Digitalteknik lab 3

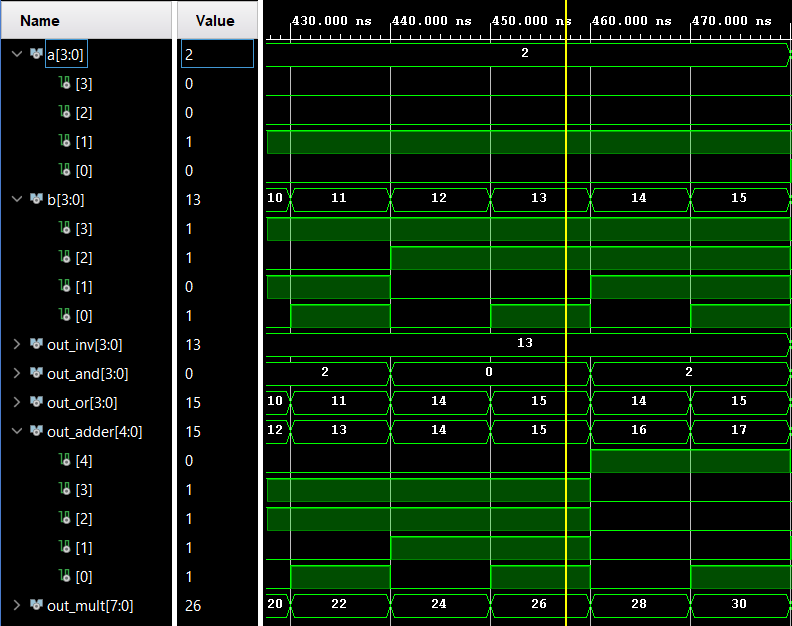
Emil Jons, ET061G, Lab 3 VHDL 4 Bit Components

**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.



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.

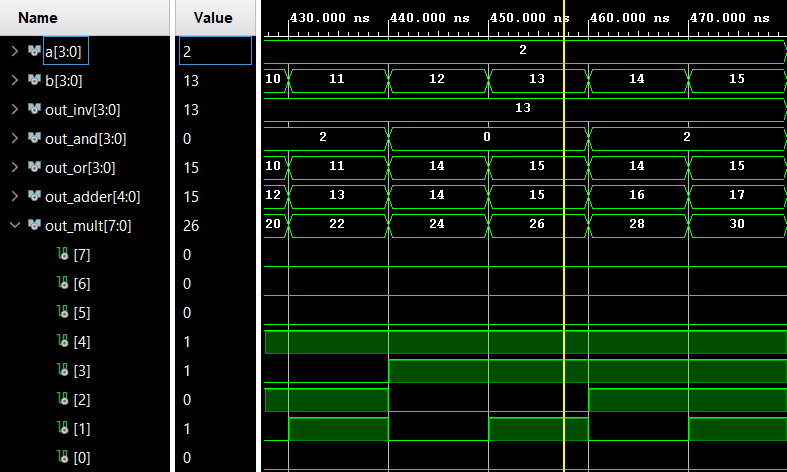
****

**4-bit Multiplier component**

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

****

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

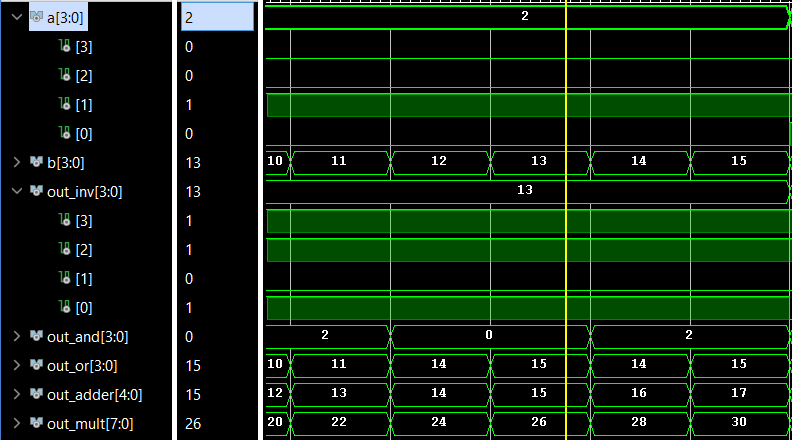


**4-bit Inverter component**

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.

****

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

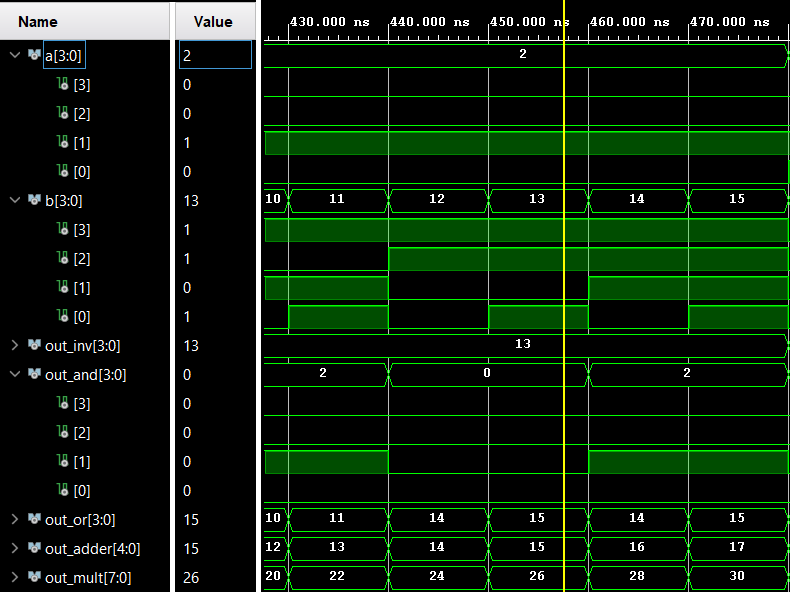
****

**4-bit AND component**

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

****

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.

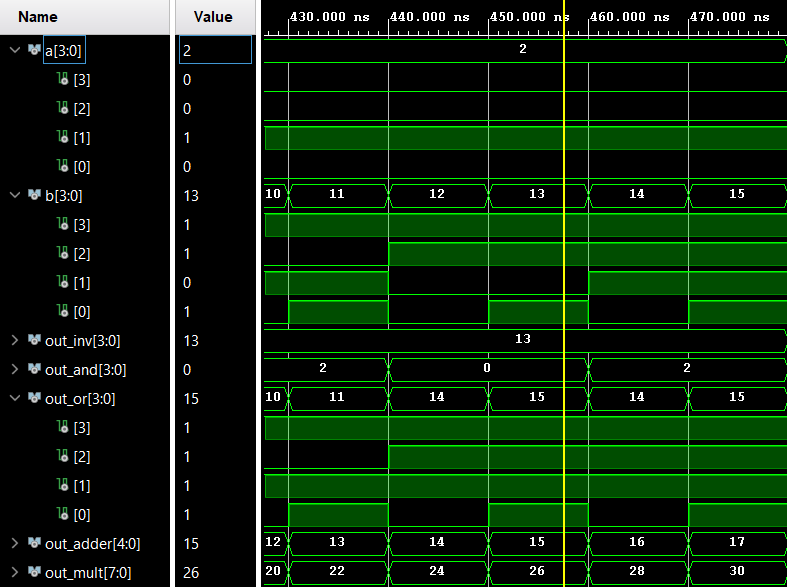
****

**4-bit OR component**

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

****

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.



**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.

****

**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.

****

**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.

****