COAL Lab # 01

Verilog (HDL) Programing Fundamentals

Name: Saad Nisar Butt

Reg. no: cs211246

Class: BS-CS 3C-1

<u>Lab Exercises:</u>

Combinational Circuits:

Literature Review:

Combinational circuits are made up of basic logic gates such as and, or and not gates. A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs. There operations are determined by set of Boolean functions. Outputs of the combinational logic circuits depend only on the present inputs. Here, we have three inputs, two are XORed and that two are ANDed and both outputs with another input are ORed to perform certain logic.

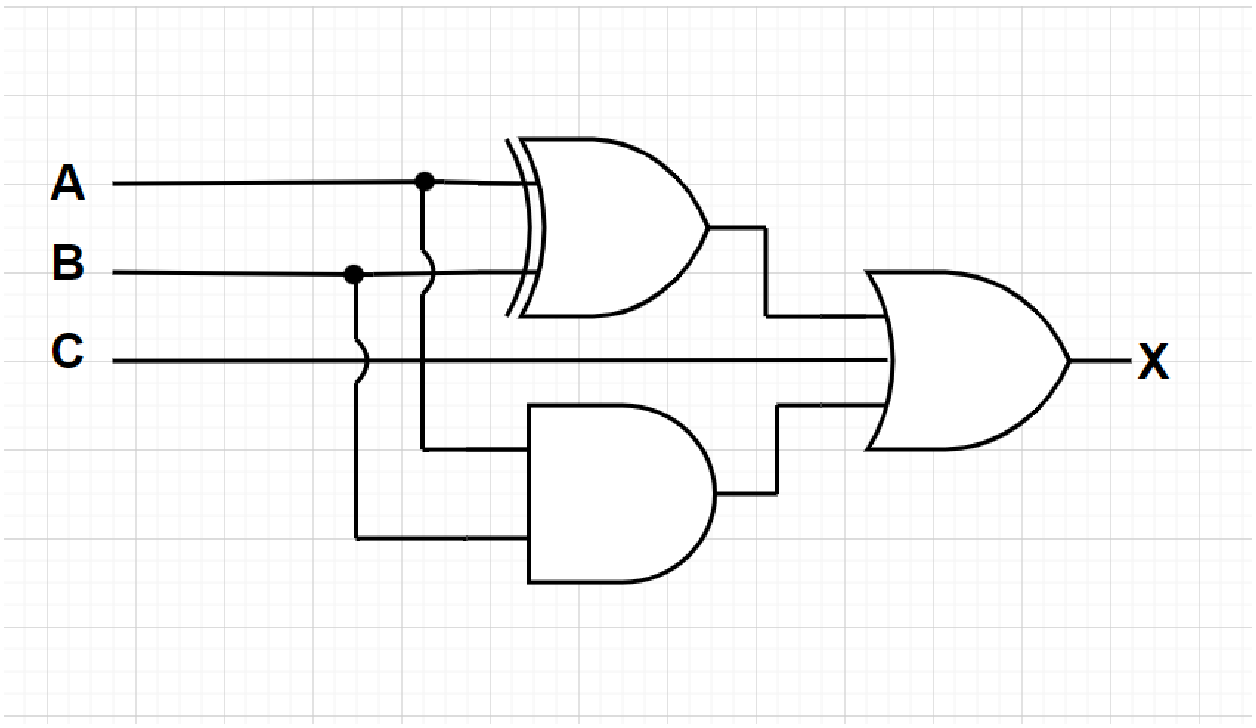
Task 01:

Implement combinational logic circuit using Verilog HDL.

Truth Table:

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Block Diagram:



Boolean Equation:

$X = A \oplus B + A . B + C$

Verilog Code:

design.v

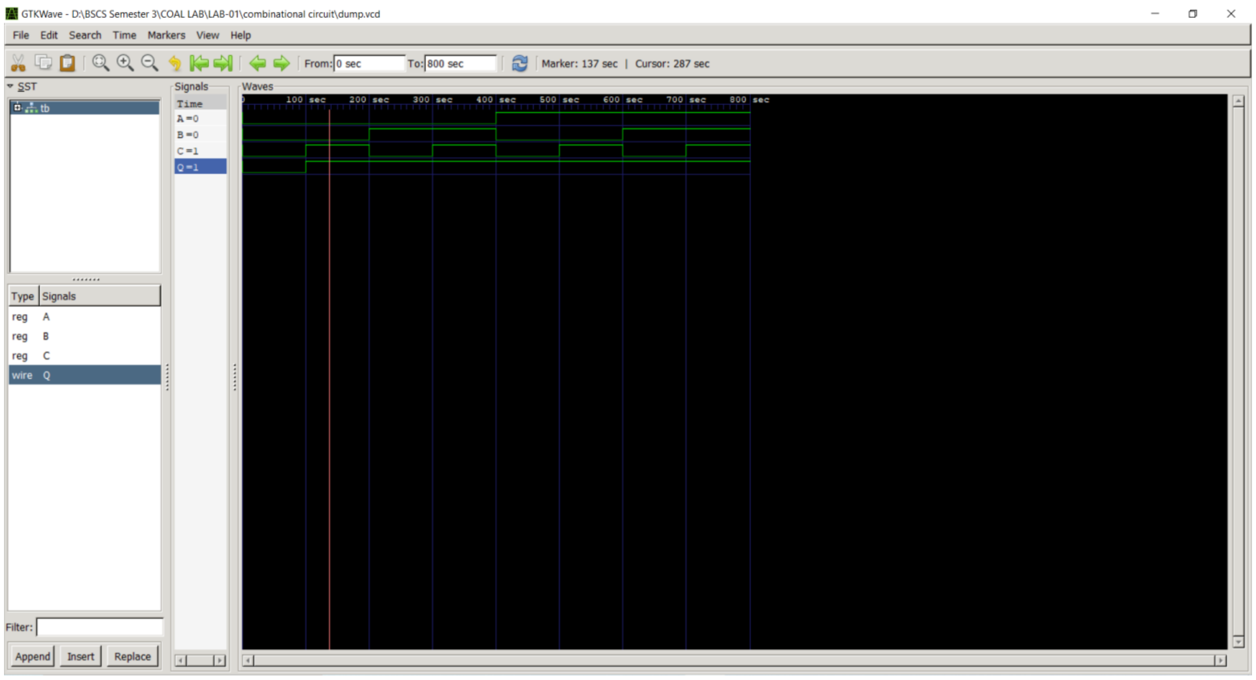
```
module combinational_circuit(A, B, C, Q);

    //inputs
    input A, B, C;
    output Q;

    //interim signals
    wire A_xor_B;
    wire A_and_B;

    //logic designing
    assign A_xor_B = A ^ B;
    assign A_and_B = A & B;
    // final output
    assign Q = C | A_xor_B | A_and_B;

endmodule

// assign Q = (A ^ B) | (A & B) | C
```

testbench.v

```
module tb();

    reg A, B, C;
    wire Q;

    // module declaration
    combinational_circuit dut (
        .A(A), .B(B), .C(C), .Q(Q)
    );

    initial begin
      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b0;
      #100;

      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b1;
      #100;

      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b0;
      #100;

      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b1;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b1;
      #100;
      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b1;
      #100;
    end

    initial begin
      $dumpfile("dump.vcd");
      $dumpvars(0);
    end
```

```
endmodule
```

Waveforms:


GTKWave - D:\BSCS Semester 3\COAL LAB\LAB-01\combinational circuit\dump.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 800 sec
Marker: 137 sec | Cursor: 287 sec
SST
tb
Type Signals
reg A
reg B
reg C
wire Q
Filter:
Append Insert Replace
Signals
Time
A=0
B=0
C=1
Q=1
Waves
0 100 sec 200 sec 300 sec 400 sec 500 sec 600 sec 700 sec 800 sec

----

Task 02:

Implement sequential logic circuit using Verilog HDL.

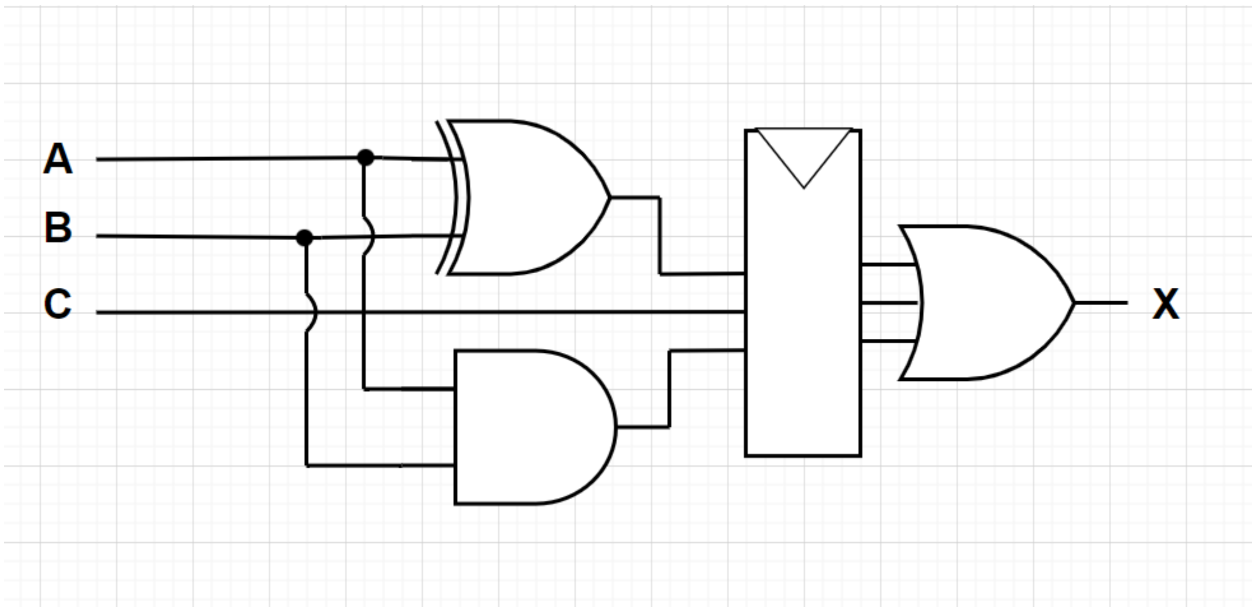
Literature Review:

Sequential Circuits have memory elements in addition to logic gates. They possess the ability to store the output for the computation of logic. Their outputs are a function of the inputs and the state of the storage elements. They depend upon current inputs as well as past inputs. They also include a time sequence of inputs and internal states. In this task, a combinational circuit is designed as a sequential circuit in which the output is stored in a flop just to show an example of sequential circuits.

Truth Table:

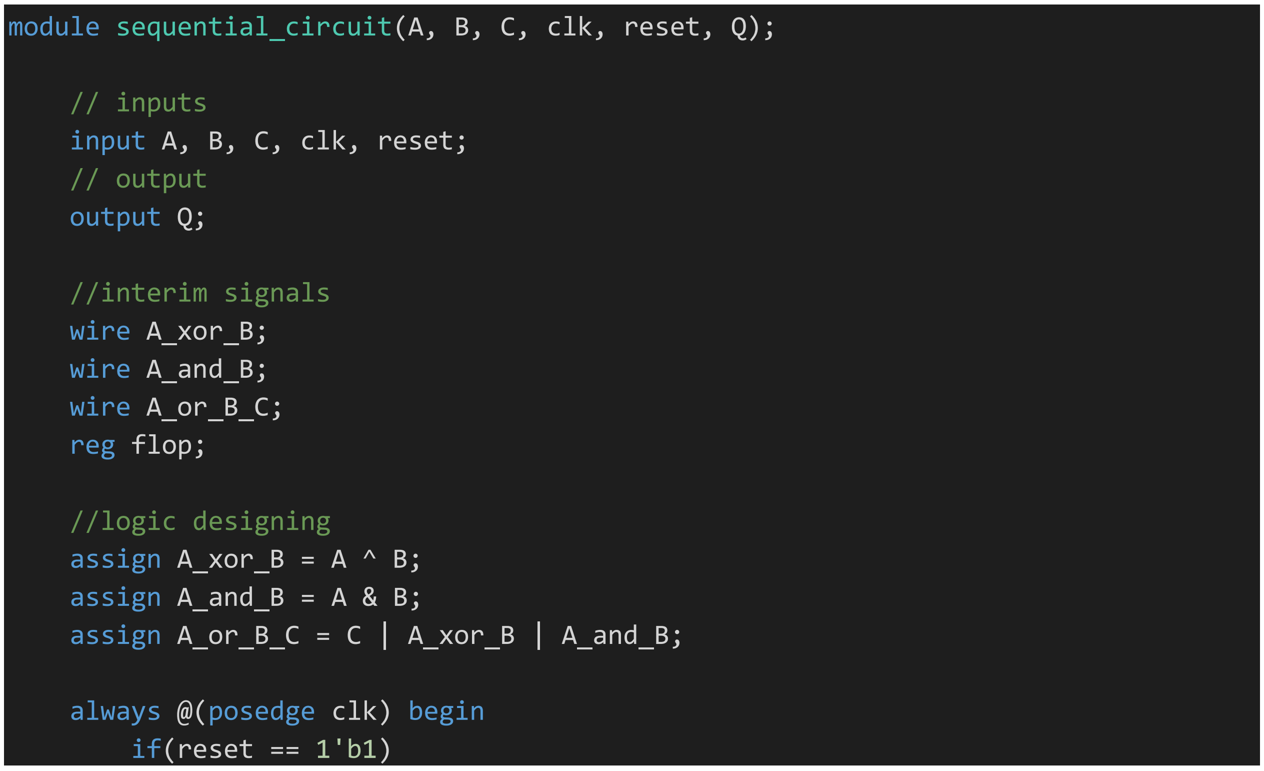
A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Block Diagram:



Verilog Code:

design.v

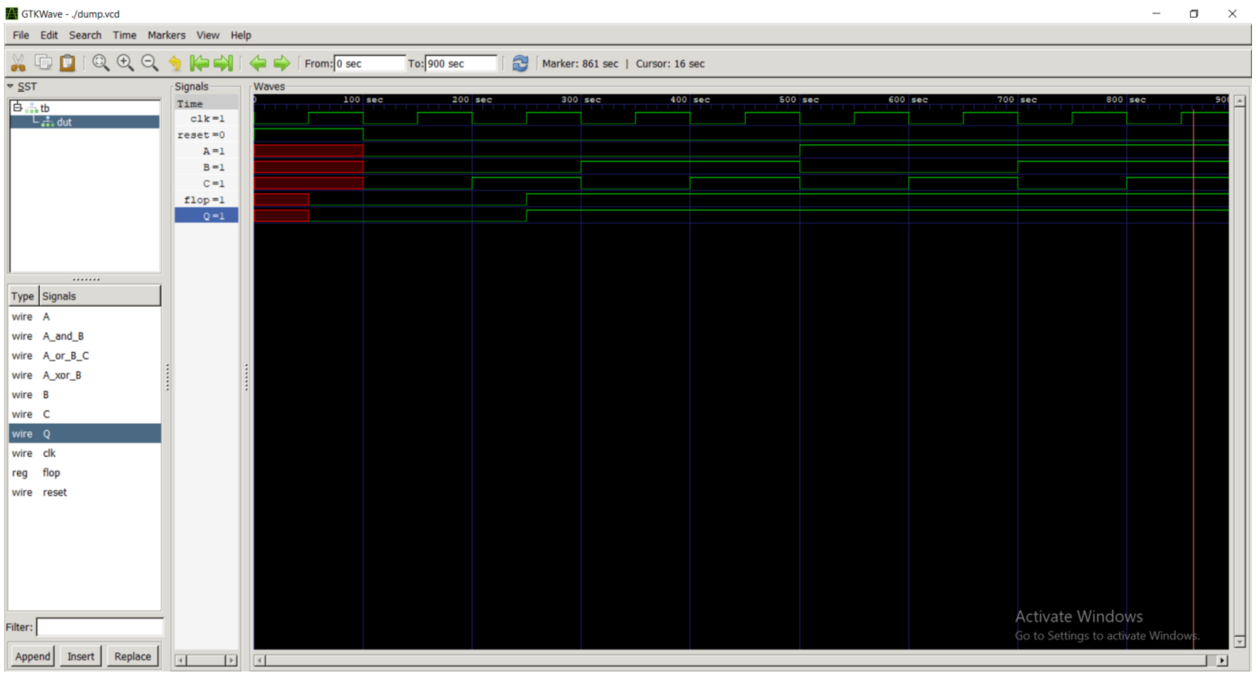


```
module sequential_circuit(A, B, C, clk, reset, Q);

    // inputs
    input A, B, C, clk, reset;
    // output
    output Q;

    //interim signals
    wire A_xor_B;
    wire A_and_B;
    wire A_or_B_C;
    reg flop;

    //logic designing
    assign A_xor_B = A ^ B;
    assign A_and_B = A & B;
    assign A_or_B_C = C | A_xor_B | A_and_B;

    always @(posedge clk) begin
        if(reset == 1'b1)
        begin
            flop <= 1'b0;
        end
        else begin
          flop <= A_or_B_C;
        end
    end

    assign Q = flop;

endmodule

// assign Q = (A ^ B) | (A & B) | C
```

testbench.v

```
module tb();

    reg A, B, C, clk, reset;
    wire Q;

    // module declaration
    sequential_circuit dut (
        .A(A), .B(B), .C(C), .clk(clk), .reset(reset), .Q(Q)
    );

    always begin
        clk <= 1'b0;
        #50;
        clk <= 1'b1;
        #50;
    end

    initial begin
      reset <= 1'b1;
      #100;
      reset <= 1'b0;

      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b0;
      #100;
      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b1;
      #100;

      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b0;
      #100;
      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b1;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b1;
      #100;
      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b1;
      #100;
      $finish;
    end

    initial begin
      $dumpfile("dump.vcd");
      $dumpvars(0);
    end

endmodule
```

Waveforms:



Type	Signals
wire	A
wire	A_and_B
wire	A_or_B_C
wire	A_xor_B
wire	B
wire	C
wire	Q
wire	clk
reg	flop
wire	reset

In Lab Tasks:

Task 03:

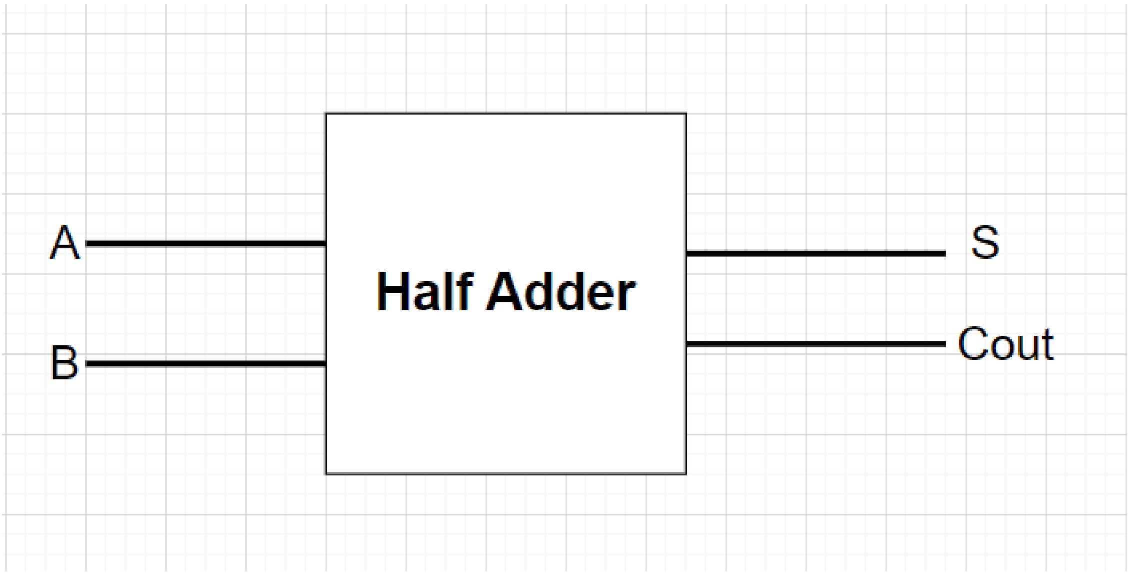
Create a Verilog module for half adder. Make a test bench for all the possible input combinations and attach the waveforms.

Half Adder

Literature Review:

Half adder is a combinational and arithmetic logic circuit which is used to add two binary digits. The adder accepts two inputs and produces tow outputs, one is sum and a carry out. Inputs are XORed to produce sum and then ANDed to produce carry out. If two half adders are cascaded, one full adder can be formed which can compute more than two bits addition.

Block Diagram:



Truth Table:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Boolean Equation:

$S = A \oplus B$

$C = A . B$

Verilog Code:

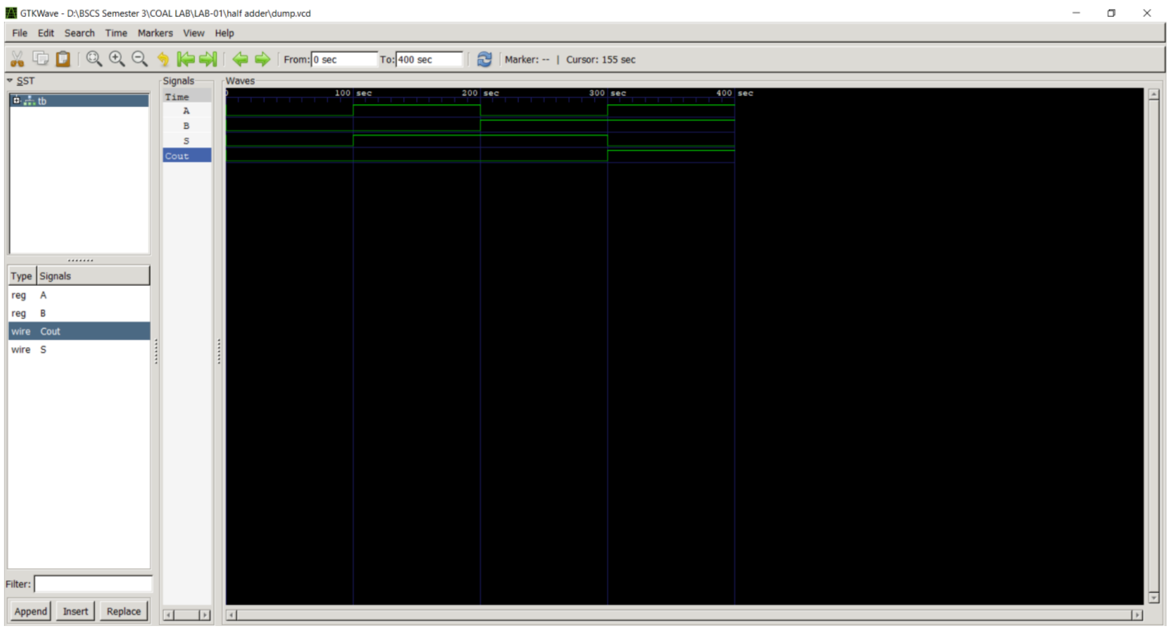
design.v

```
module half_adder(A, B, S, Cout);

    // inputs
    input A, B, C;
    // outputs
    output S, Cout;

    //logic designing
    assign S = A ^ B;
    assign Cout = A & B;

endmodule
```

testbench.v

```
module tb();

    reg A, B;
    wire S, Cout;
    // module declaration
    half_adder dut (
        .A(A), .B(B), .S(S), .Cout(Cout)
    );

    initial begin
      A <= 1'b0;
      B <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      #100;

      A <= 1'b0;
      B <= 1'b1;
      #100;

      A <= 1'b1;
      B <= 1'b1;
      #100;

    end

    initial begin
      $dumpfile("dump.vcd");
      $dumpvars(0);
    end

endmodule
```

Waveforms:


GTKWave - D:\BSCS Semester 3\COAL LAB\LAB-01\half adder\dump.vcd
File Edit Search Time Markers View Help
From: 0 sec To: 400 sec
Marker: -- | Cursor: 155 sec
SST
tb
Type Signals
reg A
reg B
wire Cout
wire S
Filter:
Append Insert Replace
Signals
Time
A
B
S
Cout
Waves
100 sec
200 sec
300 sec
400 sec

Task 04:

Create a two bit counter. Make a test bench for all the possible input combinations and attach the waveforms.

Sequential Circuit:

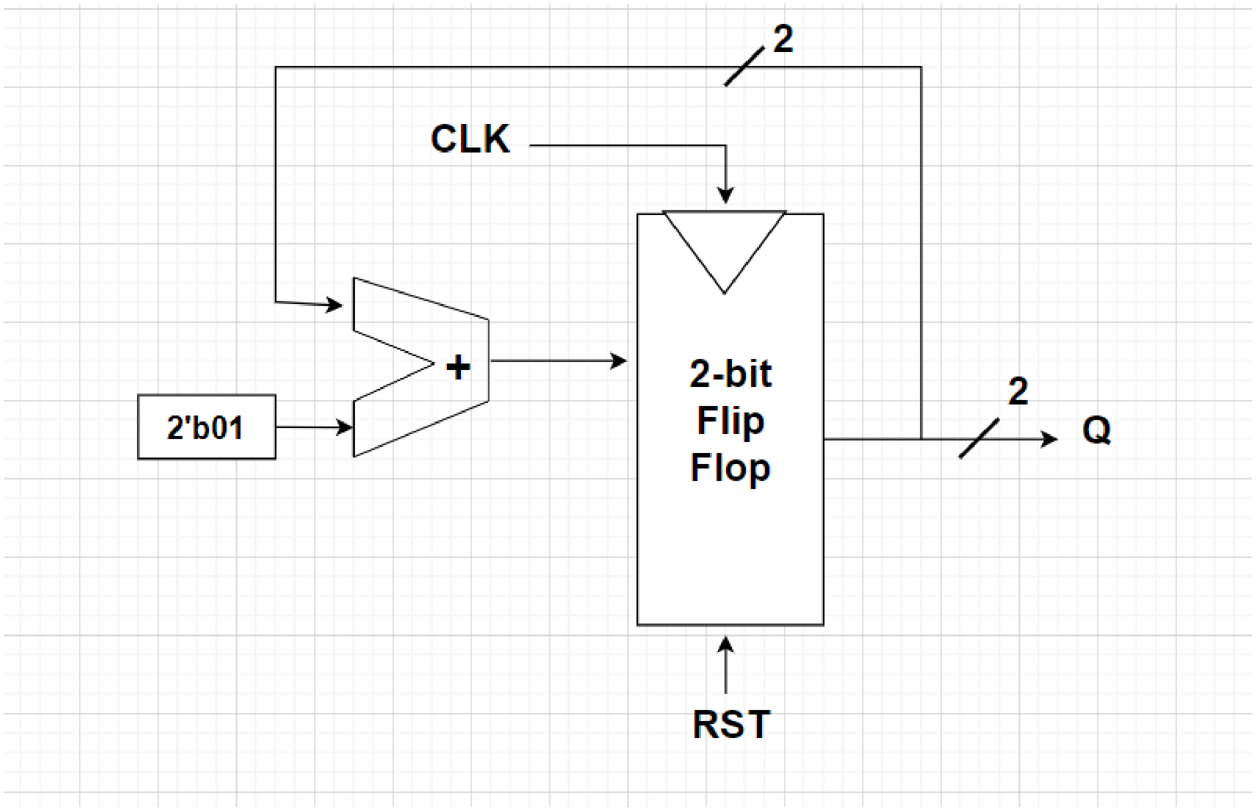
Literature Review:

Sequential circuits have certain applications, one of them is 2-bit counter. Counter simply increments one in the previous value and store it for the next computation. On every clock cycle, under defined conditions, one will add in the input and it will be stored in memory element and then for next cycle it will repeat the process until it reaches the limit of bits storage. And if the delay will still be there the cycle will restart from zero and keep on incrementing one.

Truth Table:

CLK	Q1	Q2
Initial	0	0
1	0	1
2	1	0
3	1	1
4	0	0

Block Diagram:



Verilog Code:

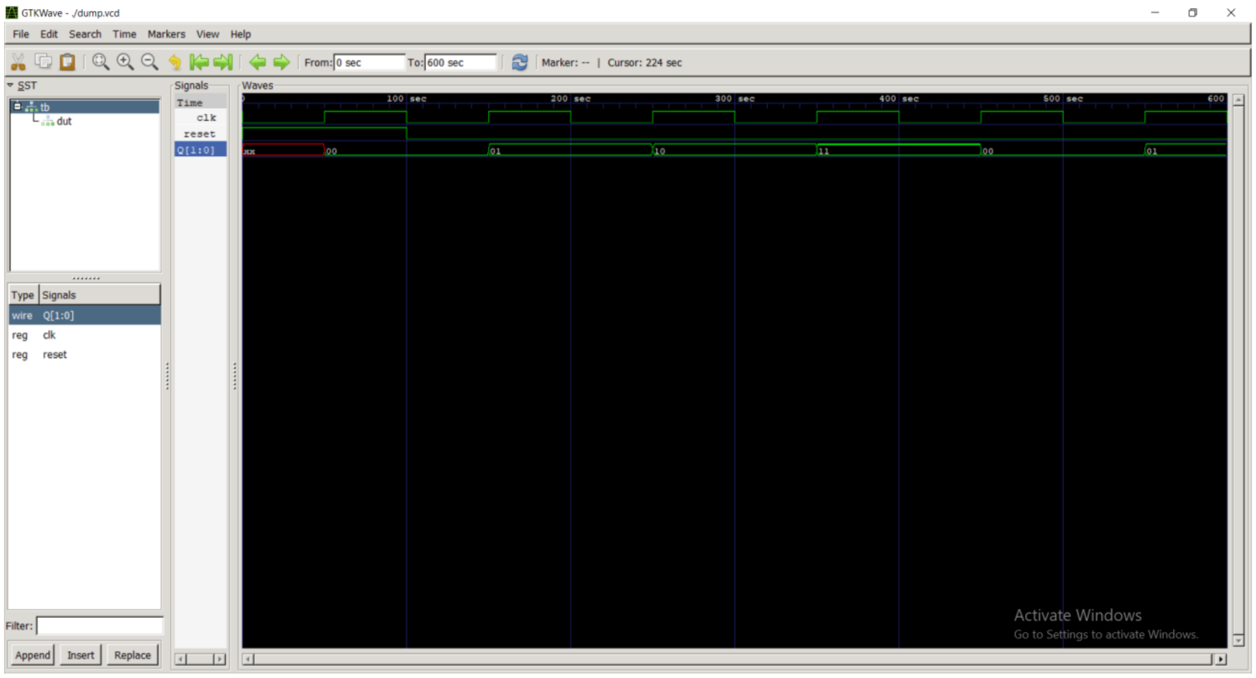
design.v

```
module two_bit_counter(input clk, reset,  output[1:0] Q);

    //input
    reg[1:0] flop;

    always @(posedge clk) begin
        if(reset == 1'b1)
        begin
            flop <= 2'b00;
        end
        else begin
          flop <= flop + 2'b01;
        end
    end
    assign Q = flop;

endmodule
```

testbench.v

```
module tb();

    reg clk, reset;
    wire[1:0] Q;

    // module declaration
    two_bit_counter dut (
        .clk(clk), .reset(reset), .Q(Q)
    );

    // clock cycle
    always begin
        clk <= 1'b0;
        #50;
        clk <= 1'b1;
        #50;
    end

    initial begin
      reset <= 1'b1;
      #100;
      reset <= 1'b0;
      #500;

      $finish;
    end

    initial begin
      $dumpfile("dump.vcd");
      $dumpvars(0);
    end

endmodule
```

Waveforms:


GTKWave - ./dump.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 600 sec
Marker: --
Cursor: 224 sec
SST
tb
dut
Signals
Time
clk
reset
Q[1:0]
Waves
100 sec
200 sec
300 sec
400 sec
500 sec
600
xx
00
01
10
11
00
01
Type Signals
wire Q[1:0]
reg clk
reg reset
Filter:
Append Insert Replace
Activate Windows
Go to Settings to activate Windows.

Post Lab Task:

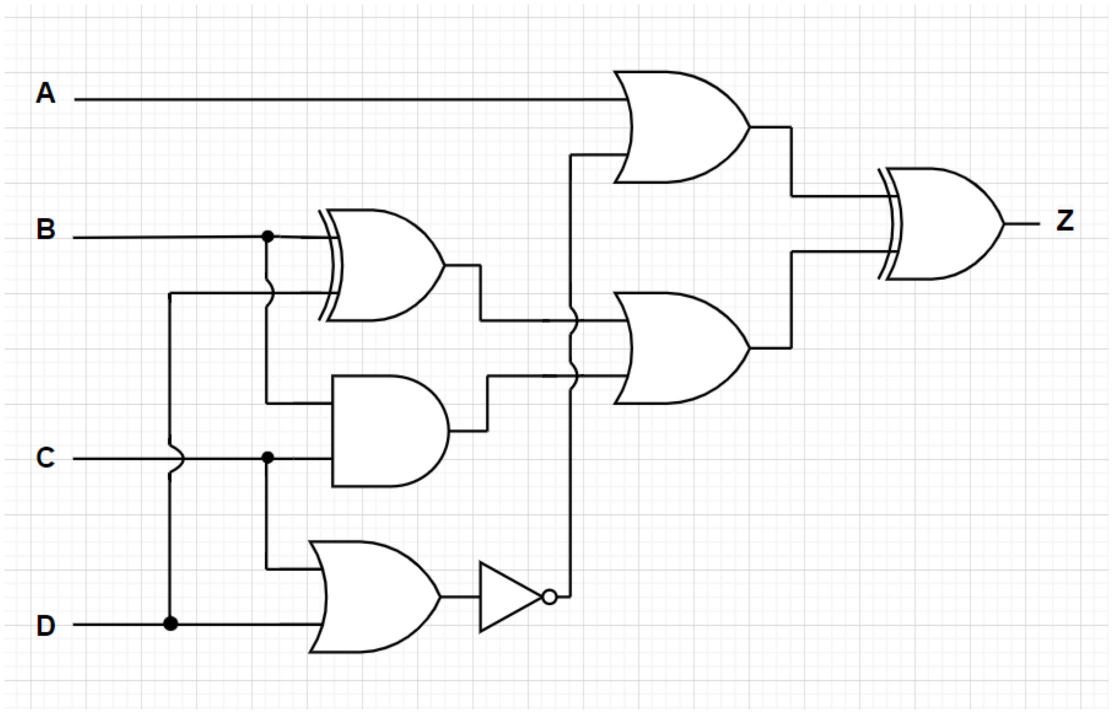
Task 05:

Below is the gate level circuit for the combinational circuit. Write down a data flow model for it. Hint: First write the Boolean expression for the below circuit.

Literature Review:

Combinational circuits are made up of basic logic gates such as and, or and not gates. A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs. There operations are determined by set of Boolean functions. Mainly, combinational circuits are used to design complex logics such as alarm system for stuck elevator or traffic lights etc. Here, we have four inputs, A,B,C and D. which are computed logically using AND, OR, NOT and XOR gate performing certain logic.

Block Diagram:



Truth Table:

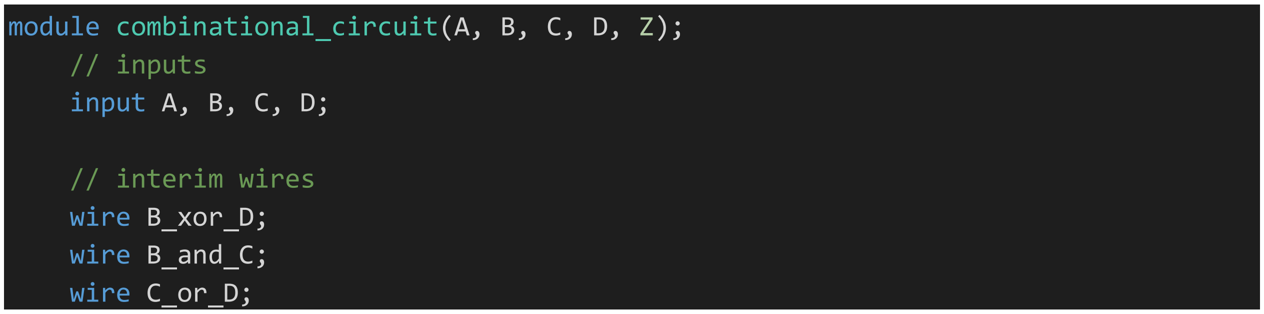
A	B	C	D	Z
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Boolean Equation:

$Z = A + \overline{(C + D)} \oplus (B \oplus D) + B.C$

Verilog Code:

design.v



```
module combinational_circuit(A, B, C, D, Z);
    // inputs
    input A, B, C, D;

    // interim wires
    wire B_xor_D;
    wire B_and_C;
    wire C_or_D;
```

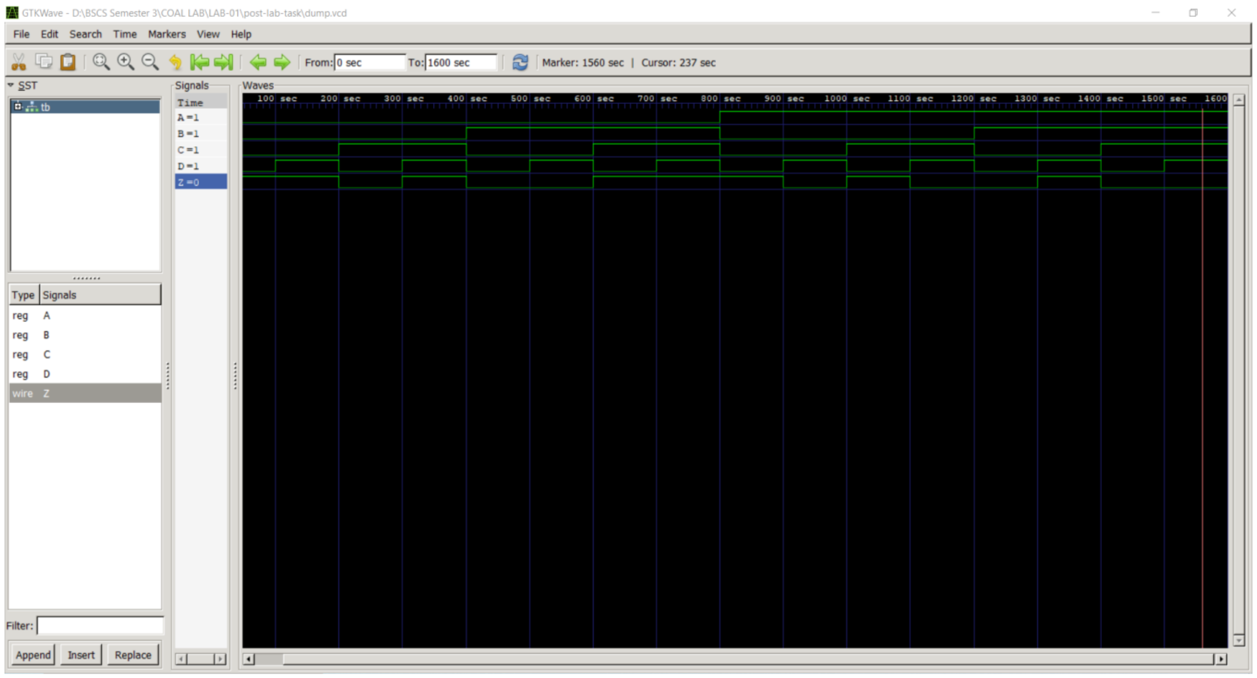
```verilog
    wire not_C_or_D;
    wire A_or_not_C_or_D;
    wire B_xor_D_or_B_and_C;

    // output
    output Z;

    // assignment with logical operations
    assign B_xor_D = B ^ D;
    assign B_and_C = B & C;
    assign C_or_D = C | D;
    assign not_C_or_D = !(C_or_D);
    assign A_or_not_C_or_D = A | not_C_or_D;
    assign B_xor_D_or_B_and_C = B_xor_D | B_and_C;
    // final output
    assign Z = A_or_not_C_or_D ^ B_xor_D_or_B_and_C;

endmodule

// assign Z = (A | (!(C | D))) ^ ((B ^ D) | (B & C));
```

testbench.v

```verilog
module tb();
    reg A, B, C, D;
    wire Z;

    // module declaration
    combinational_circuit dut (
        .A(A), .B(B), .C(C), .D(D), .Z(Z)
    );

    initial begin
      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b0;
      D <= 1'b0;
      #100;

      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b0;
      D <= 1'b1;
      #100;
      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b1;
      D <= 1'b0;
      #100;

      A <= 1'b0;
      B <= 1'b0;
      C <= 1'b1;
      D <= 1'b1;
      #100;

      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b0;
      D <= 1'b0;
      #100;

      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b0;
      D <= 1'b1;
      #100;

      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b1;
      D <= 1'b0;
      #100;
      A <= 1'b0;
      B <= 1'b1;
      C <= 1'b1;
      D <= 1'b1;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b0;
      D <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b0;
      D <= 1'b1;
      #100;

      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b1;
      D <= 1'b0;
      #100;
      A <= 1'b1;
      B <= 1'b0;
      C <= 1'b1;
      D <= 1'b1;
      #100;

      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b0;
      D <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b0;
      D <= 1'b1;
      #100;

      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b1;
      D <= 1'b0;
      #100;

      A <= 1'b1;
      B <= 1'b1;
      C <= 1'b1;
      D <= 1'b1;
      #100;

    end

    initial begin
      $dumpfile("dump.vcd");
      $dumpvars(0);
    end
```

```
endmodule
```

Waveform:


GTKWave - D:\BSCS Semester 3\COAL LAB\LAB-01\post-lab-task\dump.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 1600 sec
Marker: 1560 sec | Cursor: 237 sec
SST
tb
Type Signals
reg A
reg B
reg C
reg D
wire Z
Filter:
Append Insert Replace
Signals
Time
A=1
B=1
C=1
D=1
Z=0
Waves
100 sec 200 sec 300 sec 400 sec 500 sec 600 sec 700 sec 800 sec 900 sec 1000 sec 1100 sec 1200 sec 1300 sec 1400 sec 1500 sec 1600

* -- *