RTL CHALLENGE

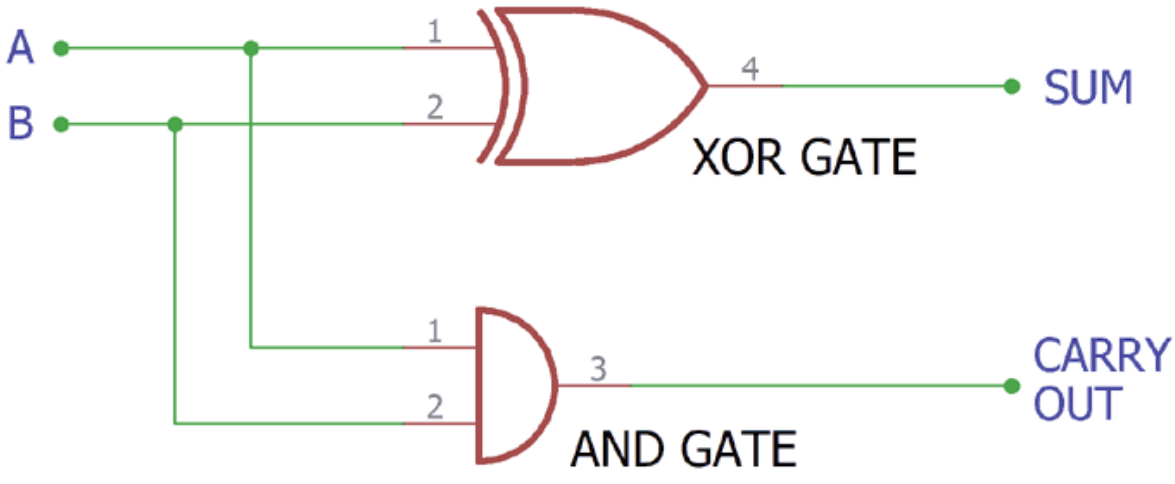
DAY – 1: Verilog Code for Implementation of Half Adder using GateLevel and Dataflow.

Software used: Xilinx Vivado (2023.1)

Theory:

A half-adder is a fundamental building block in digital circuit design, particularly in binary arithmetic circuits. It adds two single-bit binary numbers and produces the sum and carry-out as outputs. Here's a brief overview of how a half-adder works and its implementation:

Half Adder Circuit :



Functionality:

1. **Inputs**: A half-adder takes two inputs, typically labeled as A and B, representing the two single-bit binary numbers to be added.
2. **Outputs**: It produces two outputs:
   - The "sum" output (usually denoted as S) represents the least significant bit of the addition result.
   - The "carry" output (often denoted as C) indicates whether there is a carry-out from the addition operation.

Truth Table:

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

Implementation:

1. **Gate-Level Implementation**:
   - Using basic logic gates like AND, OR, and XOR gates.
   - The sum output (S) is obtained from the XOR gate of inputs A and B.
   - The carry output (C) is generated from the AND gate of inputs A and B.
2. **Dataflow Implementation**:

   Utilizing dataflow modeling, expressing the functionality of the half-adder in terms of data flow.
   It involves describing how the outputs are computed based on the inputs.
3. **Structural Implementation**:

   Constructing the half-adder by interconnecting lower-level components like basic gates.
   Allows for a hierarchical representation of the circuit, making it easier to understand and modify.

DESIGN CODES & TESTBENCH:

Code for HalfAdder using dataflow :

```
// Design Name:
// Module Name: halfadderusingdataflow


module halfadderusingdataflow(
    input a,b,
    output s,c
    );
    assign s=a^b;
    assign c=a&b;
endmodule
```

Testbench for HalfAdder using dataflow :

```
module halfadderusingdataflow_tb;
reg a;
reg b;
wire s;
wire c;
halfadderusingdataflow dut(.a(a),.b(b),.s(s),.c(c));
initial begin
a=0;b=0;#10;
a=0;b=1;#10;
a=1;b=0;#10;
a=1;b=1;#10;
end
endmodule
```

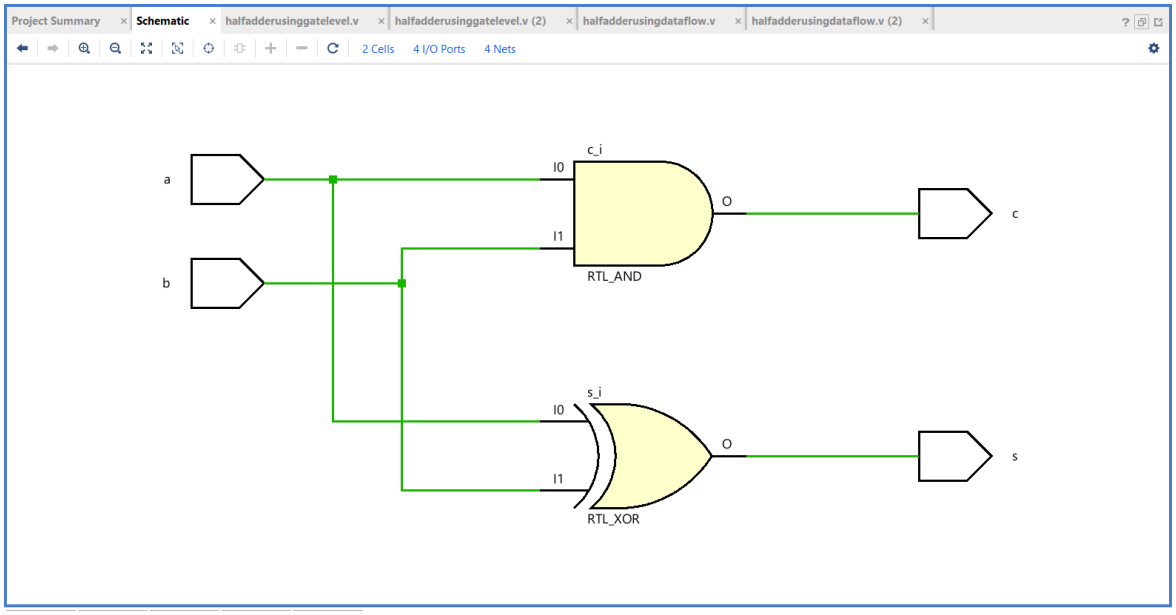
Code for HalfAdder using gatelevel :-

```
module halfadderusinggatelevel(
    input a,b,
    output s,c
    );
    xor (s,a,b);
    and (c,a,b);
endmodule
```

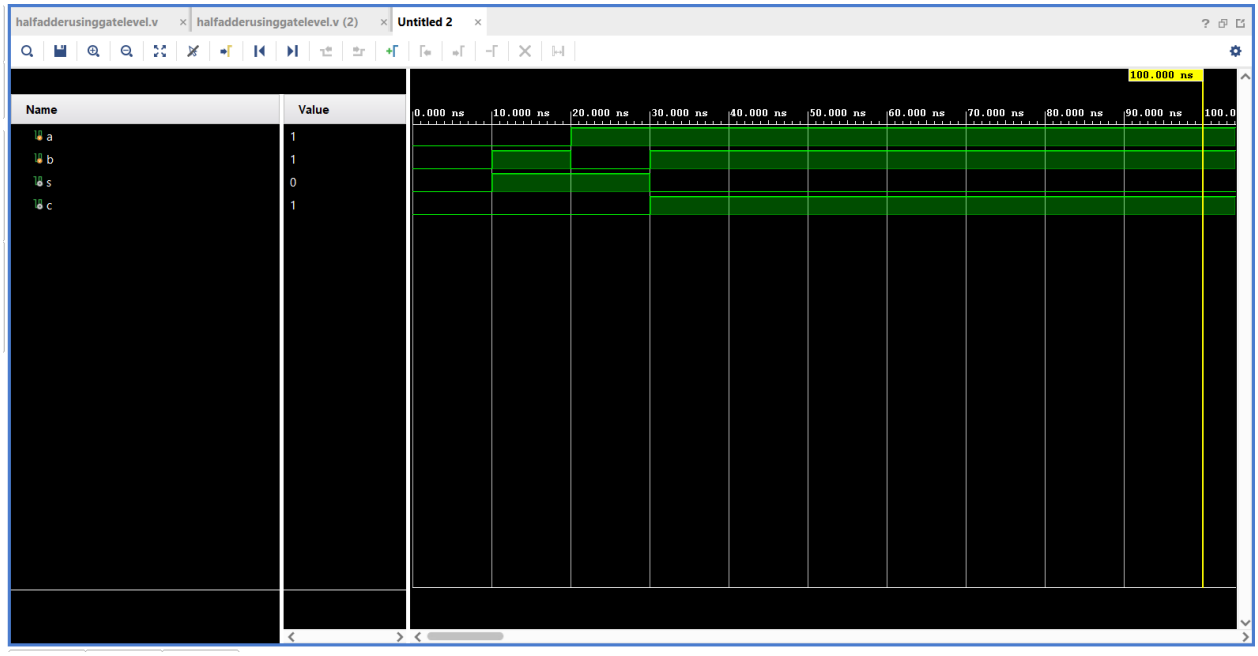
Testbench for HalfAdder using gatelevel :-

```
module halfadderusinggatelevel_tb;
reg a;
reg b;
wire s;
wire c;
halfadderusinggatelevel dut(.a(a),.b(b),.s(s),.c(c));
initial begin
a=0;b=0;#10;
a=0;b=1;#10;
a=1;b=0;#10;
a=1;b=1;#10;
end
endmodule
```

RTL SCHEMATIC:-



WAVEFORMS:


halfadderusinggatelevel.v
halfadderusinggatelevel.v (2)
Untitled 2
Name
Value
a
1
b
1
s
0
c
1
100.000 ns
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns

Code for HalfAdder using Structural :-

```
module halfadderusingstructural(
    input a,b,
    output s,c
    );
    xorgate x1 (.a(a),.b(b),.y(s));
    andgate a1 (.a(a),.b(a),.y(c));

endmodule
```

Code for XOR & AND Gates :-

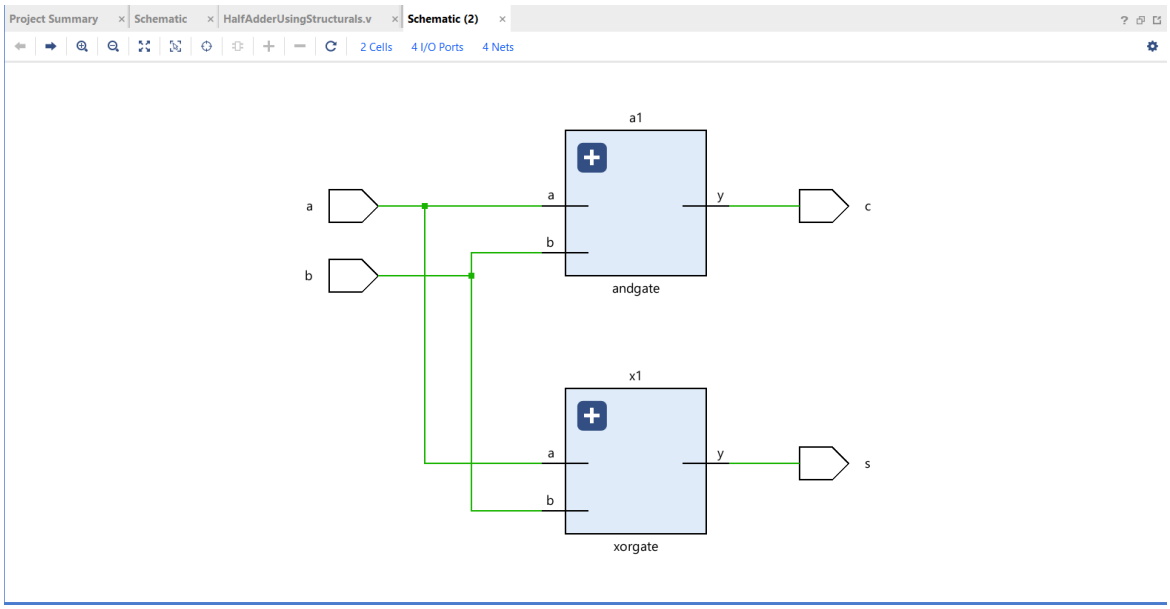
```
module xorgate(
    input a,b,
    output y
    );
    xor (y,a,b);
endmodule

module andgate(
    input a,b,
    output y
    );
    and (y,a,b);
endmodule
```

Testbench for HalfAdder using Structural :-

```
module halfadderusingstructural_tb;
reg a;
reg b;
wire s;
wire c;
halfadderusingstructural dut(.a(a),.b(b),.s(s),.c(c));
initial begin
a=0;b=0;#10;
a=0;b=1;#10;
a=1;b=0;#10;
a=1;b=1;#10;
end
endmodule
```

RTL SCHEMATIC :-



OUTPUT WAVEFORMS :-