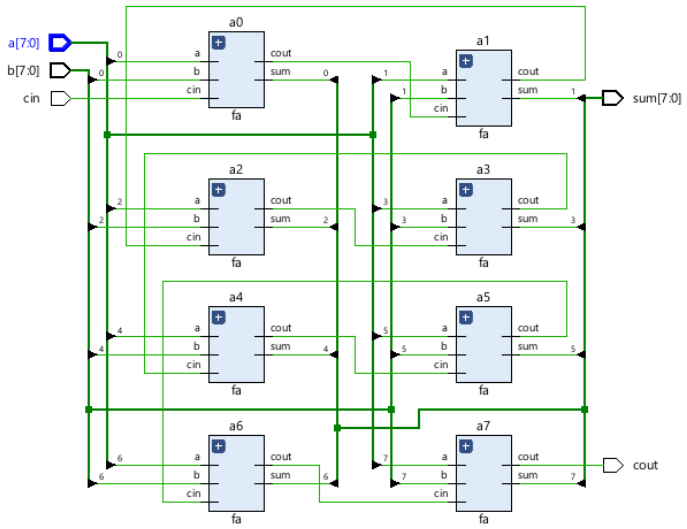
E-mail: emjo2109@student.miun.se
Kurs: ET061G

# Digitalteknik lab 2




Emil Jons, ET061G, Lab 2 VHDL 8-bit adder

E-mail: emjo2109@student.miun.se
Kurs: ET061G

# Results

## 1.1 Full Adder

Before writing the code for the module of the 1-bit full adder, a function was created which described the behavior of how the full adder should work. The truth table of how the function would function was created, with the two outputs separated. For both outputs a Karnaugh diagram was drawn to simplify each function. The function “s”, equation (2), could be translated into VHDL code and build up with XOR gates, which resulted in line 10 in the code. The function “Cout”, equation (1), was translated into VHDL code which resulted in line 11 in the code

*Tabell 1: Truth table for the 1-bit full adder*

| a | b | Cin | Cout | s |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

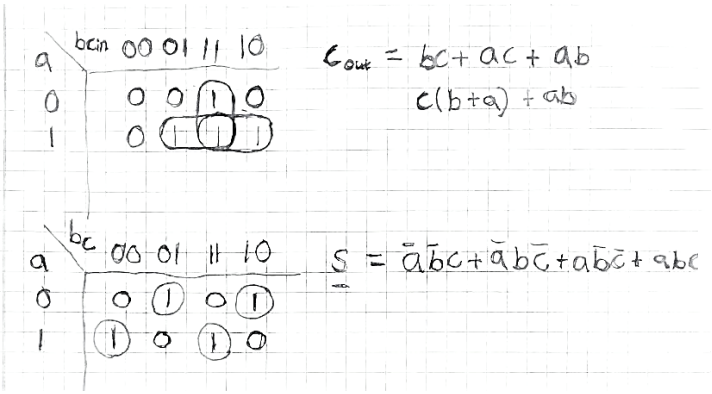



*Figure 1: Karnaugh maps for the functions*

The code below defines the full adder component and how it should behave.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity fa is
5  port (a,b,cin:in std_logic;
6      sum,cout: out std_logic);
7  end fa;
8  architecture behavioral of fa is
9  begin
10     sum <= a xor b xor cin;
11     cout <= (a and b) or (b and cin) or (cin and a);
12 end behavioral;
```

*Code 1: Full adder*

E-mail: emjo2109@student.miun.se
Kurs: ET061G

## 1.2 Simulating the full adder

In order to test the full adder, the supplied testbench was included in the project files. The testbench goes through all possible combinations of a, b and cin, like what is presented in table 1. With a functional testbench a simulation could be setup and test how the full adder behaves. Below is the waveform (figure 2) representing all the combinations as well as the sum and carry bit.

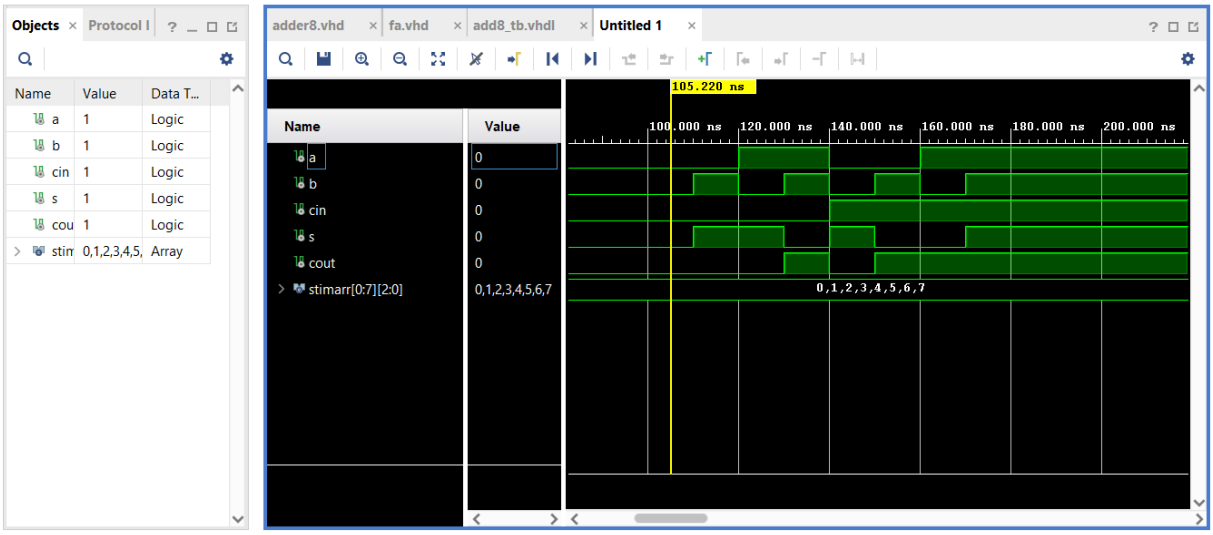



*Figure 2: Waveform of the simulation of the full adder*

When studying the waveform, you notice that the values correspond to the aforementioned truth table's values. When only one input is 1, the sum is 1. When both inputs are 1 and not Cin, the carry is 1. When all inputs are 1, the sum and the carry are 1, etc.

E-mail: emjo2109@student.miun.se
Kurs: ET061G

## 1.3 Testing the program on the FPGA board

In order to test the program on the FPGA board a constrains file was required. The file describes what ports are being used in on the board and how it correlates to the models, in the code. In this case, three inputs (a, b, cin) and two outputs (cout, s) were required. The three inputs were represented by switches and the outputs represented by led lights on the board. Looking at the figures (3, 4, 5) below, three different combinations are presented of different inputs being activated or not.

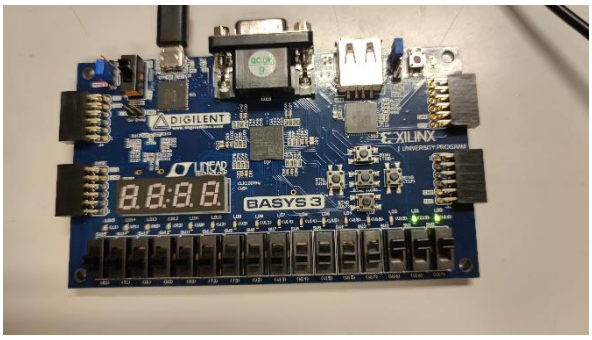



*Figure 4: FPGA board where inputs a, b, cin are 1*

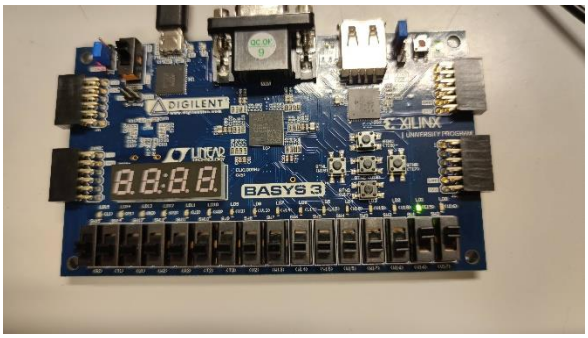



*Figure 3: FPGA board where inputs a, b are 1*

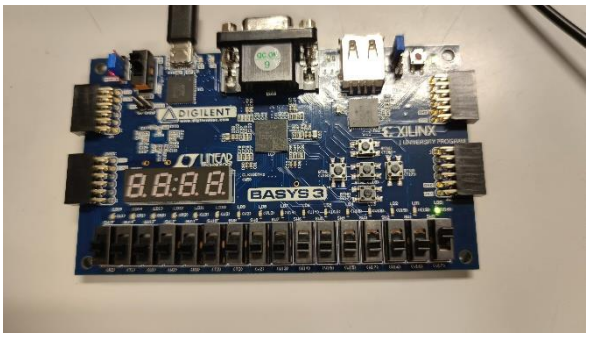



*Figure 5: FPGA board where input a is 1*

The switches from left to right represent the inputs cin, b and a. And the led's from left to right represent the outputs cout and s.

E-mail: emjo2109@student.miun.se
Kurs: ET061G

## 2.0 Structural description of an 8-bit adder

The structure of the 8-bit adder was designed according to the given figure from the lab instructions (figure 6). By connecting 8 full adders the 8-bit adder was created.

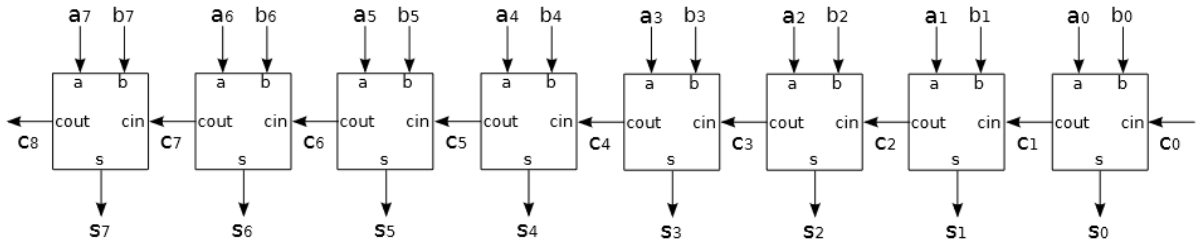



*Figure 6: Structure of the 8-bit adder*

The adder will now accept two 8-bit numbers, a and b, add them together and give an 8-bit answer, s, as well as a carry bit. The VHDL code for this is presented below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add8 is
port(a,b : in std_logic_vector(7 downto 0);
     cin : in std_logic;
      sum : out std_logic_vector(7 downto 0);
      cout : out std_logic);
end add8;
-- Structure
architecture struct of add8 is
signal cary : std_logic_vector(6 downto 0);

component fa is
  port (a,b,cin:in std_logic; sum,cout: out std_logic);
end component;
  begin
     a0:fa port map (a(0),b(0),cin,sum(0),cary(0));
     a1:fa port map (a(1),b(1),cary(0),sum(1),cary(1));
     a2:fa port map (a(2),b(2),cary(1),sum(2),cary(2));
     a3:fa port map (a(3),b(3),cary(2),sum(3),cary(3));
     a4:fa port map (a(4),b(4),cary(3),sum(4),cary(4));
     a5:fa port map (a(5),b(5),cary(4),sum(5),cary(5));
     a6:fa port map (a(6),b(6),cary(5),sum(6),cary(6));
     a7:fa port map (a(7),b(7),cary(6),sum(7),cout);
end struct;
```

*Code 2: 8-bit adder structural description*

E-mail: emjo2109@student.miun.se
Kurs: ET061G

## 2.1 Behavioral description of an 8-bit adder

The final step before simulating the 8-bit adder was to create a behavioral description. This tells the program what to do, at a higher abstraction level. The + operator was used, telling the program to add the two values. In order to account for the carry bit, “0” was concatenated to both of the vectors a and b. This added a zero as the MSB making the values 9 bits instead of 8 long. The ninth bit, or bit 8, will be the carry bit and the last eight bits, or 7 downto 0, will be the sum. Below is the code of the described behavior.

```
architecture behave of add8_behave is
    signal su: std_logic_vector(8 downto 0);
begin
    su <= ('0' & a) + ('0' & b);
    cout <= su(8);
    sum <= su(7 downto 0);
end behave;
```

*Code 3: 8-bit adder behavioral description*

## 2.2 Simulating the 8-bit adder

With the supplied testbench from the lab instructions, a simulation of the 8-bit adder was performed resulting in the waveform partly presented below. The image is zoomed in a lot for a better view of what is happening with the values. The values are represented in hexadecimal, so $4b_{hex}$ is $75_{dec}$ and $7f_{hex}$ is $79_{dec}$. In this case the action performed is 4 + 75 = 79. Because the value 79 fits in 8 bits, there is no carry bit.