Module exercise: Static Random Access Memory (SRAM)

Produced by the members of projectgroup A1
Kevin Hill 4287592
Rens Hamburger 4292936

Commissioned by
Martin Geertjes 4324285
Jeroen van Uffelen 42326690

26 november 2014
Version 1.0

1 Abstract

As part of an introduction to VHDL systems, an assignment, commissioned by Martin Geertjes and Jeroen van Uffelen, was completed. The assignment, part of the EPO-3 project of the Electrical Engineering bachelor of the TU Delft, was to create SRAM, in such a way that it could be implemented on a chip. The SRAM has to be capable of reading/writing within one clock period. This was done by creating an encoder, that puts the input on the correct place of the memory. When one wants to read, the decoder takes the address, picks the appropriate place in the memory and outputs it. This way the amount of in and the output is greatly reduced. The SRAM was designed in VHDL, and *Modelsim* was used for the simulation and testing of the design. It was then synthesized by *Go With The Flow*, and this synthesis was tested again using *Modelsim*. This way, all levels of the design were tested, from the raw VHDL code to the design on transistor-level. This way, it is possible to check each level of design for flaws.

2 Introduction

A computer needs memory. This memory needs to be as fast as possible, to achieve good performance. SRAM is an example of fast (and thus expensive) memory. However, the question is how the memory communicates to whatever it is that is connected to the memory. This can be done using an encoder and a decoder. This way, the number of ports connected to the memory is drastically reduced too. This leads the way for customizability too, for instance if someone is not happy with the size of RAM in a device, or a module breaks, perhaps it can be replaced without too much trouble (like RAM in modern computers).

3 Specifications

The SRAM was made after specifications given by an another couple with in the project group. These specifications where then discussed with the tutor to be sure the requested specifications aren't to complex to finish within the given time.

3.1 The specifications

Read speed: ability to write or read every clock cycle.
Size: 64 bits

- 4 Address lines
- 4 Directional (one way) data line for read
- 4 Directional (one way) data line for write

Use generic maps for an ease implementation of length and width.
Write and read line 0 = write, 1=read

Inputs

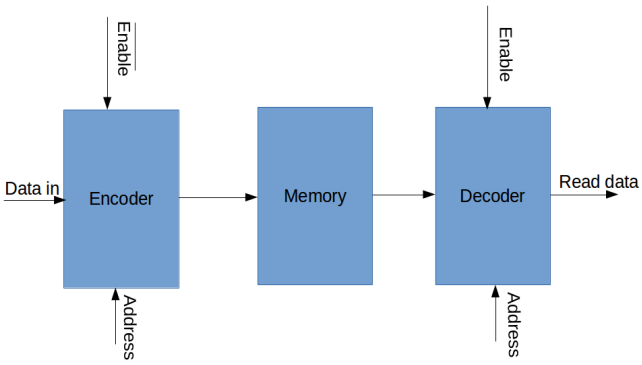
- 4 address lines
- 4 Data lines for writing
- Read write line

Output

- 4 data lines to be able to read the data

4 Design

To successfully make SRAM, the memory itself needs to be made, and the interface. This interface is used to reduce the number of input/output pins attached to the memory. now, instead of 65 input pins and 64 output pins, there are only 11 input pins and 4 output pins. The complete SRAM can be split up in three parts: the encoder, the memory and the decoder. As shown in figure 1.



Figuur 1: The input/output of the SRAM

4.1 Encoder

In the case of writing words to the memory, an encoder stores the word at the correct place in the memory by using the address-bits as select-bits. Thus, the data that needs to be written is done so correctly, and to the correct word (and only there). The encoder has the following input: 4 data-input lines, 4 address-lines, and the (inverted) enable-, clock- and reset signals. The output of the encoder consists of 64 memory-signals.

4.2 Memory

The memory itself is an array of 16 by 4 flip-flops. The input of all flip-flops is attached to the encoder, the output is attached to the decoder. All flip-flops also have a clock- and reset-signal as input.

4.3 Decoder

64 bits of memory can be saved in a 16x4 order. the word size is 4 bits, and the amount of words in the memory is 16. This means that, using 4 address bits, the correct word can be selected (because $2^4 = 16$). So, if one wants to know the second stored word, then the address needs to be '0001'. Using a decoder (or demux), this word is then assigned to the output of the decoder. The decoder has 64 data-input lines, and the enable, clock and reset-signals are also input. The output of the decoder consists of 4 data-out lines.

5 Result

The module will be tested in several ways of implementation in hardware code. For testing purpose we used the testbench in 6.1.3. The first test uses the basic VHDL code. The second test uses the synthesized code whereby the VHDL code is translated into sea of gate cells. For the third test we make a layout where an extraction is done of the VHDL code.

5.1 Behavioural

This is the original test. It looks exactly as it is supposed to look, and is successful (as seen in figure 2). This leads to the conclusion that the SRAM works as foreseen. This means that *Modelsim* knows what to do and how to handle the VHDL code, but does not say anything about how it would function on a chip in real life. Thus, two more tests need to be performed.

5.2 Synthesized

After successfully getting the SRAM working in VHDL behavioural description, the implementation on the chip was tested by synthesizing the code. This was done using the sea of gates library and *Go With The Flow*. The result can be seen in figure 3. In the synthesized SRAM module, delays are implemented. This means that if the address value changed too late (a clock edge comes too soon after) then the read-data-line will not contain the expected data at the expected time. Instead, it will contain the expected data one clock period later than expected.

5.3 Layout

After evaluating the synthesized result the circuit is then simulated with *Row placer* and *Trout*. From this circuit, the VHDL code is extracted for one more simulation in *Modelsim*. The result are shown in figure 4. The results are the same as all the other results, showing that no bugs can be found in the SRAM at transistor-level.

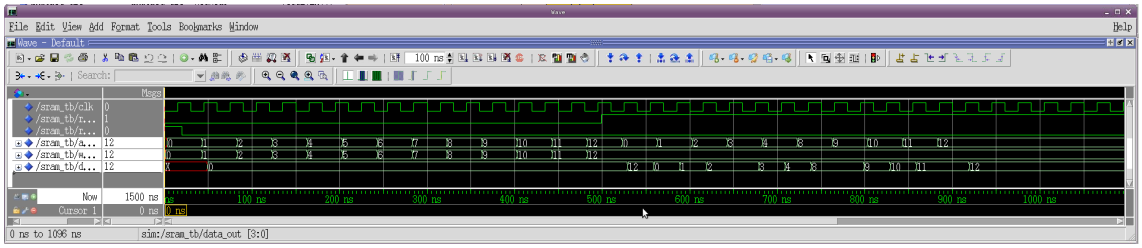
5.3.1 Switch level

To be even more sure everything works as expected, it is possible to do a switch level simulation the result can be found in figure 6. To be able to check the result of the switch-level simulation, a comparison was done to see if everything works as expected. The results are shown in figure 5. There are few glitches when signals change close to the rising edge of the clock.

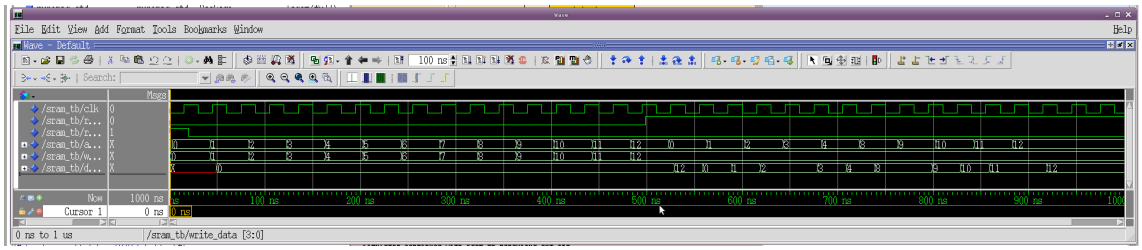
6 Appendix

6.1 Results

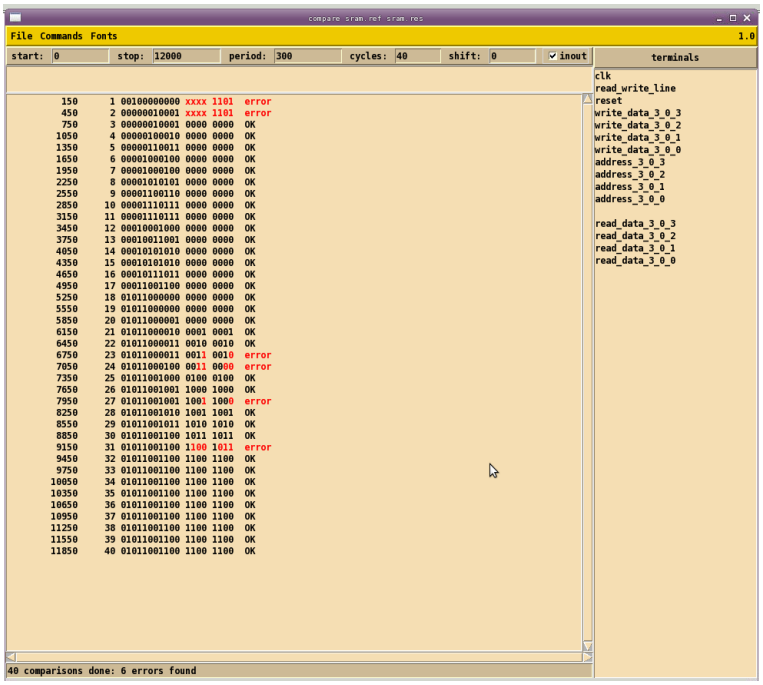
Figuur 2: Simulation result on behavioural level



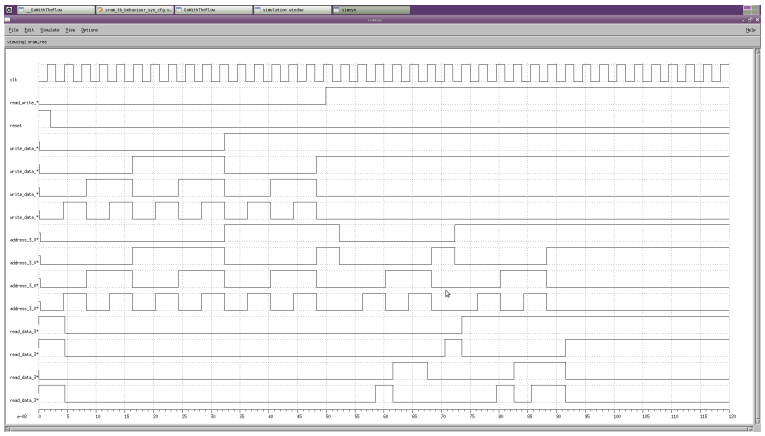
Figuur 3: Simulation result after synthesis of the circuit



Figuur 4: Simulation result on Sea of Gates cells level



Figuur 5: Comparison extracted and synthesized



Figuur 6: Switch-level simulation result

6.1.1 SRAM entity VHDL code

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.Numeric_Std.all;

entity sram is
    port(clk                    :in     std_logic;
         read_write_line        :in     std_logic;                          --0=write data 1=data lezen
         reset                  :in     std_logic;
         write_data             :in     std_logic_vector(3 downto 0);       --De data ingang
         address                :in     std_logic_vector(3 downto 0);       --Het aantal addressen in dit geval 16 (2^4)
```

```
        read_data               :out    std_logic_vector(3 downto 0));          --De data uitgang
end sram;
```

6.1.2 SRAM architecture VHDL code

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.Numeric_Std.all;


architecture behaviour of sram is
type mem_array is array (0 to (2**address'length) -1) of std_logic_vector((write_data'length -1) downto 0);--Het ram el
type memory_state is (steady,state_reset);                                      --Het definiren van de te gebruiken sta
signal ram : mem_array;
signal new_ram : mem_array;
signal state,new_state : memory_state;
signal data_out : std_logic_vector(3 downto 0);
begin
        sram : process(clk)                                                     --Bepalen van de state
        begin
                if rising_edge(clk) then
                        if (reset = '1') then
                                state <= state_reset;
                        else
                                ram <= new_ram;
                                state <= new_state;
                                read_data <= data_out;
                        end if;
                end if;
        end process sram;
        sram_state : process(state,clk,read_write_line,write_data)      --Het uit voeren van de opdrachten binnen de st
        begin
                case state is
                        when steady =>
                                if read_write_line ='1' then            --Lezen van data
                                        data_out <= ram(to_integer(unsigned(address)));
                                        new_state <= steady;
                                        new_ram <= ram;
                                elsif read_write_line = '0' then        --Data schrijven
                                        for J in 0 to 15 loop
                                                if (J = to_integer(unsigned(address))) then
                                                        new_ram(J) <= write_data;
                                                else
                                                        new_ram(J) <= ram(J);
                                                end if;
                                        end loop;
                                        data_out <= "0000";
                                        new_state <= steady;
                                end if;
                        when state_reset =>             -- Alles initialiseren en dus op 0 zetten
                                new_ram <= (others => (others => '0'));
                                read_data <= "0000";
                                new_state <= steady;
            end case;
        end process sram_state;
end behaviour;
```

6.1.3 SRAM testbench architecture VHDL code

```
library IEEE;
use IEEE.std_logic_1164.ALL;

architecture behaviour of sram_tb is
component sram is
        port(clk                :in     std_logic;
        read_write_line         :in     std_logic;
        reset                   :in             std_logic;
        write_data              :in     std_logic_vector(3 downto 0);
        address                 :in     std_logic_vector(3 downto 0);
        read_data               :out    std_logic_vector(3 downto 0));
end component sram;
```

```
signal  clk,read_write_line,reset       :       std_logic;
signal  address                         :       std_logic_vector(3 downto 0);
signal  write_data,data_out             :       std_logic_vector(3 downto 0);


begin
        clk     <=      '0' after 0 ns,
                        '1' after 15 ns  when clk /= '1' else '0' after 15 ns;

        reset   <=      '1' after 0 ns,
                        '0' after 20 ns;


        read_write_line <='0' after 0 ns,
                        '1' after 750 ns,
                        '0' after 5000 ns;

        write_data<=    "0000" after 1 ns,
                        "0001" after 43 ns,
                        "0010" after 83 ns,
                        "0011" after 123 ns,
                        "0100" after 163 ns,
                        "0101" after 203 ns,
                        "0110" after 243 ns,
                        "0111" after 283 ns,
                        "1000" after 323 ns,
                        "1001" after 363 ns,
                        "1010" after 403 ns,
                        "1011" after 443 ns,
                        "1100" after 483 ns;

        address <=      "0000" after 3 ns,
                        "0001" after 43 ns,
                        "0010" after 83 ns,
                        "0011" after 123 ns,
                        "0100" after 163 ns,
                        "0101" after 203 ns,
                        "0110" after 243 ns,
                        "0111" after 283 ns,
                        "1000" after 323 ns,
                        "1001" after 363 ns,
                        "1010" after 403 ns,
                        "1011" after 443 ns,
                        "1100" after 483 ns,
                        "0000" after 523 ns,
                        "0001" after 563 ns,
                        "0010" after 603 ns,
                        "0011" after 643 ns,
                        "0100" after 683 ns,
                        "1000" after 723 ns,
                        "1001" after 763 ns,
                        "1010" after 803 ns,
                        "1011" after 843 ns,
                        "1100" after 883 ns;

        sram_pm: sram port map(clk,read_write_line,reset,write_data,address,data_out);
end behaviour;
```

7 Discussion

A few problems were encountered during the simulation. At first, the simulation would run, but return wrong results. After changing the code a few times, even starting from scratch, it appeared that the speed the clock runs at was too high, so we had to slow the clock down a little, and all functioned perfectly. It might prove useful to keep in mind that it is not only possible, but very likely that the clock period is too small.

8 Bibliography

As source of information, both the internet and the following books have been used:

8.1 Used books:

- Digital Integrated Circuits

8.2 Used websites:

- dingen