Group 54

Eva Suska 260618967
Marcel Morin 260605670

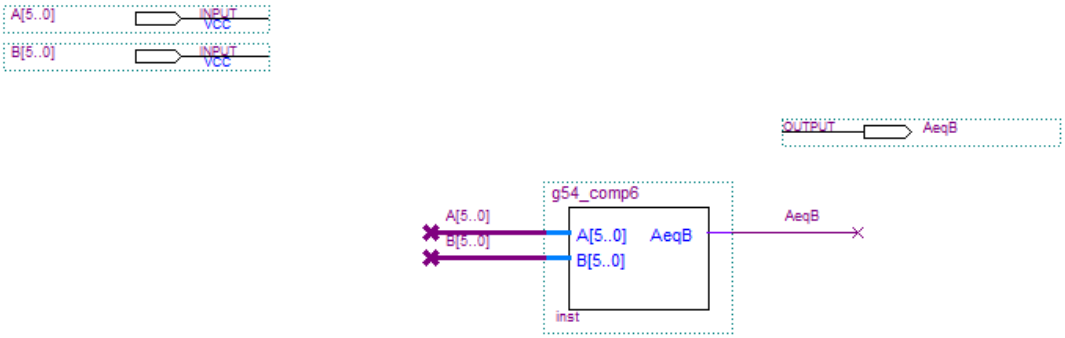
g54_comp6 - Lab 1

Description of 6 bit Comparator

Inputs: 2 6 bit inputs, A[0..5], B[0..5]
Output: 1 bit output, AeqB

This circuit takes 2 6 bit input compares the values and returns 1 if they are the same and 0 if they are not.
It does this by preforming an exclusive or with each of the input bits with each other and then anding them together.

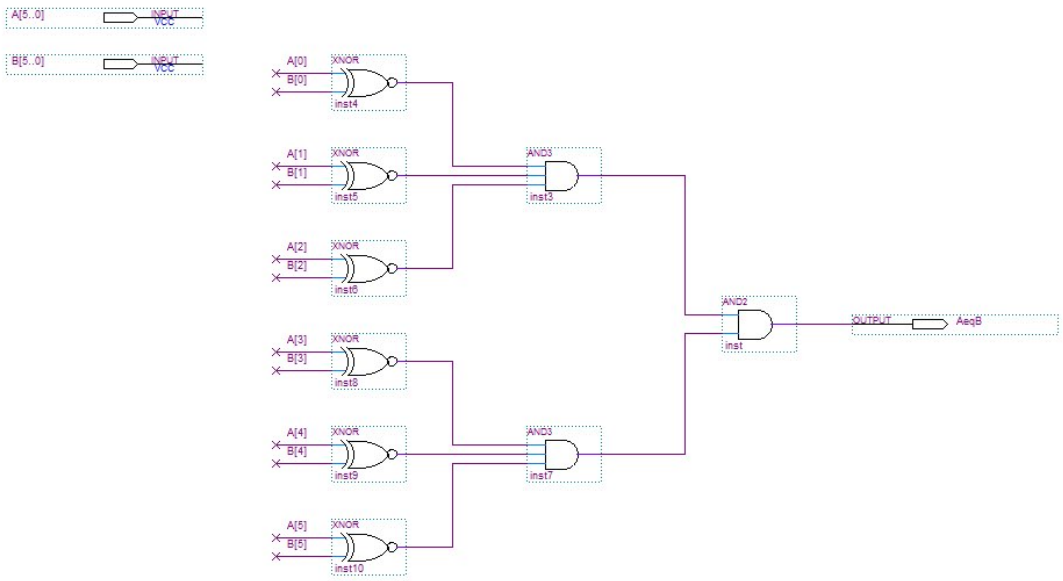
6 bit Comparator block component: g54_comp6Block



Eva Suska 260618967
Marcel Morin 260605670

Gate Level Schematic Diagram of 6 bit Comparator

6 bit Comparator: g54_comp6

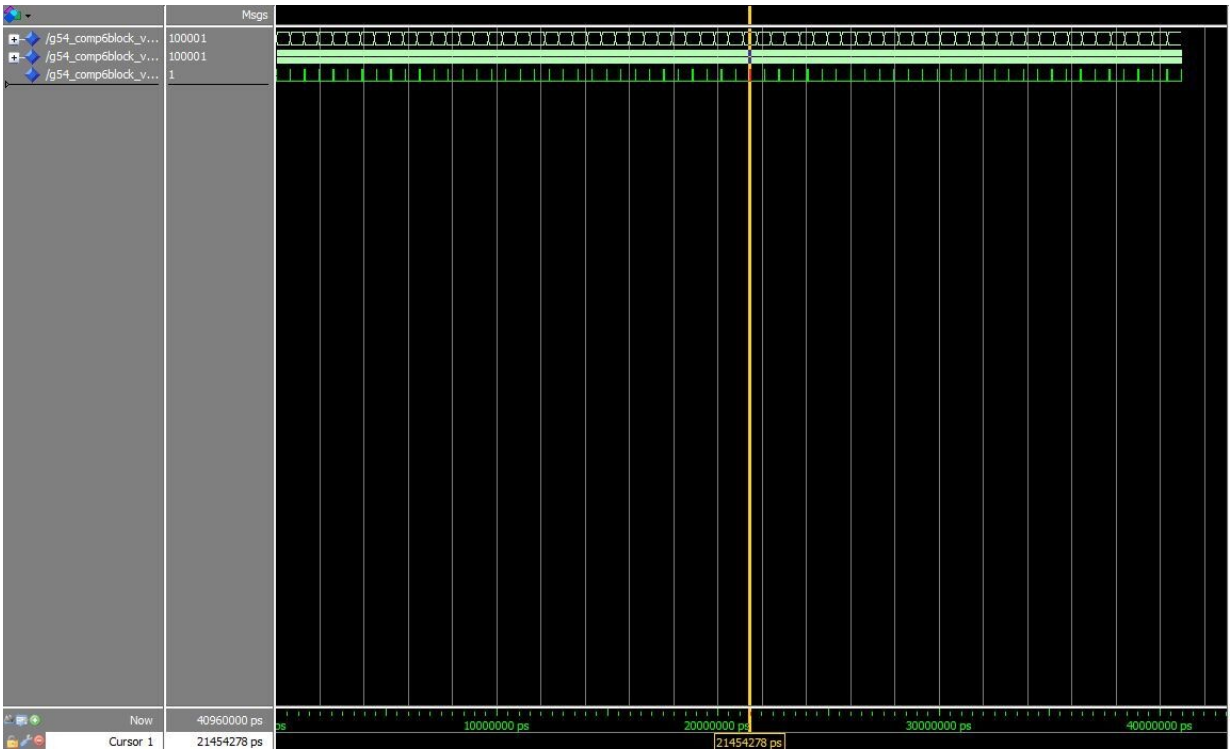


Testing

To test this circuit we first created a TestBench with different inputs that will be applied to our circuit to simulate it on the ModelSim software. We started with automatically generating a TestBench using Quartus Test Bench Writer tool to create a VHDL template, then modified it to test some values. We began with assigning the inputs different values and then waiting 10 ns in between to see if the results were correct. We tried making the A input “000000” and B “000000” and saw that the output was 1 which was correct, and then switching A to “101100” and saw that it returned 0, and then making B also “101100” returning 1, and making A “111101" then B “111101”.

In order to check that the circuit works completely correctly we must check all possible input value combinations. We did this by looping over each value of the input A with each value of B and waiting 10 ns in between changing the values to allow the signals to settle. The wave simulation result were as follows:

Eva Suska 260618967
Marcel Morin 260605670



We scanned through the results and checked if they were correct. From the wave diagram we could see where the output resulted in a 1, so we especially paid attention that the inputs were the same for all the output values of 1.

g54_6bitadder

Description of 6 bit Adder

Inputs: 2 6 bit inputs: A[0..5], B[0..5], and a 1 bit carry in: c_in
Output: 6 bit output: sum[5..0], 1 bit carry out: c_out

The 6 bit adder takes 2 6 bit inputs and a carry in bit. It takes each bit from the 2 inputs and feeds each of the pairs into a full adder to output a 6 bit sum of the inputs and a carry out.

The smallest component of the 6 bit adder is the half adder component. It takes 2 bit inputs, A[0] and B[0] preforming an exclusive OR to output the sum and an AND for the carry output.

The full adder circuit uses 2 half adder components. This consisted of taking 2 1 bit inputs and feeding them through a half adder and then the resulting sum output and the carry in input to second half adder to get the final output sum. Along with the output carries from the 2 half adders and ORed them for the output carry.

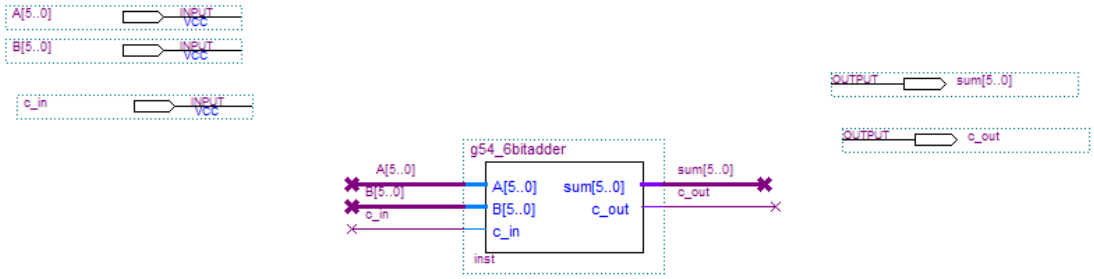
To create a full 6 bit adder we connected 6 full adder components. Fed in each bit from the inputs into each full adder and connected the carry out to the carry in of the next full adder/

We then made a component of the 6 bit adder to be able to easily implement it down the road in any other design.

Group 54

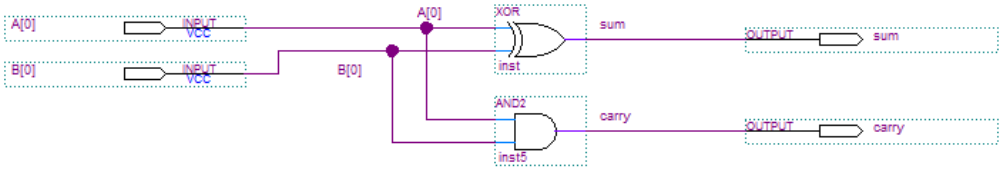
Eva Suska 260618967
Marcel Morin 260605670

6 bit adder Component: g54_6bitadderBLOCK

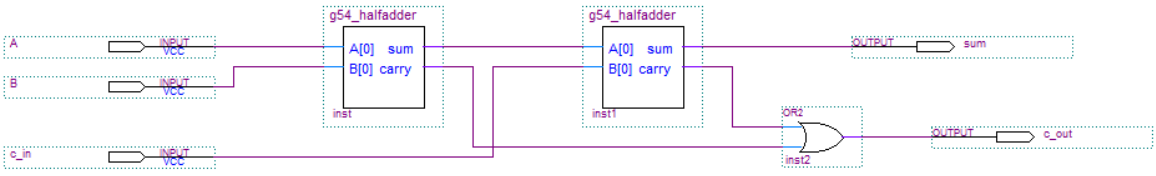


Gate Level Schematic Diagram of 6 bit Adder

Half Adder: g54_halfadder

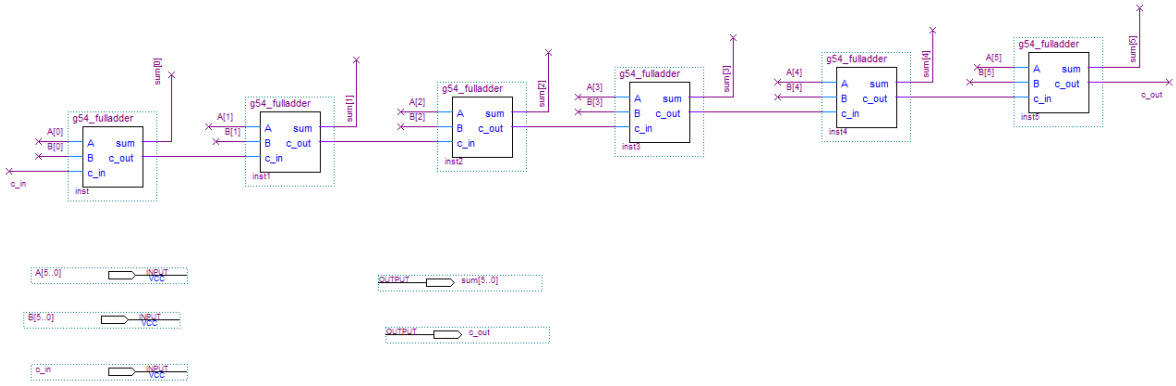


Full Adder: g54_fulladder



Eva Suska 260618967
Marcel Morin 260605670

6 bit Adder: g54_6bitadder



Testing

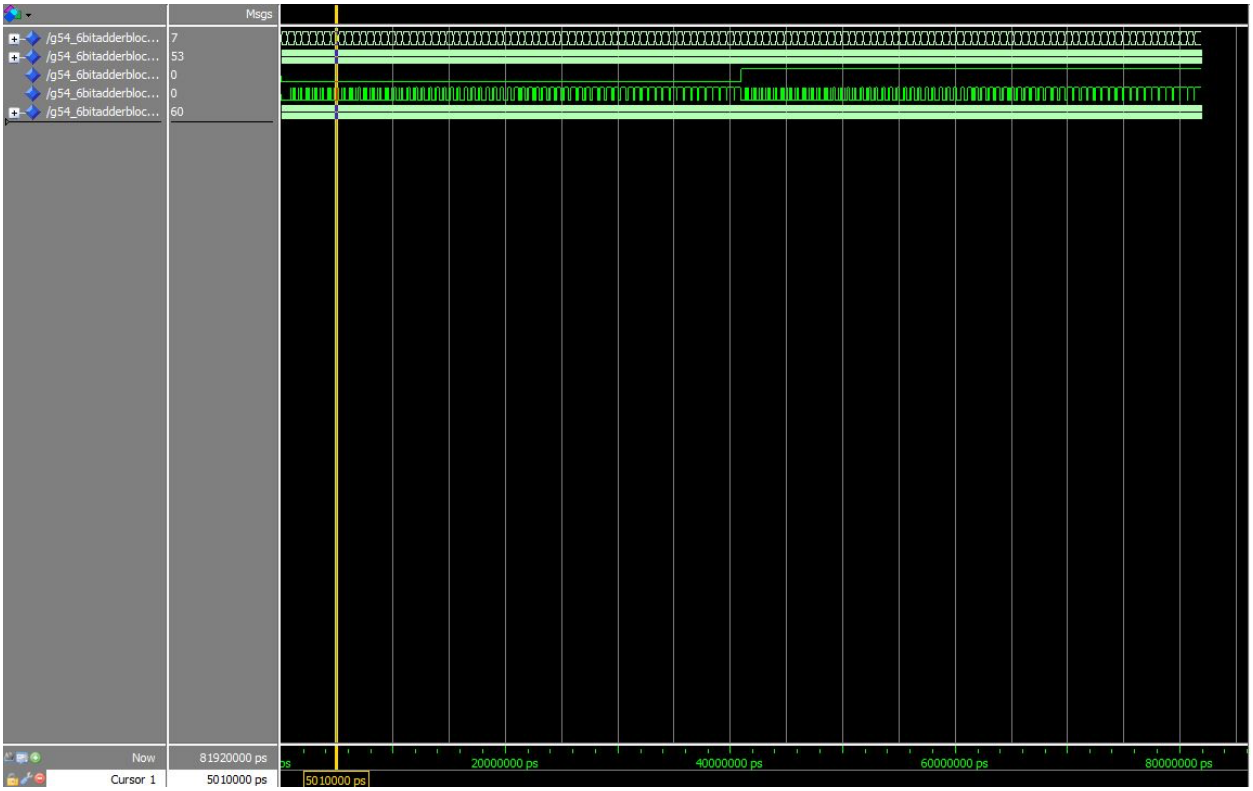
To test this circuit we created a TestBench with different inputs that will be applied to our circuit to simulate it on the ModelSim software. In order to check that the circuit works completely correctly we must check all possible input value combinations. We did this by nesting loops over each value of the carry in with input A and B and waiting 10 ns in between changing the values to allow the signals to settle.
Here is a snippet of our TestBench with our nested loops:

```
BEGIN
     FOR k in std_logic range '0' to '1' LOOP
     c_in <= k;
       FOR i IN 0 to 63 LOOP
         A <= std_logic_vector(to_unsigned(i,6));
         FOR j In 0 to 63 LOOP
           B<= std_logic_vector(to_unsigned(j,6));
           WAIT FOR 10 ns;

        -- code executes for every event on sensitivity list
     END LOOP;
   END LOOP;
  END LOOP;
WAIT;
END PROCESS always;
```

The wave simulation result were as follows:

Eva Suska 260618967
Marcel Morin 260605670



We went through the signals to check that the results were correct manually adding the inputs to see if the sum and carryout were correct.
To be sure, we also took a look at a table of the signals. The edge cases worked and therefore we assume that all values in the middle work.
Snippet of list:

ps delta	A B c_in c_out sum
0 +0	UUUUUU UUUUUU U U UUUUUU
0 +1	000000 000000 0 U UUUUUU
0 +4	000000 000000 0 0 UUUUU0
0 +5	000000 000000 0 0 UUUU00
0 +6	000000 000000 0 0 000000
10000 +1	000000 000001 0 0 000000
10000 +4	000000 000001 0 0 000001
20000 +1	000000 000010 0 0 000001
20000 +4	000000 000010 0 0 000010
30000 +1	000000 000011 0 0 000010
30000 +4	000000 000011 0 0 000011
40000 +1	000000 000100 0 0 000011
40000 +4	000000 000100 0 0 000100
50000 +1	000000 000101 0 0 000100
50000 +4	000000 000101 0 0 000101
60000 +1	000000 000110 0 0 000101
60000 +4	000000 000110 0 0 000110
70000 +1	000000 000111 0 0 000110
70000 +4	000000 000111 0 0 000111

Once input B is changed, the sum needs to propagate to the sum output signal.