"FPGA-Programming" Exercise Sheet VIII:

Memory Blocks

In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. In this exercise we will examine the general issues involved in implementing such memory.

As target hardware always choose FPGA chip Cyclone V SoC 5CSEMA5F31C6

Following figure shows the symbol of a RAM (Random Access Memory) module. It contains 32 four-bit words (*rows*), which are accessed using a five-bit address port, a four-bit data port, and a write control input.

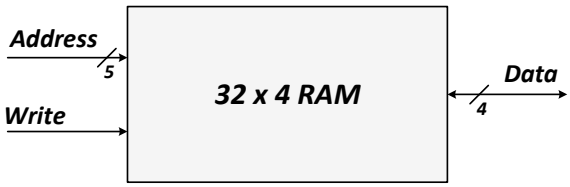

Figure 0.1: Symbol of a RAM module

The Cyclone V FPGA chip of the DE1-SoC boards includes dedicated memory resources, named **M10K** blocks. Each **M10K** offers 10240 memory bits. The blocks can be configured to implement memories of various sizes.

A common term used to specify the size of a memory is its **aspect ratio**, which gives the **depth** in words and the **width** in bits, so **aspect ratio** = **depth** $\times$ **width**. Common aspect ratios are $\mathbf{8K \times 1}$ and $\mathbf{2K \times 4}$ as well as many further. In this exercise the aspect ratio $\mathbf{2K \times 4}$ will be utilized, using only the first 32 words in the memory.

There are two additional important features of the **M10K** blocks that have to be mentioned. First, they include registers that can be used to synchronize all of the input and output signals to a clock input. The registers on the input ports must always be used, and the registers on the output ports are optional. Second, the blocks have separate ports for data being written to the memory and data being read from the memory.

Given these in formations, in this exercise sheet a modified 32×4 RAM module, as illustrated in following figure should be implemented. It includes registers for the address, data input, and write ports, and uses a separate unregistered data output port.

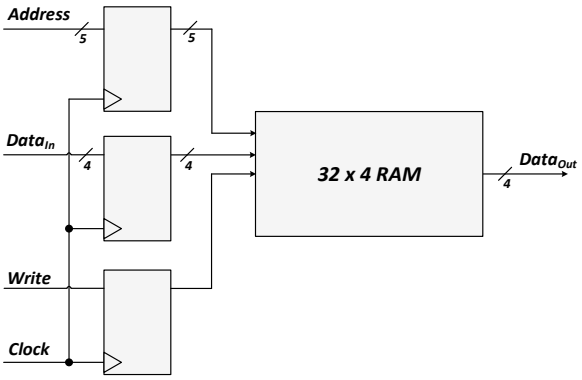

Figure 0.2: RAM circuit

1) *RAM memory with LPM module (I)*

Commonly used logic structures, such as adders, registers, counters and memories, can be implemented in an FPGA chip by using prebuilt modules that are provided in libraries. In this exercise we will use a module called the **altsyncram** to implement the memory from above figure.

A tutorial for using LPM modules in Quartus 17.0 can be found following the link:

`ftp://ftp.altera.com/up/pub/Intel_Material/17.0/Tutorials/VHDL/Using_Library_Modules.pdf`

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_RAM**.

ii) From the **IP Catalog** choose the **RAM: 1-PORT** module, which can be found under **Basic Functions** $\rightarrow$ **On Chip Memory**. Select **VHDL** as file type and name the module **e_ram32x4.vhd**. Specify a RAM size of 32 four-bit words and select **M10K** for the implementation. Ensure that only one clock will be used for the memory register and deactivate the register of the output ports. Accept the remaining default settings and create the IP variation.

iii) Examine the created file **e_ram32x4.vhd**.

iv) Instantiate this sub-level entity in the top-level entity (**e_my_RAM**). Use the following signals:

sl_Clock, **sl_Wr**, **slv_DataIn**, **slv_Address**, **slv_DataOut**

v) Compile the circuit. From the **Compilation Report** observe that the Quartus compiler uses 128 bits of one **M10K** memory cell to realize the circuit.

vi) Create a test bench, simulate the behaviour of the circuit and ensure that data can be written to and read from memory.

2) *RAM memory with LPM module (II)*

In this part, the memory circuit of part 1) should be used in an example circuit. Therefore the switches should be used in order to load data into the memory. The memory contents should be displayed on the 7-segment displays.

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_RAM2**.

ii) Describe the 7-segment decoder named **e_hex7seg**, which decodes a 4-bit vector into 7-bit for driving a 7-segment display showing hexadecimal notation. Use a *combinatorial process* together with a **case** statement.

iii) Create the same LPM module of the RAM memory named **e_ram32x4.vhd**, as in part 1).

iv) Create the top-level entity (**e_my_RAM2**), which integrates the memory. Use switches $SW[3-0]$ for supplying the input data, switches $SW[8-4]$ for selecting the RAM address. Switch $SW[9]$ should be used for the write signal, push-button $KEY[0]$ as manual clock input. The selected address should be displayed on the 7-segment displays $HEX5$ and $HEX4$, the RAM input data on $HEX2$, the RAM output on $HEX0$.

v) Import the necessary pin assignments, compile, download the circuit and test its functionality in hardware.

3) *Self-made RAM memory*

Instead of creating a memory module subcircuit by using the IP Catalog, the required memory can also be implemented by specifying its structure in VHDL code. In a VHDL-specified design it is possible to define the memory as a multidimensional **array**. A 32×4 array, which has 32 words with 4 bits per word, can be declared by following statement:

```
type t_mem is array(0 to 31) of std_logic_vector(3 downto 0);
signal a_mem: t_mem;
```

In the Cyclone series of FPGAs, such an array can be implemented either by using the flip-flops that each logic element contains or, more efficiently, by using the built-in memory blocks. The help of Quartus (Keyword: **Inferred Memory)** offers further VHDL code examples of how memory cells can be specified.

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_ownRAM**.

ii) For the 7-segment decoder (**e_hex7seg**) use the same as in part 2).

iii) Use above VHDL syntax for the description of a multi-dimensional field for the realization of the RAM memory.

iv) Create a VHDL top-level entity (**e_my_ownRAM**). For the realization of the RAM code code a positive edge-triggered *sequential process* with synchronous **Write Enable** for writing and reading the memory. For the specification of the address use an appropriate converting function.

v) Use switches $SW[3-0]$ for data input and switches $SW[8-4]$ for the RAM address. Switch $SW[9]$ should be used for the write signal and push-button $KEY[0]$ as manual clock input. The selected address should be displayed on the 7-segment displays $HEX5$ and $HEX4$, the RAM input data on $HEX2$, the RAM output on $HEX0$.

vi) Import the necessary pin assignments, compile, download the circuit and test its functionality in hardware.

vii) From the **Compilation Report** observe the amount of synthesized memory bits and compare them with the one of part 1).

4) *Simple Dual-Ported RAM with LPM module*

The RAM block of figure 0.1 has a single port that provides the address for both read and write operations. In difference to parts 1) - 3), in this part a further kind of memory module should be generated, in which there is one port for supplying the address for a read operation, and a separate port that gives the address for a write operation.

Perform following steps:

i) Create a new Quartus project for the circuit, named **e_my_2portRAM**.

ii) From the **IP Catalog** choose the **RAM: 2-PORT** module, which can be found under **Basic Functions** $\rightarrow$ **On Chip Memory**.

Select **VHDL** as file type and name the module **e_ram32x4.vhd**. Configure the module in a way that both addresses for read and write will be used. Configure the remaining settings as in part 1). For the case **read-during-write** set the setting to **don't care**. This setting specifies that it does not matter whether the memory outputs the new data being written, or the old data previously stored, in the case that the write and read addresses are the same during a write operation.

iii) In the slide **Mem Init** the usage of a initial memory content can be specified. In this part a so called **memory initialization file (MIF)** should be used. Adjust the necessary settings and specify the name as **ram32x4.mif**. Further information on MIF files can be found in Quartus help. Create a **memory initialization file** holding 32 words with a word width of 4. Specify the contents of the MIF file as follows:

```
depth = 32;
width = 4;
address_radix = HEX;
data_radix = BIN;
content
begin

00: 0000;
01: 0001;
02: 0010;
03: 0011;
04: 0100;
05: 0101;
06: 0110;
...
0D: 1101;
0E: 1110;
0F: 1111;
10: 0000;
11: 0001;
12: 0010;
13: 0011;
14: 0100;
15: 0101;
16: 0110;
...
1D: 1101;
1E: 1110;
1F: 1111;

end;
```

iv) Use the same entity for the 7-segment decoder is in part 2).

v) Describe the entity of a positive edge-triggered D-type flip-flop (**e_flip_flop**) with synchronous low-active reset and synchronous high-active enable input.

vi) Describe the entity of a parametrisable positive edge-triggered b-bit wide register (**e_regne**) with synchronous low-active resent and synchronous high-active enable input.

vii) Create the top-level entity (**e_my_2portRAM**) which instantiates the Dual-Ported RAM. The content of the memory should be displayed in a hexadecimal notation on $HEX0$. Use a *sequential process* in order to generate a enable signal with an exact frequency of 1 Hz by a 26-bit wide count signal. Use this enable signal to increment the read address of the RAM. The process furthermore should exhibit a synchronous low-active reset input, to reset the read address.

viii) The address of the displayed word should be displayed on $HEX3$ and $HEX2$. Use the clock signal $CLOCK\_50$ and push-button $KEY[0]$ as reset input. For the write address use switches $SW[8-4]$, for the data input use switches $SW[3-0]$. Display the write address on $HEX5$ and $HEX4$ the data that should be written on $HEX1$. Ensure that the switches will be synchronised to the 50 MHz clock signal. Therefore use a cascade of two instances of **e_flip_flop** and **e_regne**, respectively.

ix) Import the necessary pin assignments, compile and download the circuit. Verify that the memory initially will be filled with the preset values of the MIF file. Furthermore ensure that data can independently be written to arbitrary addresses.