Multi-Output Circuits: Memories

Introduction

In this lab you will design read only memories. *Please refer to the Vivado tutorial on how to use the Vivado tool for creating projects and verifying digital circuits*.

Objectives

After completing this lab, you will be able to:

- Use read only memories using **`reg`** data type and **`$readmemb`** system task available in Verilog

Read-Only Memories

Read-only memories (ROM) consist of interconnected arrays to store an array of binary information. Once the binary information is stored it can be read any time but cannot be altered. Large ROMs are typically used to store programs and/or data which will not change by the other circuitry in the system. Small ROMs can be used to implement combinatorial circuits. A ROM uses a decoder, similar to one designed in 1-1 earlier, to address a particular location.

A ROM will have m address input pins and n information output pins to store 2^m words information, each word being n bit in length. The content is accessed by placing an address and the content of the corresponding word is read at the output pins.

In Verilog HDL, memories can be defined as a two dimensional array using **`reg`** data type, as illustrated below:

```
reg [3:0] MY_ROM [15:0];
```

where **`reg`** is data type, MY_ROM is a 16x4 memory with 16 locations each location being 4-bit wide. If the memory is to be modeled as read only then two things must happen: (i) memory should only be read and not written into, and (ii) memory should somehow be initialized with the desired content. Verilog HDL provides a system task, called **`$readmemb`**, to initialize memory with a content. Following is an example of definition and usage of 4x2 ROM.

```
module ROM_4x2 (ROM_data, ROM_addr);
  output [1:0] ROM_data;
  input [1:0] ROM_addr;

  reg [1:0] ROM [3:0];  // defining 4x2 ROM

  assign ROM_data = ROM[ROM_addr];  // reading ROM content at the address
ROM_addr

  initial $readmemb (“ROM_data.txt”, ROM, 0, 3);   // load ROM content from
ROM_data.txt file
endmodule
```

The ROM_data.txt file, for this example, should be present in the same directory where the model is defined (since no directory path is given), and may have 8 or less lines such as:

```
10
0x
11
00
```

Note that if the number of lines is less than the size of the ROM, the unspecified locations will be initialized with 0s. Also, note that there is another system task available, called $readmembh, which allows the data file to be written using hexadecimal symbols.

3-1. Design a 2-bit comparator that compares two 2-bit numbers and asserts outputs indicating whether the decimal equivalent of word A is less than, greater than, or equal to that of word B. You will model ROM and use $readmemb task.

3-1-1. Open Vivado and create a blank project called lab3_3_1**.**

3-1-2. Create and add the Verilog module with two inputs (*a, b*) and three outputs (*lt***,** *gt***,** and *eq*) using ROM and $readmemb system task.

3-1-3. Create and add the XDC file, assigning *a* to ***SW3 to SW2***, *b* to ***SW1 to SW0***, *lt* to ***LED2***, *gt* to ***LED1*** and *eq* to ***LED0****.*

3-1-4. Create and add a text file that describes design output.

3-1-5. Synthesize and implement the design.

3-1-6. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

3-2. Implement 2-bit by 2-bit multiplier using a ROM. Output the product in binary on four LEDs.

3-2-1. Open Vivado and create a blank project called lab3_3_2**.**

3-2-2. Create and add the Verilog module with two 2-bit inputs (*a, b*), a 4-bit *product* output using ROM and $readmemb system task.

3-2-3. Create and add the XDC file, assigning ***a*** to ***SW3-SW2***, *b* to ***SW1-SW0***, and *product* to ***LED3-LED0****.*

3-2-4. Create and add a text file that describes the design output.

3-2-5. Synthesize and implement the design.

3-2-6. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

Conclusion

In this lab, you learned how to model multiple output circuits ROM. You also learned how to use a system task $readmemb to initialize ROM memory.

3-2. Implement 2-bit by 2-bit multiplier using a ROM. Output the product in binary on four LEDs.

3-2-1. Open Vivado and create a blank project called lab3_3_2**.**

3-2-2. Create and add the Verilog module with two 2-bit inputs (*a, b*), a 4-bit *product* output using ROM and $readmemb system task.

3-2-3. Create and add the XDC file, assigning ***a*** to ***SW3-SW2***, *b* to ***SW1-SW0***, and *product* to ***LED3-LED0***.

3-2-4. Create and add a text file that describes the design output.

3-2-5. Synthesize and implement the design.

3-2-6. Generate the bitstream, download it into the Nexys4 board, and verify the functionality.

Conclusion

In this lab, you learned how to model multiple output circuits such as decoders, encoders, and ROM. You also learned how to use a system task $readmemb to initialize ROM memory. There are more system tasks which the language supports and you will learn some of them in the next lab.