ANADOLU UNIVERSITY
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

LAB-4
REGISTER FILE, RAM AND ROM UNITS

1 Introduction

You will learn how to implement different memory units: Register File, RAM and ROM in vhdl. In addition to this, you will learn how to instantiate FPGA memory components by CORE generator.

2 Memory Basics

The processor communicates with the memory system over a memory interface. Figure 1 show the simple memory interface used in our singlecycle and multicycle MIPS processor. The processor sends an address over the address bus to the memory system. For a read, write is 0, read is 1 and the memory returns the data on the rddata bus. For a write, write is 1, read is 0 and the processor sends data to the memory on the wrdata bus.

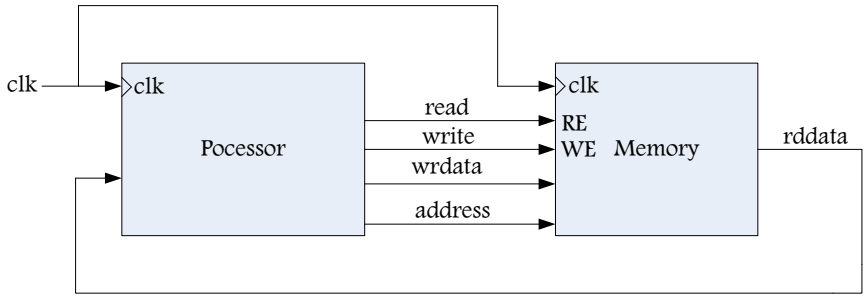

Figure 1: Example memory interface

3 Register File

The first component that you will implement in this lab is a Register File. The Register File has 32 registers which all have length of 32-bits. You will use it in the CPU which you will implement on next labs. There is a figure below which shows the Register File entity.

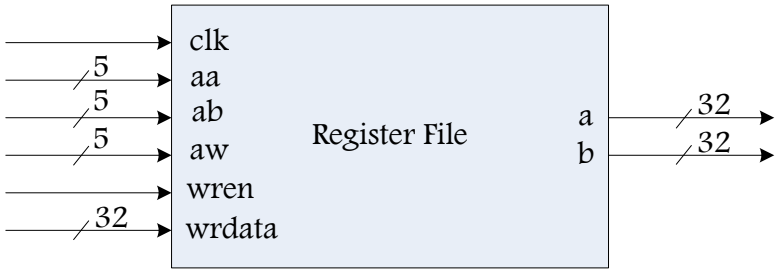

Figure 2: Register File Block Diagram

The Register File has two read ports and one write port. The read ports take 5-bit address inputs, aa and ab, each specifying one of $2^5 = 32$ registers as source operands. They read the 32-bit register values onto read data outputs a and b, respectively. The write port takes a 5-bit address input aw, and a 32-bit write data input wrdata, a write enable input wren and a clock clk.

3.1 Read from the Register File

The read process is asynchronous. The 5-bit inputs aa and ab selects the registers to be read. The read values are put respectively on the 32-bit outputs a and b. In CPU, the synchronous operations are needed. Therefore, a clock must be added to the unit. The address is set by the controller at the beginning of cycle, and the result is available before the end of this cycle.

3.2 Write to the Register File

The write process is synchronous. The wren input signal activates the writing of the value defined by the 32-bit wrdata input to the register at the address specified by the 5-bit aw input. If the write enable is 1, the register file writes the data into the specified register on the rising edge of the clock.

Writing a value to the register at address 0 has no effect, the value of this register is always 0. The address, the data and the wren input signals are set during a cycle. At the rising edge of the clock signal, the data is written to the register in the Register File.

4 Decoder

The decoder activates one of the modules on the memory system at a time. It selects the module according to the address value. Address space allocation is illustrated in the figure below.

For example, with the address 0x10F0, the selected module would be the RAM. The decoder will activate the cs_ram output signal. Recall that at most one chip select (cs) signal can be activated at the same time.

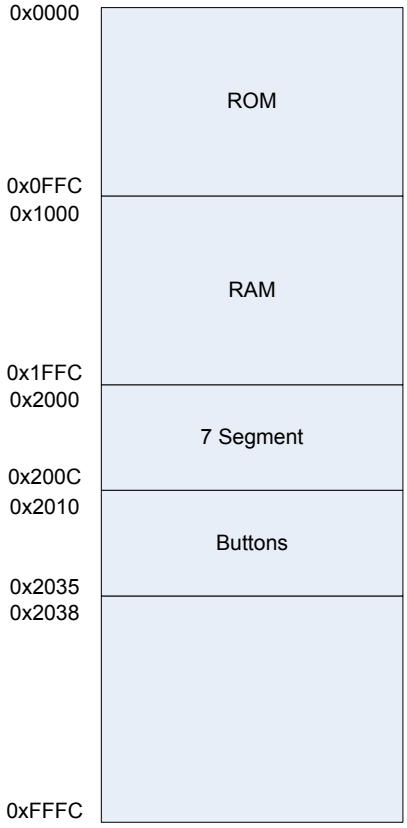

Figure 3: Address Space Allocation

5 RAM and ROM Description

The synchronous RAM and ROM have a size of 4KB. These memories should be word aligned, the 2 least significant bits of the address are ignored. Note that: There is a tri-state buffer on their rddata output. The FPGA has several synchronous memory blocks (SRAM) that we will use for the RAM and ROM.

You will use Xilinx's Core Generator Program to implement the ROM IP core. The ROM unit which you will design uses this IP core as a component. You will initialize the IP core with an external file. The details how to use the Core Generator are described below step by step.

5.1 SRAM Read Process

The read process of an SRAM is synchronous and has a latency of one cycle. The following timing diagram illustrates a typical read process in the SRAM. During cycle 0, a valid address is provided and read is set to 1.

The Decoder raises the cs signal that allows the SRAM to send the read data on the bus during the next cycle. At the rising edge of the clock, the address and the read and cs states are saved in registers by the SRAM module. During cycle 1, the registered address selects a line in the memory that is then sent to the rddata output. Another reading process may start during cycle 1.

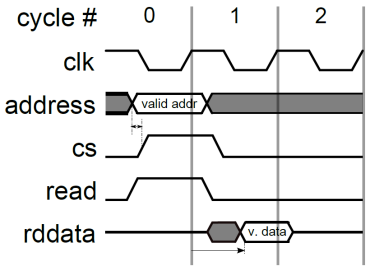

Figure 4: Timing diagram of the SRAM read process

5.2 SRAM Write Process

The write process of this system is synchronous and has no cycle latency. The following diagram illustrates a typical write process in the SRAM. It's very similar to a write in the Register File. The address, wrdata and write signals are set during the cycle 0. If the address matches the address space of the module, the Decoder will raise the cs signal. At the rising edge of the clock signal, the data will be saved by the SRAM. A new writing process can start during cycle 1.

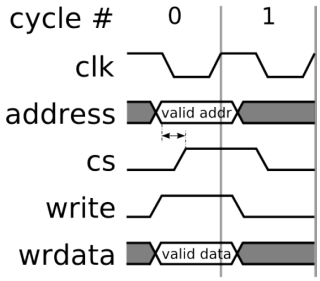

Figure 5: Timing diagram of the SRAM read process

6 Seven Segment Description

The Seven Segment module allows us to display something on the display of the FPGA Board. The dataout output will be connected to the Seven Segment Display pinouts. The Display is word aligned; therefore the 2 least significant bits of the address are ignored. Writing to the display will modify the value of an internal register connected to the dataout output. It can be considered as a very small memory of 1 word, whose content is displayed on the display of the FPGA board. Since we have 4 seven segment display, the display shows only one word at a time. In here address is not necessary since we can display only one word. However, If we can have several display or leds to show register content, then we should use an address to select which one is displayed. The address selects the register to display.

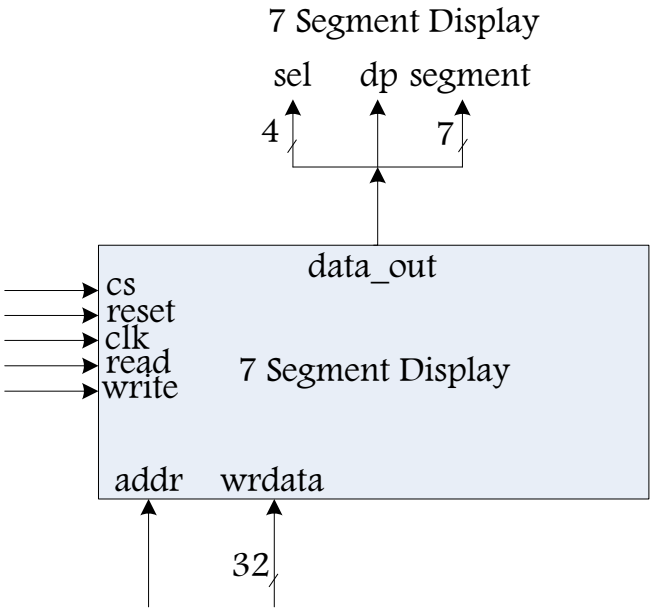

Figure 6: Seven Segment Block Diagram

7 Buttons Description

The buttons module allows us to load a value into a Memory or Display using same data bus. The data_in input is connected to the buttons which is placed on the board. Reading from the buttons will modify the value of an internal register connected to the data_in input. We can change the memory contents or display something on 7-segment with this module. The reading process is the same as the memories (1 cycle latency); this ensures compatibility among the different components connected to the same bus. A read will return the current value that is read from the internal register. Like the other module we need a tri-state buffer on the rddata output.

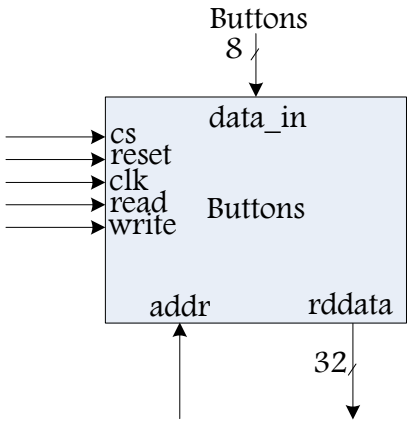

Figure 7:Buttons module block diagram

8 Synchronous Memories

In this section, you will find how to connect the components that are defined previously to get a complete memory system. The following figure illustrates this memory system.

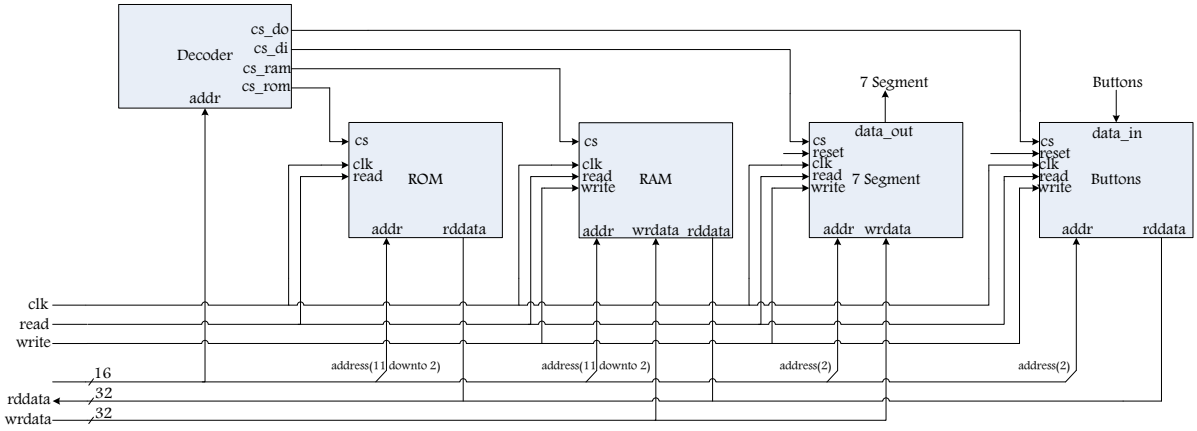

Figure 8: Connection scheme for Memory System

9 Procedure

In this lab, firstly you will design a Register File that is required for CPU in the next lab. After the implementation of Register File, you will design a memory system which has different memory components: RAM and ROM. You will connect these components together to obtain a complete memory system by the Decoder unit. In the system, There are also data input and output modules.

Register File:

- Implement the Register File described above. For that, you can use an array of std_logic_vector. The following code gives you an example of an array declaration.

  ```
  architecture synth of register_file is
  type reg_type is array(0 to 31) of std_logic_vector(31 downto 0);
  signal reg: reg_type;
  ```

  To read a value in an array, specify the index between parentheses exactly as you do when you select one bit in a vector. The index cannot be defined by a std_logic_vector; instead, use an integer. You may need the conv_integer() function that converts a std_logic_vector to an integer:

  ```
  data <= reg(conv_integer(address));
  ```

- Write the VHDL description of the Register File.
- Simulate it and show that the component properly works with screenshot from your simulation results.

Decoder:

- Write the VHDL description of the decoder.
- Implement it and simulate it.

Synchronous RAM:

You must implement a 4KB RAM.

- Create a new VHDL file to implement the RAM described previously.

Synchronous ROM:

For the ROM we will use Xilinx's Memory Block Generation Tool.

To create a ROM, you will follow the steps below.

- On a Windows system:

  Select Start > Programs > Xilinx ISE > Accessories > CORE Generator.

  Select Start > Run in Windows. At the command prompt, type coregen.
- Select File > New Project.
- In the New Project dialog box, do the following:
  - Enter a name and location for the CORE Generator project or click the Browse button to navigate to the project directory using the Browse for Folder dialog box.
  - Click OK.

    The Project Options dialog box opens.
- When the Project Options dialog box appears as part of creating the new project, select the following in the Part tab of the dialog box:
  - Select Family as Spartan 3.
  - Select Device as XC3S200 FPGA.
  - Select Package as FT256.
- Select the following in the Generation tab of the dialog box:
  - In the Flow section, select Design Entry and VHDL.
  - In the Flow Settings section, select the ISE Vendor.
  - Do not change the options in the Preferred Implementation Files and Simulation Files sections for this project.
- After options are selected, Select OK to create a project.
- In the IP selection window, Double-click the

  Memories & Storage Elements > RAMs & ROMs > Block Memory Generator.
- When the core customization GUI for the IP you selected appears, set following customization options for the IP core.
  - Enter component name as blk_mem_gen_v2_8.
  - Select Memory type as a Single Port ROM.
  - Select Algorithm as Minimum Area.

- Click NEXT to select other options.

  Memory Size Options

  - Write down 32 for Read Width.
  - Write down 1024 for Read Depth.
- Click NEXT to select other options.

  Memory Initialization
- Select Load Init File box and Browse the Coe file (An ASCII input coefficient file) which initialize your ROM.

Note that: You must create a Coe File for your ROM with Memory Editor Tool. You can access this tool from the Tools menu of Core Generator Program. The details are described below.

- Click Finish to build the customized IP core.

Now, you have a ROM memory. You should instantiate this IP core in your Memory System.

You should follow the steps below to learn how to instantiate the ROM in your design.

- You will add your ROM IP core whose name is blk_mem_gen_v2_8 as a component into your ROM top module.
- Your ROM module uses this IP core and read from this core with the rules defined previously in SRAM read process section.

Now, you will create a Coe File to initialize your ROM IP core with Memory Editor Tool.

- The Memory Editor tool is accessed by selecting Tools > Memory Editor from the CORE Generator menu bar.
- Memory Editor is opened two dialog box.
- Click Add Block and enter a name to the block in the memory editor dialog box.
  - Write down 1024 for Block Depth.
  - Write down 32 for Data Width.
- In the Memory contents dialog box, you can configure the contents of your ROM depending on the address.

- After completing the configuration,

  Select File > Generate

  Select COE file, browse your project directory for the ROM and click OK.

  The Coe File is now generated. You should use this file where you will initialize your IP core (ROM).

Seven Segments:

- You will use the seven_four.vhd as a component in this module.
- Write the VHDL description of the Seven Segments.
- Implement it and simulate it.

Buttons:

- Write the VHDL description of the buttons.
- Implement it and simulate it.

After completed the modules; RAM, ROM, Decoder, Seven Segments and Buttons. You will connect them to make a memory system according to the rules which is given on the Synchronous Memories Section.

- You should simulate your completed design using ISE or another VHDL simulator to prove the correctness of your design. Prepare a short report with the VHDL codes and the simulation results.
- **Bonus Grades** are available for those who make HW implementation of the memory Unit.

  For that, you can design a controller which assigns values for the address, rddata and wrdata signals in memory system. For example the controller which reads an address from the ROM. According to the value, the RAM content is displayed on the 7-segment. After that, the data on the buttons are read and displayed on the 7-segment. The controller produces necessary signals for the memory units. You will add this controller to the memory system you implemented. This master unit will copy data from one location on the ROM into another in the RAM.