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

Digitalteknik lab 3

Emil Jons, ET061G, Lab 3 VHDL 4 Bit Components

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

4-bit adder component

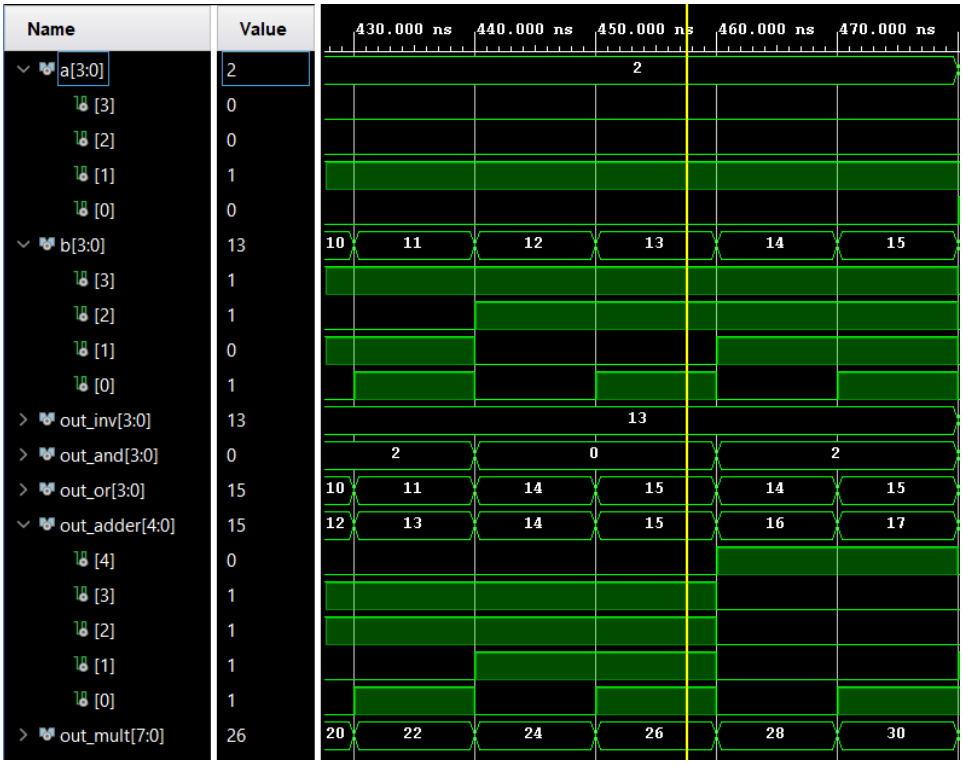
The adder works by adding two four bit numbers which maximum results in a five bit number, where the fifth bit is the carry bit, the MSB. When adding two numbers of *n* bits the maximum number of bits required to display the result is *n+1* bits. For instance when adding two 8 bit numbers you need 9 bits for the answer, etc. Inplementing this in vhdl you will need to concatenate a '0' as the MSB to both the numbers, making them five bits.

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


entity add4 is
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
s_out : out STD_LOGIC_VECTOR (4 downto 0));
end add4;

architecture behave of add4 is
begin
    s_out <= ( '0' & in_1 ) + ( '0' & in_2 );
end behave;
```

Below is the signal waveform where 2 is added to 13. The result is 15 which is displayed in out_adder. There is no overflow so the number 15 fits in four bits, leaving the fifth bit empty, as 0.



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

4-bit Multiplier component

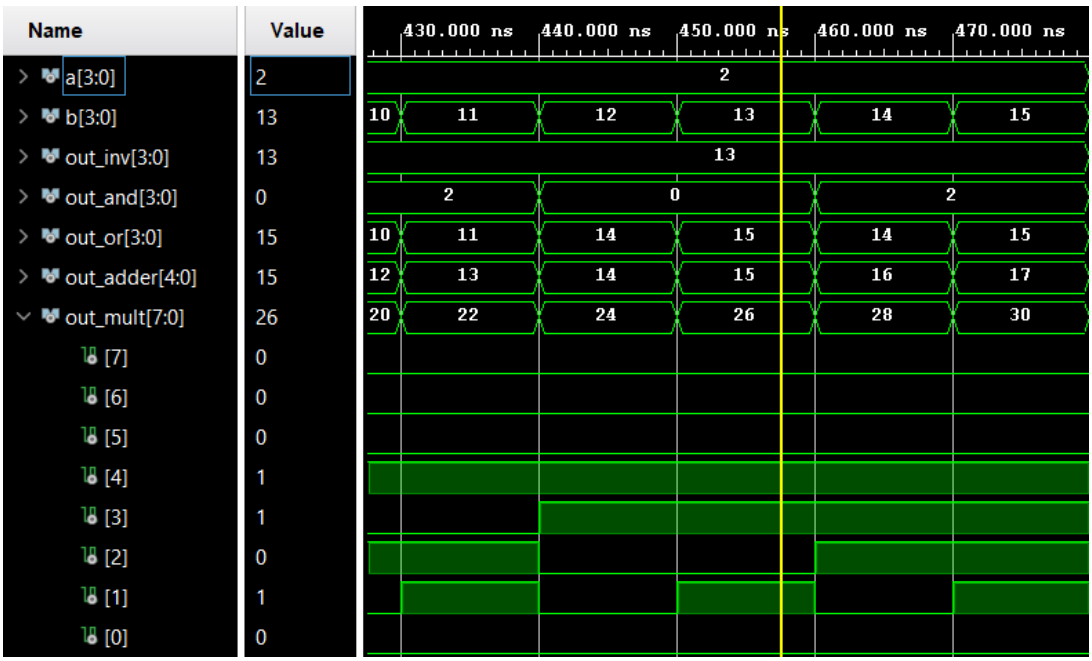
The multiplier works by multiplying two four bit number, resulting in a maximum size of an eight bit number.

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


entity mul4 is
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
m_out : out STD_LOGIC_VECTOR (7 downto 0));
end mul4;

architecture behave of mul4 is
begin
    m_out <= in_1 * in_2;
end behave;
```

Below is the signal waveform where 2 is multiplied by 13. The result is 26 which is displayed in out_mult.



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

4-bit Inverter component

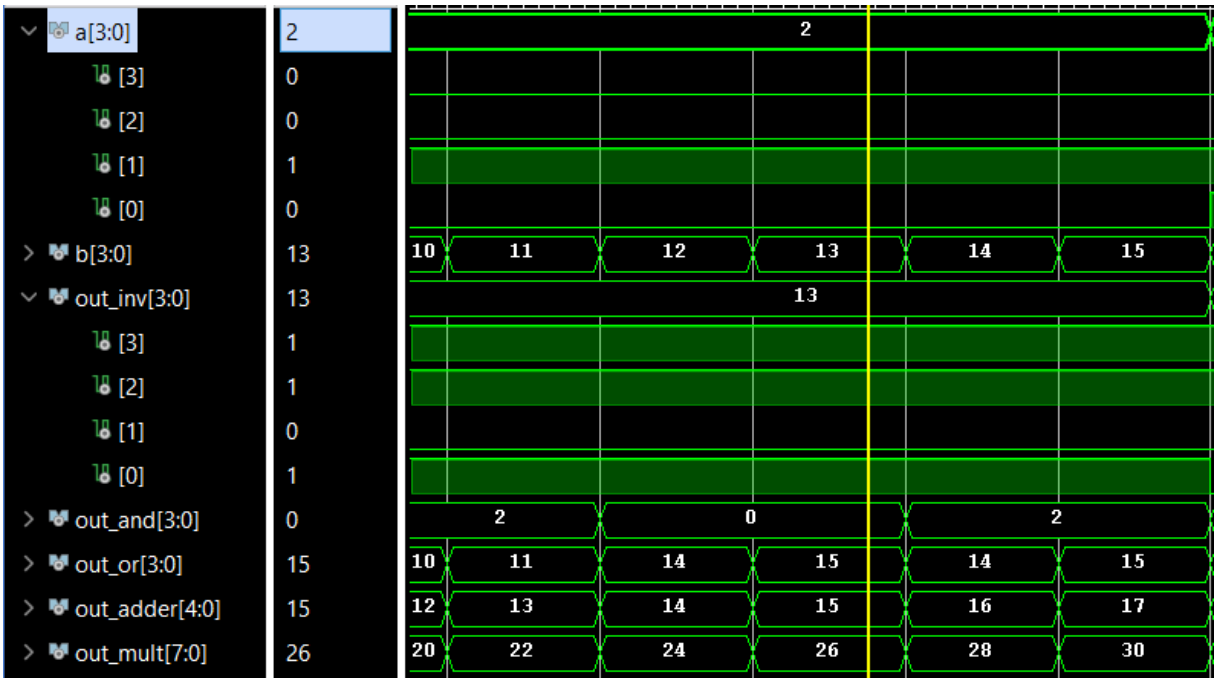
The inverter simply inverts the input signal to the output. Both the input and the output will be four bits due to there is no change in the amount of bits need to display the output.

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

entity invrt4 is
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
i_out : out STD_LOGIC_VECTOR (3 downto 0));
end invrt4;

architecture behave of invrt4 is
begin
    i_out <= NOT ( in_1 );
end behave;
```

Below is the signal waveform where 2, or “0010” is inverted. The result is 13, or “1101” which is displayed in out_inv.



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

4-bit AND component

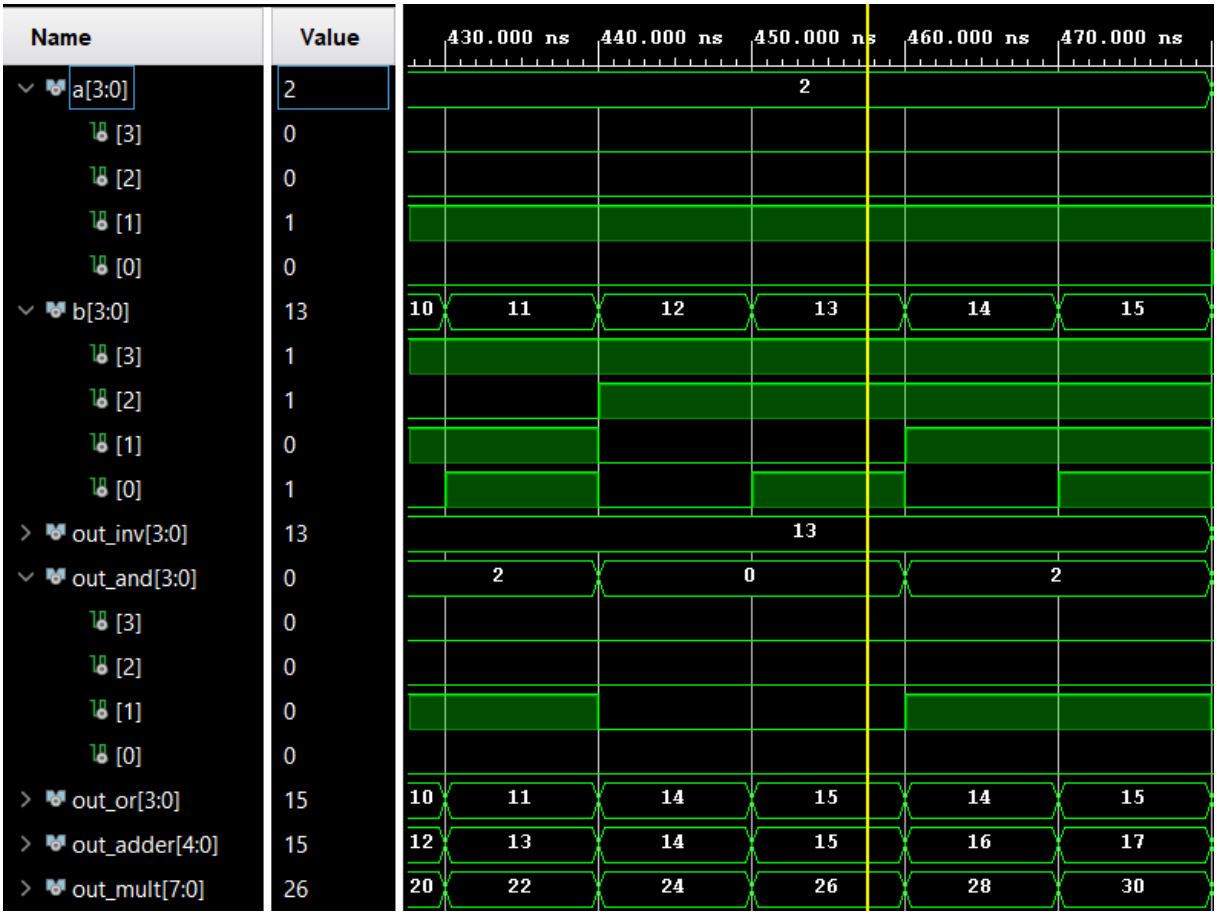
The AND component performs the AND operation on the two inputs.

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

entity bit_and4 is
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
a_out : out STD_LOGIC_VECTOR (3 downto 0));
end bit_and4;

architecture behave of bit_and4 is
begin
    a_out <= ( in_1 AND in_2 );
end behave;
```

Below is the signal waveform where the logical operation AND is performed with 2 and 13, or “0010” and “1101”. The result is 0 which is displayed in out_and.



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

4-bit OR component

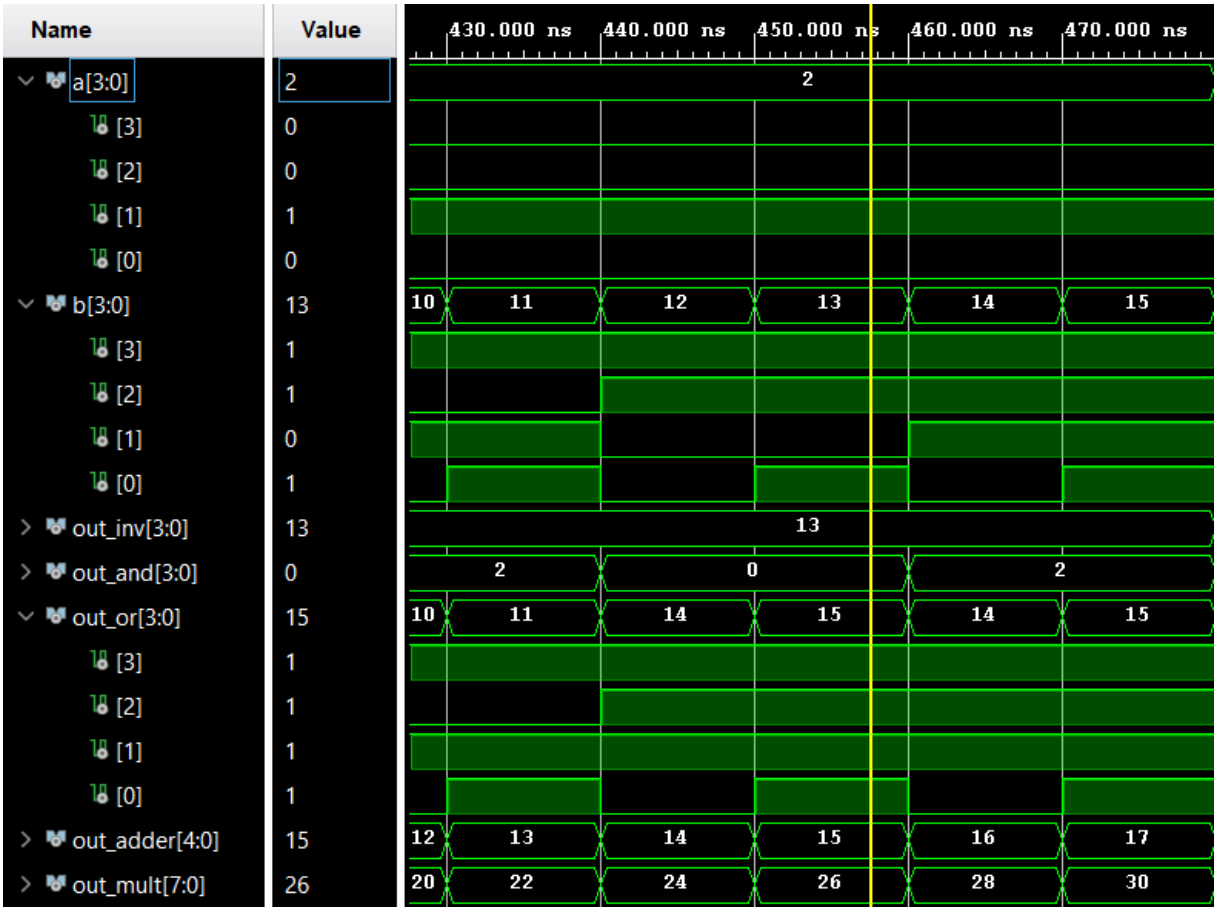
The OR component performs the OR operation on the two inputs.

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

entity bit_or4 is
Port ( in_1 : in STD_LOGIC_VECTOR (3 downto 0);
in_2 : in STD_LOGIC_VECTOR (3 downto 0);
o_out : out STD_LOGIC_VECTOR (3 downto 0));
end bit_or4;

architecture behave of bit_or4 is
begin
    o_out <= ( in_1 OR in_2);
end behave;
```

Below is the signal waveform where the logical operation OR is performed with 2 and 13, or “0010” and “1101”. The result is 15 which is displayed in out_and.



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

Explaining how the testbench works

Testbench for the components:

Below is the VHDL code of the component decleration in the testbench. It declares what ports are being used in the components. In our case we use all the ports, resulting in a copy of the code from each of the components.

```
Library IEEE;
use work.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.all;

entity bit4_tb is
end bit4_tb;

architecture behave of bit4_tb is

  component add4
    port (
      in_1, in_2 : in  std_logic_vector(3 downto 0);
      s_out  : out std_logic_vector(4 downto 0));
  end component;

   component mul4
    port (
      in_1, in_2 : in  std_logic_vector(3 downto 0);
      m_out  : out std_logic_vector(7 downto 0));
  end component;

  component invrt4
    port (
      in_1 : in  std_logic_vector(3 downto 0);
      i_out  : out std_logic_vector(3 downto 0));
  end component;

  component bit_and4
    port (
      in_1, in_2 : in  std_logic_vector(3 downto 0);
      a_out  : out std_logic_vector(3 downto 0));
  end component;

  component bit_or4
    port (
      in_1, in_2 : in  std_logic_vector(3 downto 0);
      o_out  : out std_logic_vector(3 downto 0));
  end component;
```

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

Testbench signals and port mapping

The signal part of the testbench, declares what signals are to be used. The signals a and b being the input to each component and the rest being the output signals depending on what component is used. The portmapping part of the testbench connects the ports to the signals.

```
  signal a, b : std_logic_vector(3 downto 0);
  signal out_inv, out_and, out_or : std_logic_vector(3 downto 0);
  signal out_adder : std_logic_vector(4 downto 0);
  signal out_mult : std_logic_vector(7 downto 0);

begin  -- behave
add4_0 : add4 port map (
    in_1           => a,
    in_2           => b,
    s_out          => out_adder);

mul4_0 : mul4 port map (
  in_1        => a,
  in_2        => b,
  m_out     => out_mult);

invrt4_0 : invrt4 port map (
    in_1           => a,
    i_out          => out_inv);

bit_and4_0 : bit_and4  port map (
    in_1           => a,
    in_2           => b,
    a_out          => out_and);

bit_or4_0 : bit_or4  port map (
    in_1           => a,
    in_2           => b,
    o_out          => out_or);
```

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

Testbench process

The final part of the testbench contains the process, which in our case loops through all possible combinations of inputs that a and b can have. It does this with a nested loop, that goes from 0 to 15 with the a value and 0 to 15 with the b value. This gives the components all possible inputs and in turn all possible outputs for each of the component. This is done in order to test if the components behave as intended when faced with a specific input.

```
process
   variable a_vector, b_vector : std_logic_vector(3 downto 0):=(others=>'0');
begin  -- process

  for a_vector in 0 to 15 loop
    for b_vector in 0 to 15 loop
      a <= conv_std_logic_vector(a_vector,4);
      b <= conv_std_logic_vector(b_vector,4);
      wait for 10 ns;
    end loop;  -- b_vector
  end loop;  -- a_vector
end process;

end behave;
```