Bialystok University of Technology

Faculty of Computer Science

EMBEDDED SYSTEMS FCS-00072

Report

Lab 1: Work scenario with Quartus Prime

Teacher: Adam Klimowicz

Student: Sevda Ghasemzadehnaghadehy

Task 1

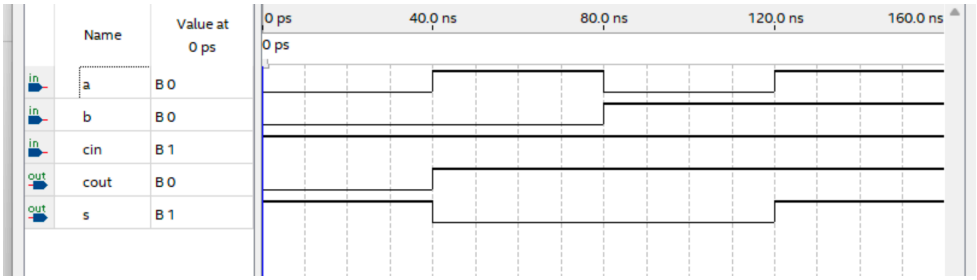
Objective:

Implement a one-bit adder project using assign operators.

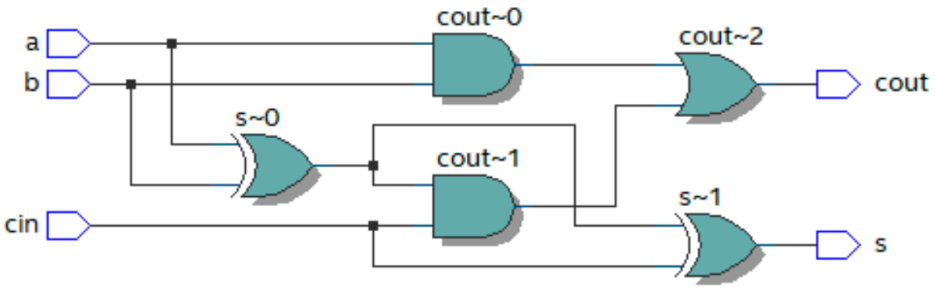
Verilog code:

```
module assigment1 (
    input a,
    input b,
    input cin,
    output s,
    output cout
);

assign s = a ^ b ^ cin;
assign cout = (a & b) | (cin & (a ^ b));

endmodule
```

Simulation:



Implementation (RTL level):



Conclusions:

In this task, I created a one-bit adder using assign statements in Verilog. I used basic logic operations like XOR, AND, and OR to describe how the sum and carry work. After running the simulation, I saw that the results matched the expected outputs.

Task 2

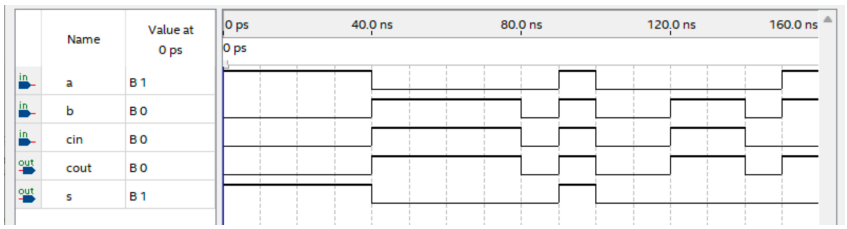
Objective:

Implement a one-bit adder design using Verilog gate primitives.

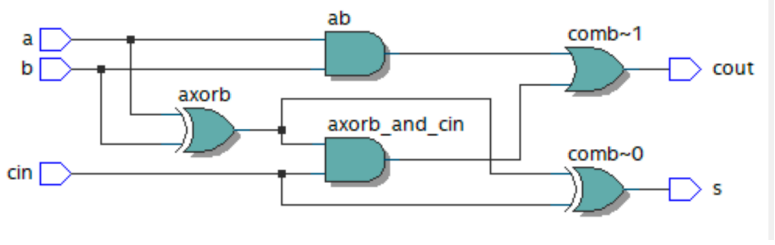
Verilog code:

```
module assigment1_gate (
    input a,
    input b,
    input cin,
    output s,
    output cout
);

    wire axorb;
    wire ab;
    wire axorb_and_cin;

    xor (axorb, a, b);
    xor (s, axorb, cin);

    and (ab, a, b);
    and (axorb_and_cin, axorb, cin);
    or  (cout, ab, axorb_and_cin);
endmodule
```

Simulation:



Implementation (RTL level):



Conclusions:

In this task, I implemented a one-bit full adder using basic gate primitives like XOR, AND and OR in Verilog. Instead of writing equations directly, I built the circuit step by step using logic gates. After simulating the design, I confirmed that the output matched the expected full adder behavior.

Task 3

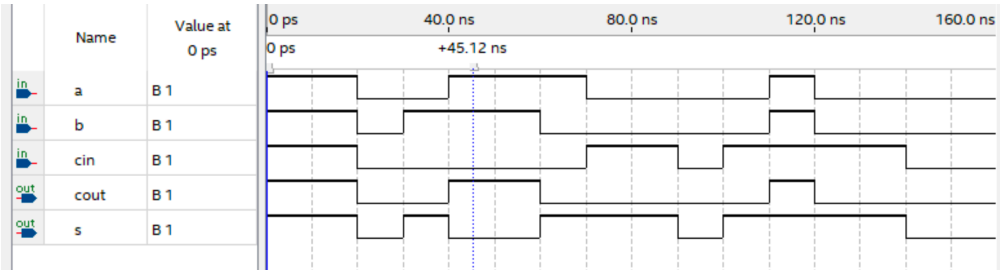
Objective:

Implement a one-bit adder using the arithmetic operation "+".

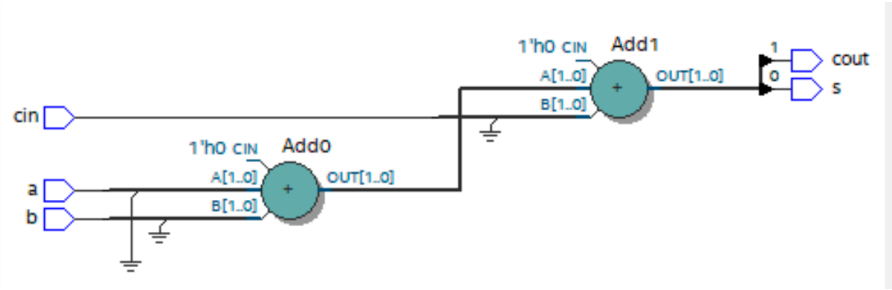
Verilog code:

```
module assigment1_plus (
    input a,
    input b,
    input cin,
    output s,
    output cout
);

assign {cout, s} = a + b + cin;

endmodule
```

Simulation:



Implementation (RTL level):



Conclusions:

In this task, I used the + operator to make a one-bit adder in Verilog. It was an easier way compared to using logic gates. Verilog automatically handled the addition, and the simulation showed correct results.

Task 4

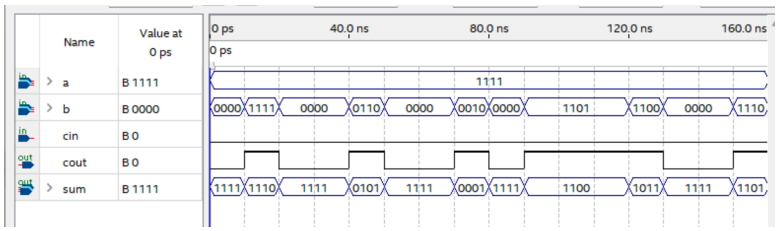
Objective:

Implement a 4-bit adder with ripple carry using four instances of one-bit adder.

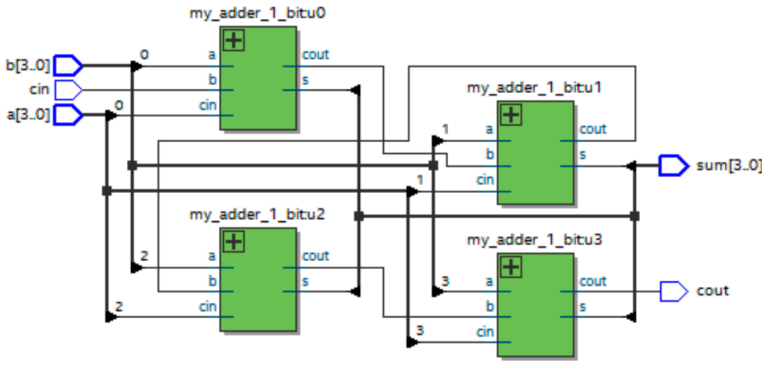
Verilog code:

```
assigment1_4_bit (
    input [3:0] a,
    input [3:0] b,
    input cin,
    output [3:0] sum,
    output cout
);

    wire c1, c2, c3;

    my_adder_1_bit u0 (a[0], b[0], cin,      sum[0], c1);
    my_adder_1_bit u1 (a[1], b[1], c1,       sum[1], c2);
    my_adder_1_bit u2 (a[2], b[2], c2,       sum[2], c3);
    my_adder_1_bit u3 (a[3], b[3], c3,       sum[3], cout);

endmodule
```

Simulation:



Implementation (RTL level):



Conclusions:

In this task, I used four instances of the one-bit adder I previously created to build a 4-bit ripple-carry adder. Each adder handled one bit, and the carry output from each was passed to the next. This helped me understand how larger digital systems are built from smaller modules. After compiling and simulating the design, the results were correct and matched expected values.