CENTER OF NANOELECTRONICS & DEVICES

CND 111: Introduction to Digital Design

Lab #: 1

Section #: 16

Submitted by:

Student Name	ID
Amr Emad Mokhtar Hamouda	V23009911
Karim Mahmoud Kamal	V23010174
Omar Kamal	

Submitted to TA: Mohamed El-Shafei

Date: 23/9/2023

1. Multi-Function Gate

i. A screenshot of the Verilog code of the behavioral level of abstraction:

```
module Multi_Function_Gate_Behavioral (XY, A, B, F);        input [1:0] XY;

        input A, B;

        output reg F;

        always @ (*) begin

                case (XY)

                        2'b00: F = (A & B);

                        2'b01: F = (A | B);

                        2'b10: F = (~(A & B));

                        2'b11: F = (~(A | B));

                endcase

        end
```

```
endmodule
```

ii. A screenshot of the Verilog code of the structural level of abstraction:

```
module Multi_Function_Gate_Structural (A, B, X, Y, F);

        input A, B, X, Y;

        output F;

        wire W1, W2, W3, W4;

        and A1 (W1, ~A, ~B, X);

        and A2 (W2, ~A, B, Y);

        and A3 (W3, A, B, ~X);

        and A4 (W4, A, ~B, Y);

        or O1 (F, W1, W2, W3, W4);
```

```
endmodule
```

iii. A screenshot of the Verilog code of the data flow level of abstraction:

```
module Multi_Function_Gate_Dataflow (A, B, X, Y, F);

        input A, B, X, Y;

        output F;

        assign F = (((~A) & (~B) & X) | ((~A) & B & Y) | (A & B & (~X)) | (A & (~B) & Y));

endmodule
```

iv. **Comment** on the results:

We had a problem with the license, so we couldn't run the simulation.

2. Half Adder:

i. A screenshot of the Verilog code of the behavioral level of abstraction:

```
module Half_Adder_Behavioral (A, B, Sum, Cout);

	input A, B;

	output reg Sum, Cout;

	always @ (*) begin
		case ({A, B})

			3'b00: begin
				Sum = 0;
				Cout = 0;
			end

			3'b01: begin
				Sum = 1;
				Cout = 0;

			3'b10: begin
				Sum = 1;
				Cout = 0;
			end

			3'b11: begin
				Sum = 0;
				Cout = 1;
			end

		endcase

	end

endmodule
```

ii. A screenshot of the Verilog code of the structural level of abstraction:

```
module Half_Adder_Structural (A, B, Sum, Cout);
        input A, B;
        output Sum, Cout;
        xor X1 (Sum, A, B);
        and A1 (Cout, A, B);
endmodule
```

iii. A screenshot of the Verilog code of the data flow level of abstraction:

```
module Half_Adder_DataFlow (A, B, Sum, Cout);

        input A, B;

        output Sum, Cout;

        assign Sum = (A ^ B);

        assign Cout = (A & B);

endmodule
```

iv. **Comment** on the results:

We had a problem with the license, so we couldn't run the simulation.

3. Half Subtractor:

i. A screenshot of the Verilog code of the behavioral level of abstraction:

```
module Half_Subtractor_Behavioral (A, B, D, Bout);
        input A, B;
        output reg D, Bout;
        always @ (*) begin
                case ({A, B})
                        3'b00: begin
                                D = 0;
                                Bout = 0;
                        end
                        3'b01: begin
                                D = 1;
                                Bout = 1;
                        end
                        3'b10: begin
                                D = 1;
                                Bout = 0;
                        end
                        3'b11: begin
                                D = 0;
                                Bout = 0;
                        end
                endcase
        end
endmodule
```

ii. A screenshot of the Verilog code of the structural level of abstraction:

```
module Half_Subtractor_Structural (A, B, D, Bout);
        input A, B;
        output D, Bout;
        wire W1;
        xor X1 (D, A, B);
        not N1 (W1, A);
        and A1 (Bout, W1, B);
endmodule
```

iii. A screenshot of the Verilog code of the data flow level of abstraction:

```
module Half_Subtractor_DataFlow (A, B, D, Bout);

        input A, B;

        output D, Bout;

        assign D = (A ^ B);

        assign Bout = ((~A) & B);

endmodule
```

iv. **Comment** on the results:

We had a problem with the license, so we couldn't run the simulation.

4. Full Adder:

i. A screenshot of the Verilog code of the behavioral level of abstraction:

```
module FullAdder_Behavioral (A, B, Cin, Sum,
Cout);

        input A, B, Cin;

        output reg Sum, Cout;

        always @ (*) begin

                case({A, B, Cin})
```

```
                    3'b000: begin
                            Sum = 1'b0;
                            Cout = 1'b0;
                    end

                    3'b001: begin
                            Sum = 1'b1;
                            Cout = 1'b0;
                    end

                    3'b010: begin
                            Sum = 1'b1;
                            Cout = 1'b0;
                    end

                    3'b011: begin
                            Sum = 1'b0;
                            Cout = 1'b1;
                    end

                    3'b100: begin
                            Sum = 1'b1;
                            Cout = 1'b0;
                    end

                    3'b101: begin
                            Sum = 1'b0;
                            Cout = 1'b1;
                    end

                    3'b110: begin
                            Sum = 1'b0;
                            Cout = 1'b1;
                    end

                    3'b111: begin
                            Sum = 1'b1;
                            Cout = 1'b1;
                    end

            endcase

    end

endmodule
```

ii. A screenshot of the Verilog code of the structural level of abstraction:

```
module FullAdder_Structural (A, B, Cin, Sum, Cout);
        input A, B, Cin;
        output sum, Cout;
        wire W1, W2, W3;
        xor X1 (W1, A, B);
        and A1 (W2, W1, Cin);
        xor X2 (Sum, W1, Cin);
        and A2 (W3, A, B);
        or O1 (Cout, W3, W2);
endmodule
```

iii. A screenshot of the Verilog code of the data flow level of abstraction:

```
module FullAdder_DataFlow (A, B, Cin, Sum, Cout);

        input A, B, Cin;

        output Sum, Cout;

        assign Sum = ((A ^ B) ^ Cin);

        assign Cout = ((A & B) | (Cin & (A ^ B)));

endmodule
```

iv. **Comment** on the results:

We had a problem with the license, so we couldn't run the simulation.

5. Full Subtractor:

i. A screenshot of the Verilog code of the behavioral level of abstraction:

```
module FullSubtractor_Behavioral (A, B, Bin, D, Bout);

	input A, B, Bin;

	output reg D, Bout;

	always @ (*) begin

		case({A, B, Bin})

			3'b000: begin
				D = 1'b0;
				Bout = 1'b0;
			end

			3'b001: begin
				D = 1'b1;
				Bout = 1'b1;
			end

			3'b010: begin
				D = 1'b1;
				Bout = 1'b1;
			end

			3'b011: begin
				D = 1'b0;
				Bout = 1'b1;
			end

			3'b100: begin
				D = 1'b1;
				Bout = 1'b0;
			end

			3'b101: begin
				D = 1'b0;
				Bout = 1'b0;
			end

			3'b110: begin
				D = 1'b0;
				Bout = 1'b0;
			end

			3'b111: begin
				D = 1'b1;
				Bout = 1'b1;
			end

		endcase

	end

endmodule
```

ii. A screenshot of the Verilog code of the structural level of abstraction:

```
module FullSubtractor_DataFlow (A, B, Bin, D, Bout);
        input A, B, Bin;
        output D, Bout;
        wire W1, W2, W3;
        xor X1 (W1, B, Bin);
        xor X2 (D, W1, A);
        and A1 (W2, Bin, ~B);
        and A2 (W3, A, ~W1);
        or O1 (Bout, W3, W2);
endmodule
```

iii. A screenshot of the Verilog code of the data flow level of abstraction:

```
module FullSubtractor_DataFlow (A, B, Bin, D, Bout);
        input A, B, Bin;
        output D, Bout;
        assign D = ((A ^ B) ^ Bin);
        assign Bout = ((~B) & Bin) | ( A & (~( B ^ Bin )) );
endmodule
```

iv. **Comment** on the results:

We had a problem with the license, so we couldn't run the simulation.

6. 4-Bit Binary Adder and Subtractor:

i. A screenshot of the Verilog code of the behavioral level of abstraction:

```
module FA_FS_behavioural ( in1, in2, Cin, sum, Cout, Diff, Borrow, m );

	input [3:0] in1, in2;
	input m, Cin;

	output [3:0] sum, Diff;
	output Cout,Borrow;

	reg [3:0] sum, Diff;
	reg Borrow,Cout;

	always @ (*) begin
	if (m)
	begin
		case({in1,in2,Cin})
			3'b0000_0000_0: begin
				sum=4'b0000;
				Cout=1'b0;
			end
			3'b0001_0001_0: begin
				sum=4'b0010;
				Cout=1'b0;
			end
			3'b0010_0010_0: begin
				sum=4'b0100;
				Cout=1'b0;
			end
			3'b0011_0011_0: begin
				sum=4'b0110;
				Cout=1'b0;
			end

			3'b0100_0100_0: begin
				sum=4'b1000;
				Cout=1'b0;
			end
			3'b0101_0101_0: begin
				sum=4'b1010;
				Cout=1'b0;
			end
			3'b0110_0110_0: begin
				sum=4'b1100;
				Cout=1'b0;
			end
			3'b0111_0111_0: begin
				sum=4'b1110;
				Cout=1'b0;
```

```
                    end

                    3'b1000_1000_0: begin
                            sum=4'b0000;
                            Cout=1'b1;
                    end
                    3'b1001_1001_0: begin
                            sum=4'b0010;
                            Cout=1'b1;
                    end
                    3'b1010_1010_0: begin
                            sum=4'b0100;
                            Cout=1'b1;
                    end
                    3'b1011_1011_0: begin
                            sum=4'b0110;
                            Cout=1'b1;
                    end

                    3'b1100_1100_0: begin
                            sum=4'b1000;
                            Cout=1'b1;
                    end
                    3'b1101_1101_0: begin
                            sum=4'b1010;
                            Cout=1'b1;
                    end
                    3'b1110_1110: begin
                            sum=4'b1100;
                            Cout=1'b1;
                    end
                    3'b1111_1111_0: begin
                            sum=4'b1110;
                            Cout=1'b1;
                    end

                    3'b1111_1111_1: begin
                            sum=4'b1111;
                            Cout=1'b1;
                    end
            endcase
            end
    else
            begin
             case({in1,in2,Cin})
                    3'b0001_0000_0: begin
                            Diff=4'b001;
```

```
                        Borrow=1'b0;
                    end
                    3'b0000_0001_0: begin
                        Diff=4'b1111;
                        Borrow=1'b0;
                    end
                    3'b1000_0001: begin
                        Diff=4'b0111;
                        Borrow=1'b1;
                    end
                    3'b0011_0001_0: begin
                        Diff=4'b0010;
                        Borrow=1'b0;
                    end

                    3'b0100_0100_0: begin
                        Diff=4'b0000;
                        Borrow=1'b0;
                    end
                endcase
                end
        end

endmodule
```

ii. A screenshot of the Verilog code of the structural level of abstraction:

```
// Structural
module FA_structural_fourBits ( in1, in2, Cin, sum, Cout, Diff, Borrow, m );

        input [3:0] in1, in2;
        input m, Cin;

        output [3:0] sum, Diff;
        output Cout,Borrow;

        reg [3:0] sum, Diff,sum_temp,Diff_temp;
        reg Borrow,Cout,Cout_temp,Borrow_temp;

        wire [3:0] w1, w2, w3,w4,w5,w6;

        xor x1 ( w1, in1, in2);
        and a1 ( w2, w1, Cin );
        xor x2 ( sum_temp, w2, Cin);

        and a2 ( w3, in1, in2);
        or o1  ( Cout_temp, w3, w2);


        xor x11 ( w4, in2, Cin);
        xor a11 ( Diff, w4, in1);

        and a22 ( w5, Cin, ~in2);
        and a33 ( w6, in1, ~w4);
        or o11  ( Borrow, w6, w5);

        always @(*) begin
                if( m ==0 ) begin
                        sum = sum_temp;
                        Cout = Cout_temp;
                        Diff= 0;
                        Borrow=0;
                end
                else begin
                        sum = 0;
                        Cout = 0;
```

```
46.                         Diff= Diff_temp;
47.                         Borrow=Borrow_temp;
48.                 end
49.
50.         end
51.
52.     endmodule
```

i. A screenshot of the Verilog code of the data flow level of abstraction:

```
53.   // DataFlow
54.   module        FA_FS_dataFlow_fourBits ( in1, in2, Cin,sum, Cout, Diff, Borrow );
55.
56.         input [3:0] in1, in2, Cin;
57.         output [3:0] Diff,sum;
58.         output Borrow,Cout;
59.
60.         assign Diff = in1 ^ in2 ^ Cin;
61.         assign Borrow = (~in2 & Cin) | ( in1 & ~( in2 ^ Cin ) );
62.
63.         assign sum = in1 ^ in2 ^ Cin;
64.         assign Cout = (in1 & in2 ) | (Cin & ( in1 ^ in2 ));
65.
66.   endmodule
```

i. **Comment** on the results:

We had a problem with the license, so we couldn't run the simulation.

67. 32-Bit ALU:

i. A screenshot of the Verilog code of the data flow level of abstraction:

```
module ALU_Behavioral (A, B, S0, S1, S2, F);
        input [31:0] A, B;
        input S0, S1, S2;
        output reg [31:0] F;
        always @ (*) begin
                case({S2, S1, S0})
                        3'b000: F = A & B;
                        3'b001: F = A | B;
                        3'b010: F = A ^ B;
                        3'b011: F = ~(A & B);
                        3'b100: F = A * B;
                        3'b101: F = A / B;
                        3'b110: F = A + B;
                        3'b111: F = A - B;
                endcase       end
endmodule
```