Lab 2
Combinational Circuits #1

Objective

Design, model, and simulate combinational circuits using behavioral, structural, and data flow description methodologies.

1- Multi-Function Gate

In this lab, the Multi-Function gate with two inputs, ***A*** and ***B***, and a single output, ***F,*** can be designed to perform four different logic operations by setting the control values of ***X*** and ***Y***. Figure 1 shows the block diagram and the circuit of such a gate.

The boolean expression of the Multi-Function Gate:

$$F = A'B'\,X + A'BY + ABX' + AB'Y$$

The function codes are summarized in the table below.

X	Y	F
0	0	A **AND** B
0	1	A **OR** B
1	0	A **NOR** B
1	1	A **NAND** B

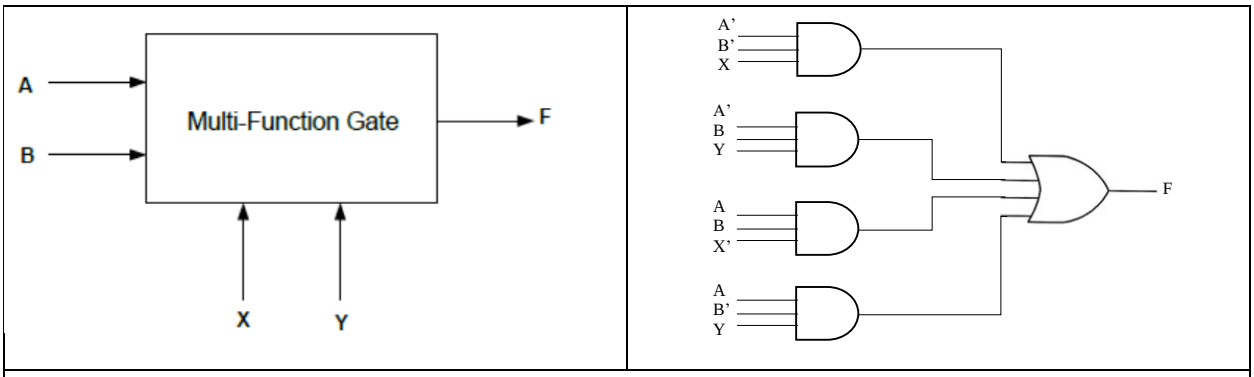


Figure 1. (a) Logic Symbol of Multi-Function Gate, (b) Circuit Diagram of Multi-Function Gate

Procedure:

1. Open Quartus software to design and simulate the following structural modeling for the multi-gate. Apply different patterns for the inputs and control signals and observe the output. Use Questa for the simulation.

Code 1: Gate/Structural Verilog model for Multi-Function Gate

```
module multi_fun_Structural (A, B, X, Y, F);
input A, B, X, Y;
output F;

wire w1, w2, w3, w4;

and g1(w1, ~A, ~B, X);
and g2(w2, ~A, B, Y);
and g3(w3, A, B, ~X);
and g4(w4, A, ~B, Y);
or  g5(F, w1, w2, w3, w4);

endmodule
```

2. Using modeling as per the code below, for data flow description. Use Questa for the simulation. Verify that both designs (data flow and structural) give the same results.

Code 2: Data flow Verilog model for Multi-Function Gate

```
module multi_fun_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
```

3. Using behavioral modeling as per the code below. Use Questa for the simulation. Verify that the three designs (behavioral, structural, and the data flow) give the same results.

Code 3: Behavioral Verilog model for Multi-Function Gate

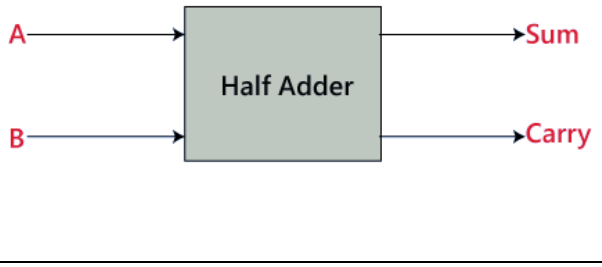
```
module multi_fun_behavioral (XY,A,B,F);

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);
   Default: F = A&B;
   endcase
  end
endmodule
```

2- Half Adder

Half adder is a combinational arithmetic circuit that adds two numbers and produces a sum bit (s) and carry bit (c) both as output. The addition of two bits is done using a combination circuit called a Half adder. The half-adder can be implemented using basic gates such as XOR and AND gates.
The SUM output is the least significant bit (LSB) of the result, which is the XOR of the two inputs A and B. The XOR gate implements the addition operation for binary digits, where a "1" is generated in the SUM output only when one of the inputs is "1".

The CARRY output is the most significant bit (MSB) of the result, indicating whether there was a carry-over from adding the two inputs. The CARRY output is the AND of the two inputs, A and B. The AND gate generates a "1" in the CARRY output only when both inputs are "1". Let us consider two input bits, A and B, and then sum bit (s) is the X-OR of A and B. It is evident from the function of a half adder that it requires one X-OR gate and one AND gate for its construction.

Sum = A XOR B
Carry = A AND B



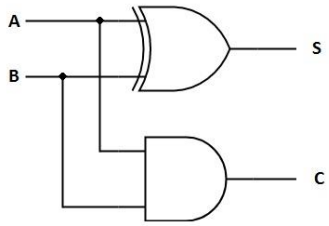


Figure 2. (a) Logic Symbol of Half Adder, (b) Circuit Diagram of Half Adder

Procedure:

- Open Quartus software to design and simulate the following three codes for the half adder apply different patterns for the inputs and control signals and observe the output. Use Questa for the simulation.

Code 4: Gate/Structural Verilog model for Half Adder

```
module half_adder_gate (A, B, C, S);
 input A, B;
 output C, S;
 xor g1(S, A, B);
 and g2(C, A, B);
endmodule
```

Code 5: Data flow Verilog model for Half Adder

```
module half_adder_dataflow (A, B, C, S);
 input A, B;
 output C, S;
 assign S=A ^ B;
 assign C=A & B;
endmodule
```

Code 6: Behavioral Verilog model for Half Adder

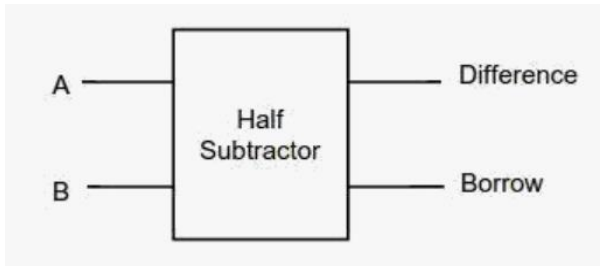
```
module half_adder_behavioral (A, B, S, C);
 input A, B;
 output reg S, C;

 always @ (*)
  begin

  case ({A, B})
  3'b00: S = 0;
  3'b01: S = 1;
  3'b10: S = 1;
  3'b11: S = 0;
  endcase

  case ({A, B})
  3'b00: C = 0;
  3'b01: C = 0;
  3'b10: C = 0;
  3'b11: C = 1;
  endcase
  end
endmodule
```

3- Half Subtractor

A half subtractor is a combinational circuit with two inputs and two outputs that produce the difference between the two binary bits at the input. The half subtractor can be implemented using basic gates such as XOR and NOT gates. The Boolean expression of the Half subtractor:

Difference = A' B +A B'

Borrow = A' B



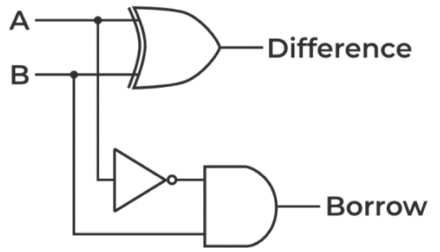


Figure 3. (a) Logic Symbol of Half subtractor , (b) Circuit Diagram of Half subtractor

Procedure:

Open Quartus software to design and simulate the following three codes for the half subtractor apply different patterns for the inputs and control signals and observe the output. Use Questa for the simulation.

Code 7: Structural Verilog model for Half Subtractor

```
module half_subtractor_gate (A, B, Diff, Borrow);

 input A, B;
 output Diff, Borrow;
 wire w1;

 xor g1(Diff, A, B);
 not g2(w1, A);
 and g3(Borrow, w1, B);
endmodule
```

Code 8: Data Flow Verilog model for Half Subtractor

```
module half_subtractor_dataflow (A, B, Diff, Borrow);

 input A, B;
 output Diff, Borrow;
 assign Diff= A ^ B;
 assign Borrow= ~A & B;
endmodule
```

Code 9: Behavioral Verilog model for Half Subtractor

```
module half_subtractor_behavioral (A, B, Diff, Borrow);
 input A, B;
 output reg Diff, Borrow;

 always @ (*)
  begin

  case ({A, B})
  3'b00: Diff = 0;
  3'b01: Diff = 1;
  3'b10: Diff = 1;
  3'b11: Diff = 0;
  endcase

  case ({A, B})
  3'b00: Borrow = 0;
  3'b01: Borrow = 1;
  3'b10: Borrow = 0;
  3'b11: Borrow = 0;
  endcase
  end
endmodule
```

Assignment

1. Write structural, behavioral, and data flow models for the following:
   - full adder.
   - full subtractor

   - **Bonus** – 32-bit ALU with at least 8 logic and arithmetic operations

Deliverables

- Verilog codes for all designs with screenshots for simulated waveform including all test cases. Also, screenshots for the RTL and synthesis results of all designs