Faculty of Engineering & Technology
Electrical & Computer Engineering Department

DIGITAL ELECTRONICS AND COMPUTER ORGANIZATION LABORATORY - ENCS2110

Report #5

Introduction to QUARTUS II Software

Prepared by:

Obada Tahayna 1191319

Instructor: Dr. Adnan Yahya
Assistant: Eng. Katy Sadi

Section: 5
Date: 27/10/2020

Abstract

The aim of the experiment is to use the QUARTUS II software to code in Verilog HDL and simulate the implemented circuit. In addition of the block diagram and turning the code into FPGA and test it .

Contents

Abstract .... أ
Theory 3
I. **Quartus II Software** 3
II. **Verilog HDL** 3
Procedure 4
I. **fullAdder** 4
II. **fourBitFullAdder** 5
III. **fourBitComparator** 6
IV. **twoByOneMux** 7
V. **The Complete System** 8
Conclusion 10
References 11

Theory

I. Quartus II Software

The Altera Quartus II design software provides a complete multiplatform design environment that easily adapts to your specific design needs. it's a comprehensive environment for system-on-a-programmable-chip (SOPC) design. The Quartus II software includes solutions for all phases of FPGA and CPLD design.[1]

II. Verilog HDL

Verilog is a hardware description language (HDL) used to model electronic systems, it's most commonly used in the design and verification of digital circuits at the register transfer level of abstraction *[2].* A Verilog can be used in Quartus II by creating a Verilog file. And a module can be described in any one (or a combination) of the following modeling techniques:

- ❖ **Gate-Level Modeling:** using instantiation of primitive gate and user defined modules.
- ❖ **Data-Flow Modeling:** using continues assignment statements with keyword assign.
- ❖ **Behavioral Modeling:** using procedural assignment statements with keyword always.

Procedure

In this experiment, a 4-bit full adder was constructed using 1-bit full adders then the comparator was created and the system was connected via 2x1 Mux.

I. fullAdder

The module was set up structurally as follows:

```
module fullAdder(a, b, cIn, sum, cOut);

    input a, b, cIn;
    output reg sum, cOut;


    always @(a, b, cIn)
    begin
        {cOut, sum} = a + b + cIn;
    end

endmodule
```

Figure 1: fullAdder Code

After simulating the module, the output came out as follows:

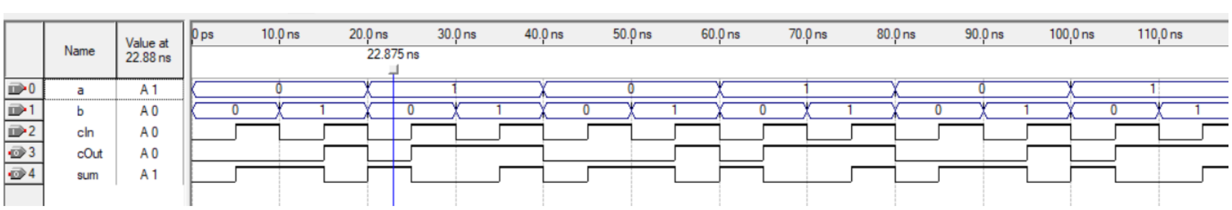

Figure 2: fullAdder Simulation

II. fourBitFullAdder

The 4-bit full adder was created by implementing four 1-bit full adder modules

```
module fourBitFullAdder(a, b, cIn, sum);

    input [3:0] a, b;
    input cIn;
    output [4:0] sum;

    wire w1, w2, w3;

    fullAdder f1(a[0], b[0], cIn, sum[0], w1);
    fullAdder f2(a[1], b[1], w1, sum[1], w2);
    fullAdder f3(a[2], b[2], w2, sum[2], w3);
    fullAdder f4(a[3], b[3], w3, sum[3], sum[4]);

endmodule
```

Figure 3: fourBitFullAdder Code

After simulating the module, the output came out as follows:

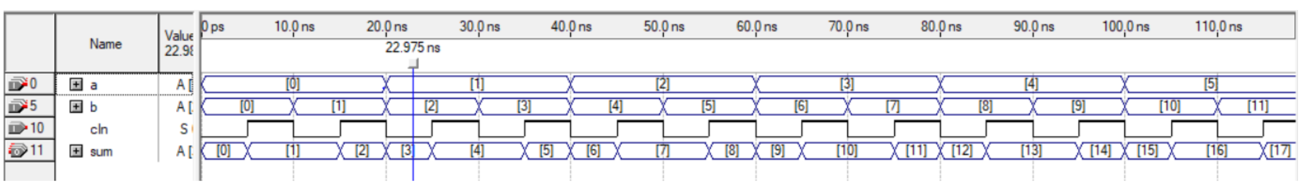

Figure 4: fourBitFullAdder Simulation

III. fourBitComparator

The code was written behaviorally to compare between two inputs, if the first input was greater then the output will be 100, if both inputs were equal then the output will be 010, else the output will be 001.

```
module fourBitComparator(a, b, result);

    input [3:0] a, b;
    output reg [2:0] result;

    always @(a, b)
    begin

        if(a>b)
        begin
            result = 3'b100;
        end


        else if(a==b)
        begin
            result = 3'b010;
        end


        else if(a<b)
        begin
            result = 3'b001;
        end

    end
endmodule
```

Figure 5: fourBitComparator Code

After simulating the code, results came as shown below.

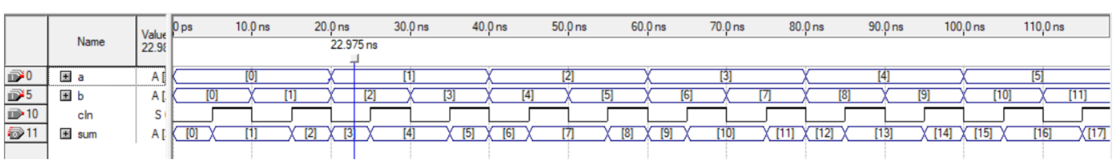

Figure 6: fourBitComparator Simulation

IV. twoByOneMux

The code was built behaviorally to choose between the two functions, if the selection input was 0 then the output will be the first option, else it will be the second one.

```
module twoByOneMux(d0, d1, sel, result);

    input [4:0] d0;
    input [2:0] d1;
    input sel;
    output reg [4:0] result;

    always @(d0, d1)
    begin

        if(sel)
            result = d1;

        else
            result = d0;

    end

endmodule
```

Figure 7: twoByOneMux Code

The figure below explains the output of the module.

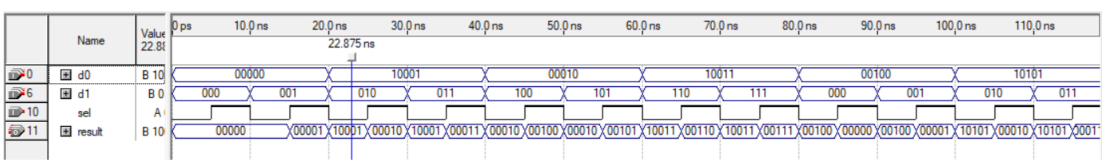

Figure 8: twoByOneMux Simulation

V. The Complete System

The System contains inputs, which are two 4-bit inputs, initial carry and the selection. The operation is executed depending on the selection input (adding or comparing), and outputs which are a 4-bit output and the final carry.

```
module project(a, b, cIn, select, out);

    input [3:0] a, b;
    input cIn, select;
    output [4:0] out;

    wire [4:0]  w1;
    wire [2:0] w2;


    fourBitFullAdder(a, b, cIn, w1);
    fourBitComparator(a, b, w2);

    twoByOneMux(w1, w2, select, out);

endmodule
```

Figure 9: The Complete System Code

If selection equals 0, adding will be initiated. Other than that, comparing will be initiated as explained below.

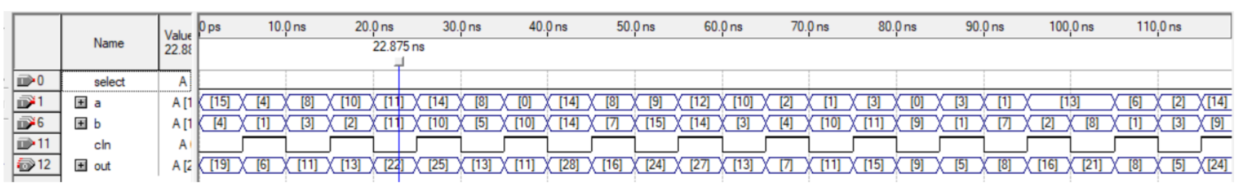

Figure 10: The System Simulation - Select=0 (Adding)

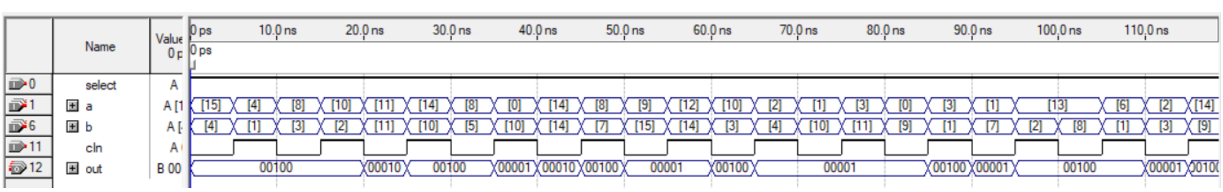

Figure 11: The System Simulation - Select=0 (Comparing)

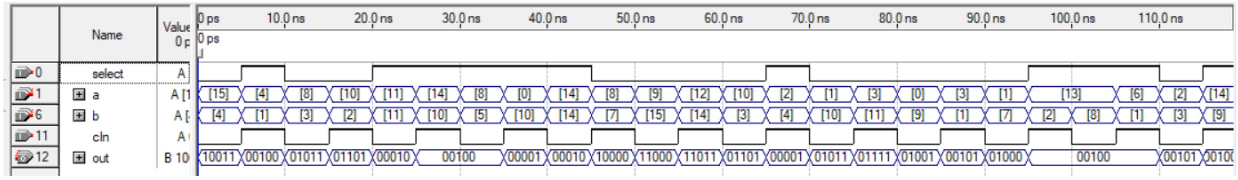

Figure 12: The System Simulation - Both Selections (Randomly)

After setting all the three modules, we created a block diagram and assigned the pins to convert the design into FPGA. the block diagram:

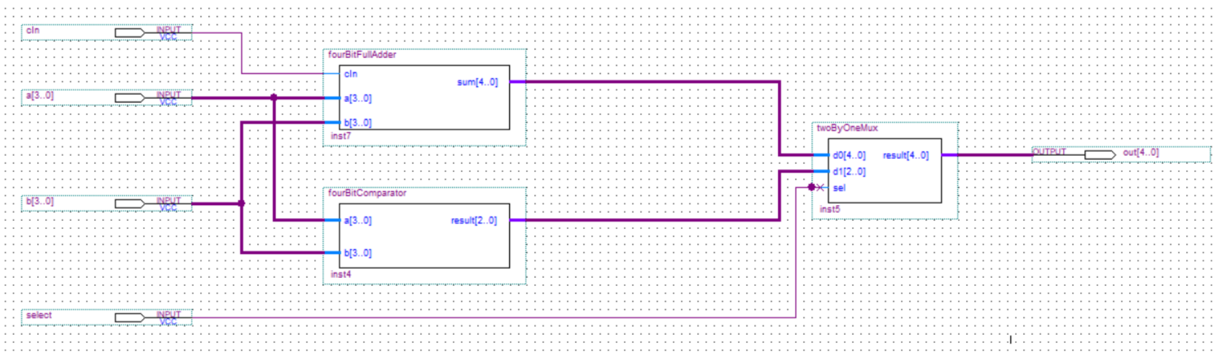

Figure 13: The System Schematic

Conclusion

Verilog HDL and FPGA are useful ways to implements designs easily and to avoid the gates complexity. Using these methods will give you the ability to implement complex designs just by defining the functionality of the design and without the gate level complexity. Also, it gives you the ability to simulate and test the design before implementing it on the hardware.

References

[1]: Introduction to the Quartus II software design Book Page 2.

[2]: https://en.wikipedia.org/wiki/Verilog Accessed on 26-10-2020 at 10:23pm