Digital Design Laboratory

Dr. Oniga István
University of Debrecen, Faculty of Informatics

This work was supported by the construction EFOP-3.4.3-16-2016-00021.
The project was supported by the European Union, co-financed by the European Social Fund.

5. Laboratory assignments

- Comparators
- Parity generators/checkers
- Adders

Lab5_1a assignment:

Comparators for 2-bits numbers

- Create a new HDL project (Lab5_1)
- Add a new Verilog source (Lab5_1.v)
- Edit the Lab5_1.v file according with description bellow.
- Add and adapt the constraints file Nexysx.UCF
- Generate the configuration file, test the circuit and fill the table.

```
module compare_2 (output reg A_lt_B, A_gt_B, A_eq_B,
                  input [1:0] A,B);

always @ (A or B)
        begin
            A_lt_B = 0;
            A_gt_B = 0;
            A_eq_B = 0;

            if (A==B) A_eq_B = 1;
            else if (A>B) A_gt_B = 1;
            else A_lt_B = 1;

        end

endmodule
```

A1	A0	B1	B0	A_lt_B	A_eq_B	A_gt_B
0	0	0	0			
0	0	0	1			
0	0	1	0			
1	1	1	1			

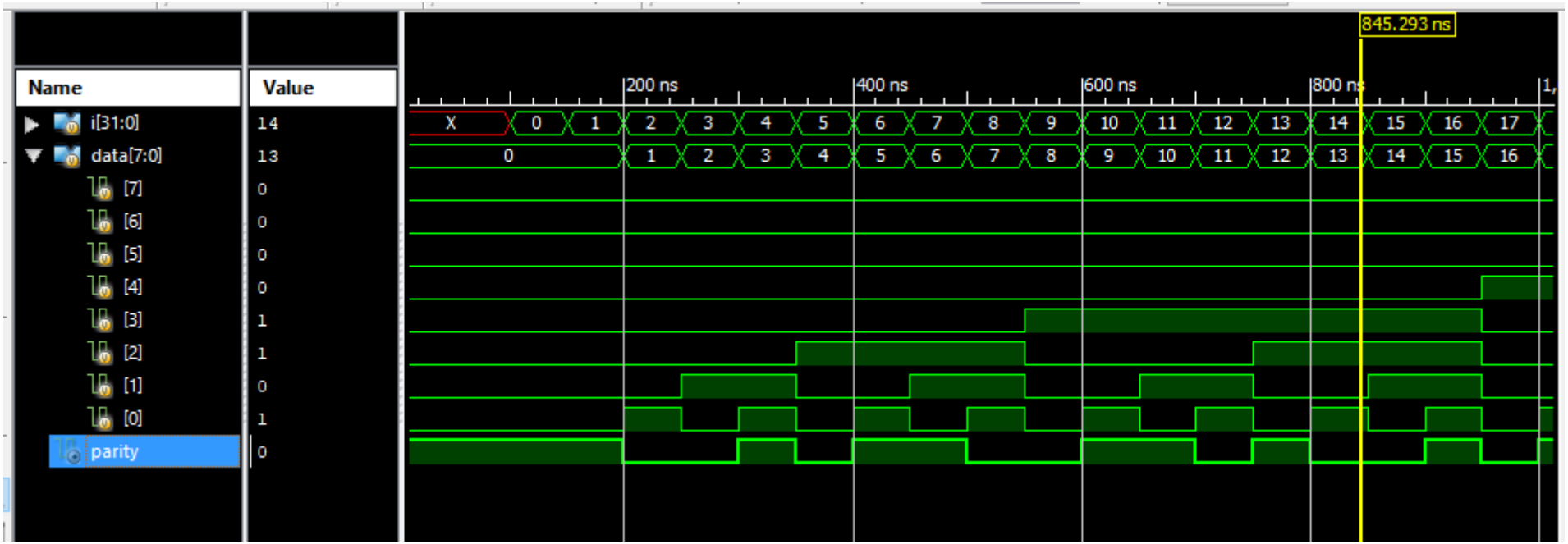
Lab5_2: Parity generator

- Create a new HDL project (Lab5_2)
- Add a new Verilog source (Lab5_2.v) and edit it according with description bellow
- Add a new Verilog text fixture file (Lab5_2_tf.v) and add the stimulus
- Simulate the circuit

```
module oddparity_for (output reg parity, input [7:0] data);
integer k;
always@(data)
begin
        parity = 1;
        for (k = 0; k <= 7; k = k+1)
        begin
                if (data[k] == 1)
                   parity = ~parity;
                end
        end
endmodule
```

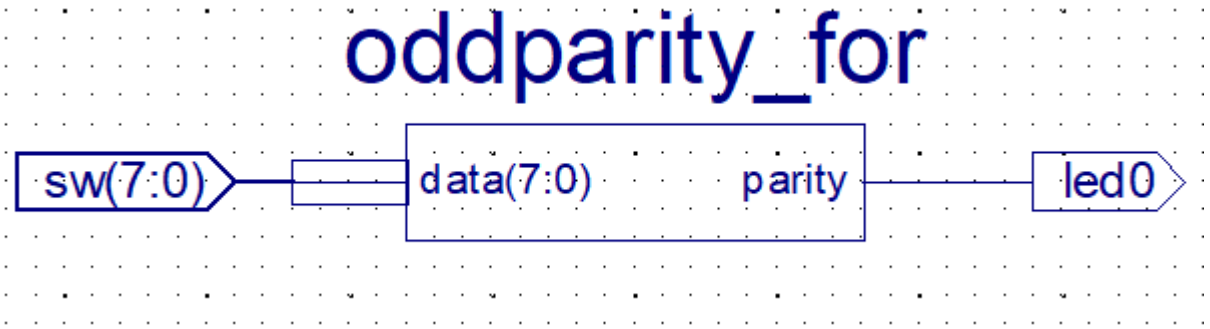
```
25  module oddparity_test;
26
27      // Inputs
28      reg [7:0] data;
29
30      // Outputs
31      wire parity;
32
33      // Instantiate the Unit Under Test (UUT)
34      oddparity_for uut (
35          .parity(parity),
36          .data(data)
37      );
38      integer i;
39      initial begin
40          // Initialize Inputs
41          data = 0;
42
43          // Wait 100 ns for global reset to finish
44          #100;
45
46          for (i = 0; i <= 255; i = i+1)
47          begin
48              #50 data =i;
49
50          end
51
52      end
53
54  endmodule
```

Lab5_2: Parity generator simulation

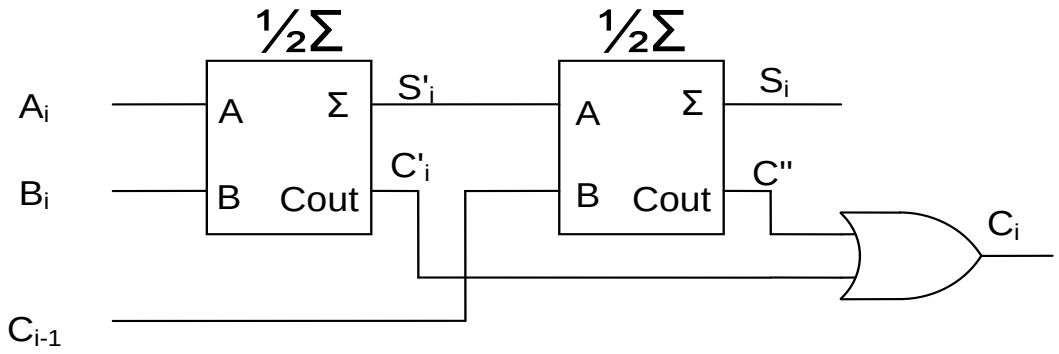


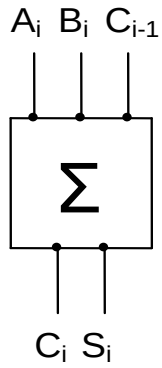
Lab5_2: Parity generator implementation

- Add and adapt the constraints file Nexysx.UCF
- Generate the configuration file and test the circuit.

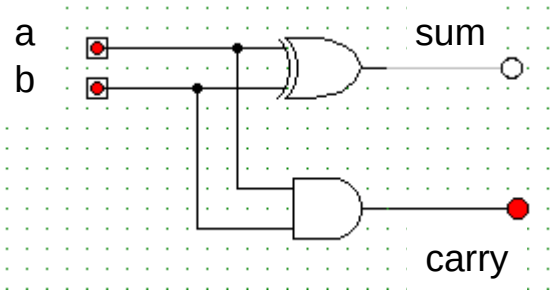


Lab5_3: 1-bit full adder





- Create a new HDL project (Lab5_3)
- Add a new Verilog source (half_add.v)

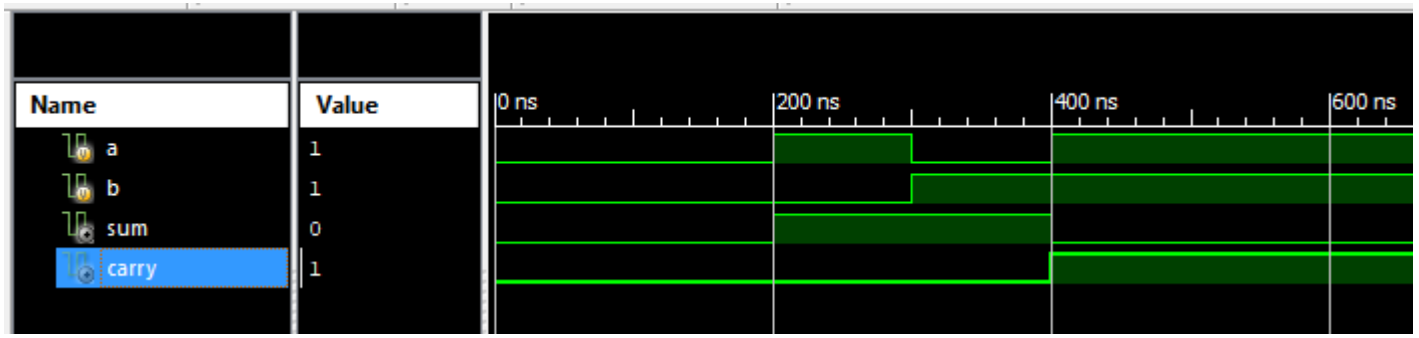



```
module half_add (output sum, carry, input a, b);
        xor (sum, a, b);     // exclusive OR
        and (carry, a ,b);   // AND
endmodule
```

Lab5_3a: 1-bit half adder simulation

- Add a new Verilog text fixture file (half_add_test.v)
- Specify the stimulus signals
- Simulate the circuit and verify the results

```
// Add stimulus here
#100
        a=1;
#100
        a=0;
        b=1;
#100
        a=1;
        b=1;
```



Lab5_3b: 1-bit full adder

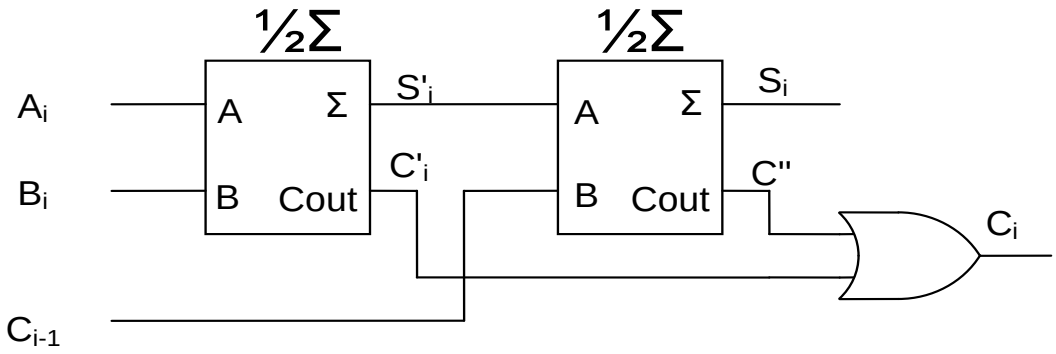
Structural description of 1-bit full adder

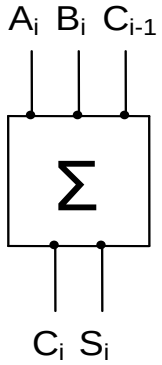
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$$

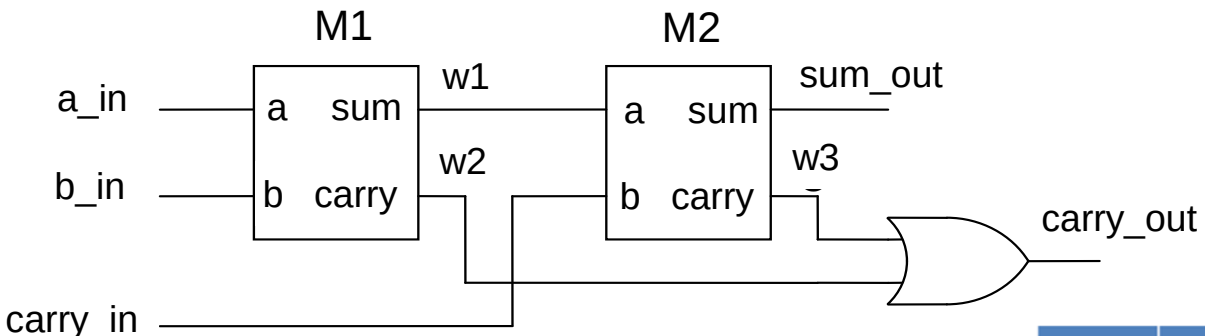
Full adder built from two half_adders:





Lab5_3b: 1-bit full adder

- Add a new Verilog file (full_add.v)
- Specify the full adder model as in the next figure




```
module full_add (output sum_out, carry_out, input a_in, b_in, carry_in );

wire w1, w2, w3;

        half_add M1 (.a(a_in), .sum(w1), .b(b_in), .carry(w2));
        half_add M2 (.sum(sum_out), .b(w1), .carry(w3), .a(carry_in));

        or (carry_out, w2, w3);
endmodule
```

Cin	A	B	Sum	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

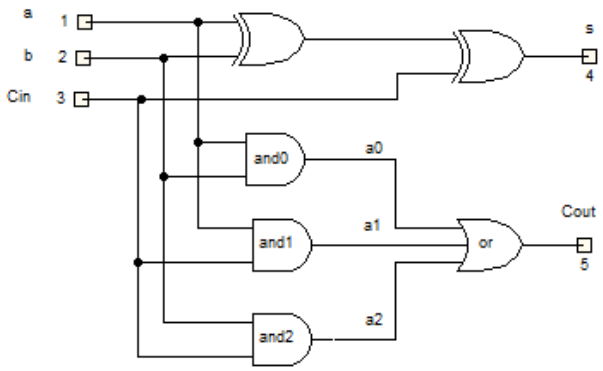
- Add and adapt the constraints file Nexysx.UCF
- Generate the configuration file and test the circuit.

Lab5_4: 1-bit full adder

```
// A version
module add1_full (input a, b, cin, output cout, s);
xor3_m xor(.i0(a), .i1(b), .i2(cin), .o(s));
wire a0, a1, a2;
and2_m and0(.i0(a), .i1(b), .o(a0));
and2_m and1(.i0(a), .i1(cin), .o(a1));
and2_m and2(.i0(b), .i1(cin), .o(a2));
or3_m or(.i0(a0), .i1(a1), .i2(a2) , .o(cout))
endmodule
```

```
// B version
module add1_full (input a, b, cin, output cout, s);
assign s = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);
endmodule
```

```
// C version
module add1_full (input a, b, cin, output cout, s);
assign {cout, s} = a + b + cin;
endmodule
```



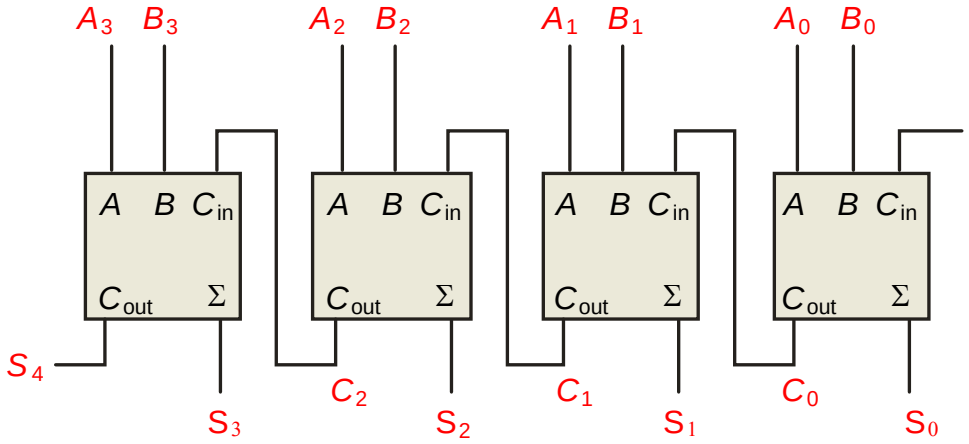
Cin	A	B	Sum	Cout
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

- Create a new HDL project (Lab5_4)
- Add a new Verilog source (add1_full.v). Use one of the tree source from above
- Add and adapt the constraints file Nexysx.UCF
- Generate the configuration file and test the circuit.

Lab5_5a: 4-bits adder

A 4-bit adder can be built using four 1-bit full adders

Ripple carry adder



- Create a new HDL project (Lab5_5a)
- Add a new Verilog source file
- Use the structural description to create a 4-bit adder (next slide)

Lab5_5a: 4-bits adder

- Create a new HDL project (Lab5_5a)
- Create a new Verilog source file add4.v with the content as bellow
- Add a copy of the source created in the previous project (add1_full.v).
- Add and adapt the constraints file Nexysx.UCF
- Generate the configuration file and test the circuit.

```
// 5_5a 4-bits adder
module add4 (input [3:0] a, b, output [4:0] s);
wire [3:0] c;
add1_full add0(.a(a[0]), .b(b[0]), .cin(1'b0),    .cout(c[0]), .s(s[0]));
add1_full add1(.a(a[1]), .b(b[1]), .cin(c[0]), .cout(c[1]), .s(s[1]));
add1_full add2(.a(a[2]), .b(b[2]), .cin(c[1]), .cout(c[2]), .s(s[2]));
add1_full add3(.a(a[3]), .b(b[3]), .cin(c[2]), .cout(s[4]),    .s(s[3]));
endmodule
```

Lab5_5b: 4-bits adder behavioral

- Create a new HDL project (Lab5_5b)
- Create a new Verilog source file add4.v with the content as bellow
- Add a copy of the source created in the previous project (add1_full.v).

```
// 5_5b
module add4 (input [3:0] a, b, output [4:0] s);
assign s = a + b;
endmodule
```

- Add and adapt the constraints file Nexysx.UCF
- Generate the configuration file and test the circuit.