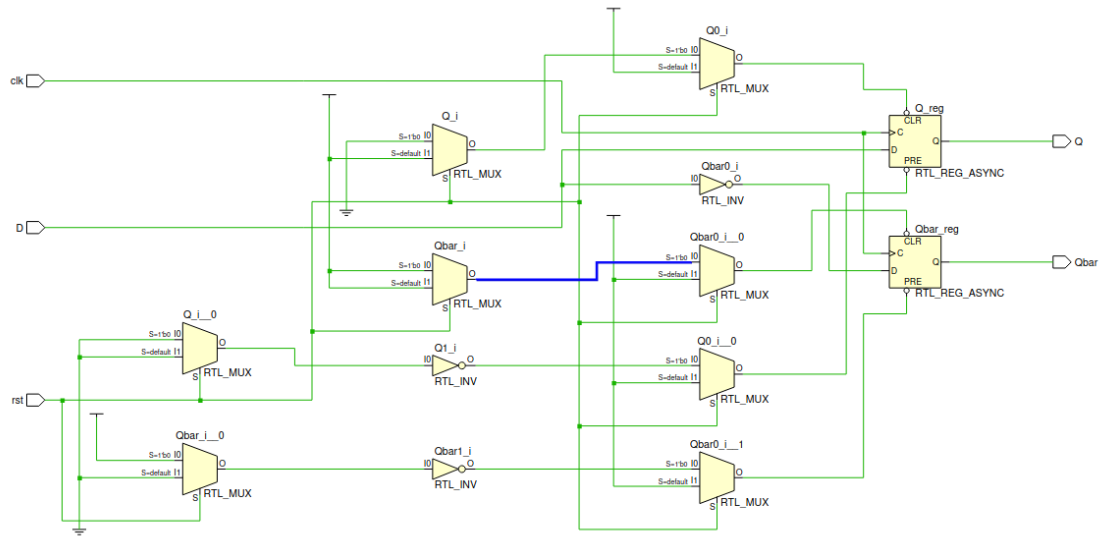


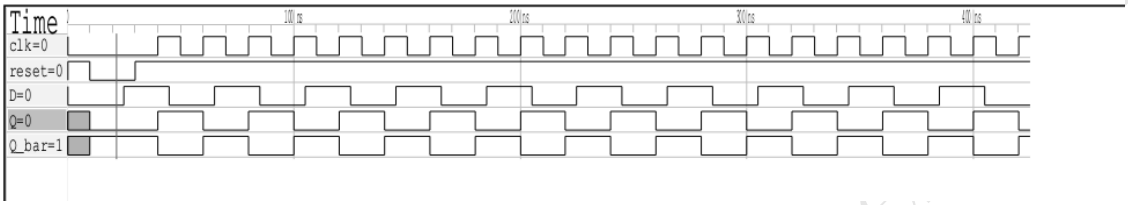
Figure 2: Schematics of the D-Flipflop with asynchronous reset

Testbench

```
`timescale 1ns/100ps

module test;
        reg clk,reset,D;
        wire Q,Q_bar;

        D_FF d(D,clk,reset,Q,Q_bar);

        initial
        begin
                clk=0;reset=1;D=0;
                #10 reset=0; // This negative edge trigger will cause initial values to get assigned to 0
                #20 reset=1;
                repeat(40) // The number of cycles
                begin
                        #10 clk=~(clk); // Inverting the clock pulse
                end
        end
```

```
        initial
        begin
                #5;
                repeat(20)
                        #20 D=~(D);

                #20 $finish;
        end
endmodule
```

Simulation



Clock Divider

```
module clk_gen(input clk,reset,output out);
        reg out;
        reg [3:0] count=0;

        always@(posedge clk or negedge reset)
        begin
                if(reset==1'b0)
                begin
                        out=0;
                        count=0;
                end
                else
                begin
                        count+=1;
                        if(count==5)
                        begin
                                out=~(out);
                                count=0;
                        end
                end
        end
endmodule
```

Johnson Counter

Behavioral Model

```
module decoder(input[2:0] cntr,output [7:0] led);
        reg [6:0] val;
        assign led={1'b1,~(val)};

        always@(cntr)
        begin
                case(cntr)
                3'd0 : val=7'b0111111;
                3'd1 : val=7'b0000110;
                3'd2 : val=7'b1011011;
                3'd3 : val=7'b1001111;
                3'd4 : val=7'b1100110;
                3'd5 : val=7'b1101101;
                3'd6 : val=7'b1111101;
                3'd7 : val=7'b0000111;
                endcase
        end
endmodule

module Johnson_count(input clk_in,reset,input [3:0] digit,output [7:0] led);
        wire [2:0] cntr;
        wire q0,q1,q2;
        wire q0_bar,q1_bar,q2_bar;
        wire clk_out;

        assign digit=4'b0001;

        clk_gen c1(clk_in,reset,clk_out);

        D_FF d1(q0_bar,clk_out,reset,q2,q2_bar);
        D_FF d2(q2,clk_out,reset,q1,q1_bar);
        D_FF d3(q1,clk_out,reset,q0,q0_bar);

        assign cntr={q2,q1,q0};

        decoder de1(cntr,led);

endmodule
```

Schematics :

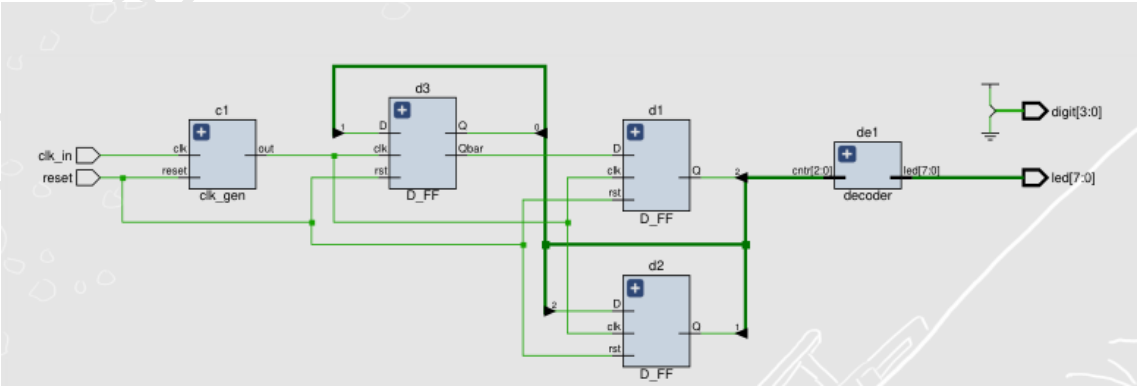


Figure 3: Schematics of the Johnson Counter

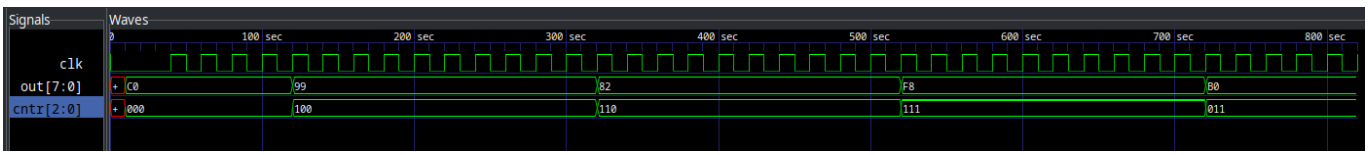
Testbench

```
    module test;
        reg clk,reset;
        wire [7:0] out;
        wire [3:0] digit;
        assign digit = 4'b0001;
        Johnson_count d(clk, reset, digit, out);

        initial
        begin
                clk=0;reset=1;
                #10 reset=0;
                #20 reset=1;
                repeat(80)
                begin
                        #10 clk=~(clk);
                end
        end
        initial
        begin
                #820 $finish;
        end
endmodule
```

Constraints on ports of FPGA

```
set_property IOSTANDARD LVCMOS33 [get_ports {digit[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {digit[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {digit[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {digit[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk_in]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN F2 [get_ports {digit[0]}]
set_property PACKAGE_PIN E1 [get_ports {digit[1]}]
set_property PACKAGE_PIN G5 [get_ports {digit[2]}]
set_property PACKAGE_PIN G4 [get_ports {digit[3]}]
set_property PACKAGE_PIN H1 [get_ports {led[7]}]
set_property PACKAGE_PIN H2 [get_ports {led[6]}]
set_property PACKAGE_PIN J4 [get_ports {led[5]}]
set_property PACKAGE_PIN J5 [get_ports {led[4]}]
set_property PACKAGE_PIN H4 [get_ports {led[3]}]
set_property PACKAGE_PIN H5 [get_ports {led[2]}]
set_property PACKAGE_PIN G1 [get_ports {led[1]}]
set_property PACKAGE_PIN G2 [get_ports {led[0]}]
set_property PACKAGE_PIN N11 [get_ports clk_in]
set_property PACKAGE_PIN L5 [get_ports reset]
```

Simulation :



Reports :

Resources utilized :

- 47 **SLICE LUT** were used, where LUT is used as a logic and not memory.
- 37 Slice Registers were used as Flip Flop.

Clock :

- only 1 Global Clock was utilized.

Clock Region Name	*Global Clock (Used)*	**FF (Used)**	**LUTM (Used)**
X0Y0	*0*	*0*	*0*
X1Y0	*0*	*0*	*0*
X0Y1	*0*	*0*	*0*
X1Y1	*1*	*37*	*6*
X0Y2	*0*	*0*	*0*
X1Y2	*0*	*0*	*0*

Time delays

Type of Path	*Path taken*	**Time delay (ns)**
Min Delay path	**d3 → d1**	*0.220*
	d2 → d3	*0.368*
	d1 → d2	*0.376*
Max Delay path	**c1/count_reg[5] → c1/count_reg[29]**	*8.051*
	c1/count_reg[5] → c1/count_reg[31]	*8.030*
	c1/count_reg[5] → c1/count_reg[30]	*7.956*

Power consumed

- The power consumed by FPGA is **0.325W**