Lab 8 – Exploring the Operation of a 2-Read, 1-Write Memory for use in a Basic Processor

Introduction

As described in the class, a 2-read, 1-write random access memory is the basic building block for the registers inside microprocessors. In this lab you will be instantiating a small 8-word deep by 4-bit (0-F) wide memory.

Most standard memories are single port, i.e., they have a single address bus that selects the input data path and the output data path. The next level is dual port, 1-read, 1-write memories. These have independent address and control signals for the read side and the write side of the memory cells. In this lab you will be exploring the operation of a 2-read, 1-write memory device. It has three separate address buses to access the same internal memory cells: one for writing, and two for reading. Thus, one can read two different words or memory locations simultaneously. Here we separate memory locations into two memory banks (one reads system memory and other one reads user memory).

In the implementation, you will be controlling the read and write operations as well as displaying the memory contents by selecting various combinations of switches and push buttons on your DE10-Lite board.

Detailed Specification:

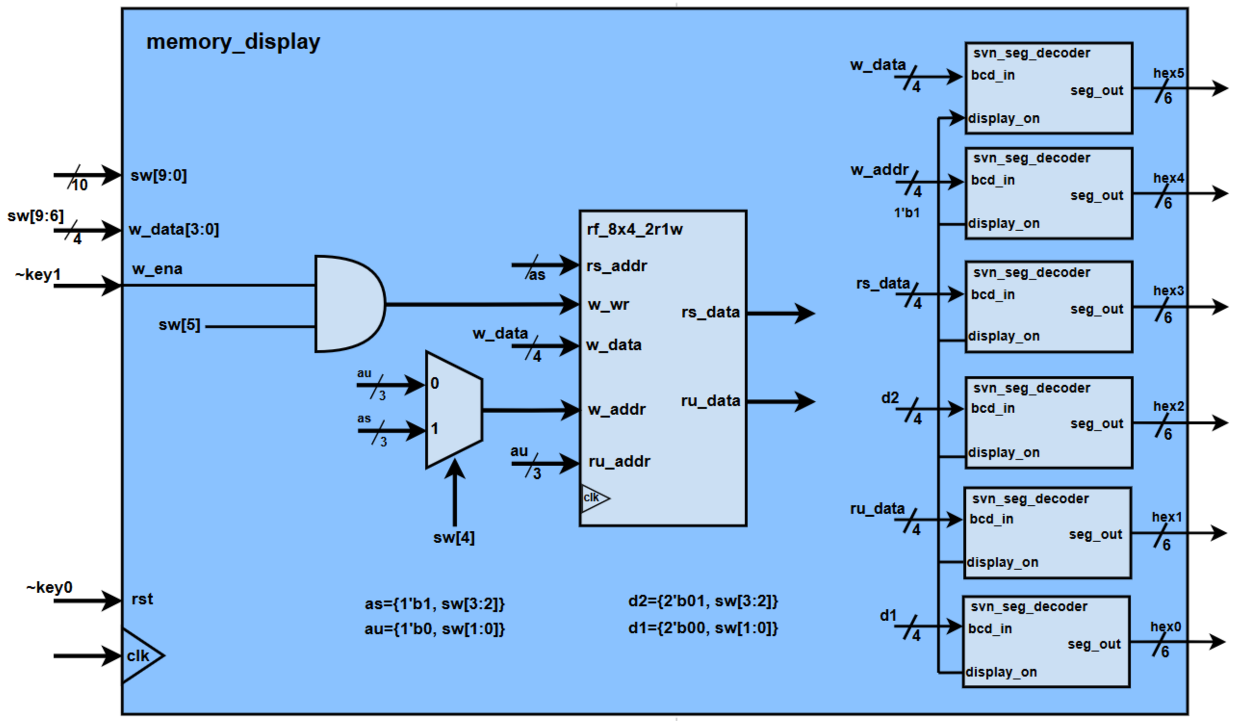
A block diagram for the top module, memory_display, is illustrated below. Note there are 6 input signals/busses:

a) the clock (clk),
b) the reset signal (rst) (connected to the pushbutton key0),
c) switches (sw[4:0] where s[4:2] is system memory addresses when s[4] is high, and {s[4], s[1:0]} is user addresses when s[4] is low. i.e. memory locations 0-3 is for user memory and 4-7 is system memory),
d) a memory write signal (m_write) (connected to the pushbutton key1),
e) a write enable signal (w_ena) (connected to switch 5),
f) a data write bus (w_data) (connected to switches 6-9).

There are five output signals:

a) anode and cathode to drive the seven segment displays,
b) rs_data and ru_data which are the two read ports from the memory.

From the block diagram, map out the functional operation of the design for the switch combinations. Compare to the functionality described in the Functional Table below it. This will provide you with an operating template when your design is implemented in hardware. Verify that it matches the block diagram. Note that the write functionality and the information on display digits 4 and 5 are determined by switch 5 (connected to w_ena).



Functional Table	
Function: reset	
reset	center pushbutton (rst input signal in memory_display)
Function: write data to memory	
Inputs:	
Select write function	key1 – on (w_ena input signal in memory_display) and switch 5 on
Write address	3-bits: switches [4,3,2] or switches [4,1,0]
Write data	4-bits: switches 9-6 (w_data input signal in memory_display)
Command: write data to address	Down key1 pushbutton (m_write input signal in memory_display)
Outputs:	
Digit 4	3-bits: hexadecimal, address to write to (switches 4,1,0)
Digit 5	4-bits: hexadecimal
Function: read data from memory	
Inputs:	
Select read function	key1 – off (w_ena input signal in memory_display)
Read address, port u	3-bits: switches 4,1,0
Read address, port s	3-bits: switches 4,3,2
Outputs:	
Digit 0	3-bits: hexadecimal, address user memory
Digit 1	4-bits: hexadecimal, data read from address user memory
Digit 2	3-bits: hexadecimal, address system memory
Digit 3	4-bits: hexadecimal, data read from address system memory

Design

You will be using modules/files from your past labs/designs: display_driver, svn_seg_decoder, and anode_decoder. These will be linked automatically for you in the simulation and synthesis command scripts.

You will be provided the 2-read, 1-write memory module design: rf_8x4_2r1w.

A skeleton file is provided for the module shown in the block diagram: memory_display. This is the only module that you need to design. The design basically involves:

- instantiating the memory,
- instantiating the display driver,
- creating the AND gate and a multiplexer (I recommend an assign statement and a combinational always block),
- and connecting all the signals.

Simulation/Verification

In the simulation view, there is one self checking test bench wrapper for the above module: tb_memory_display.sv. This file will read the test vectors from tb_memory_display.txt. Neither of these two files should be edited.

Verification step:

□ memory_display.m_sim is used to simulate the memory_display module.

The test bench has 10 test vectors that transition based on a change of the anode signals. Each set of 4 vectors are monitoring the anode and cathode signals so as to monitor the values presented to each digit. The first 32 test vectors write to the 8-words of memory. In particular the address-data combinations are (in hexadecimal): 0-F, 1-D, 2-B, 3-9, 4-7, 5-5, 6-3, 7-1. The final 16 test vectors read the 8-words of memory, 2 words at a time.

The output from a successful simulation will be:

```
Match--loop index i:        0
Match--loop index i:        1
Match--loop index i:        2 …
Match--loop index i:        9
Simulation complete - no mismatches!!!
```

If you have mismatches, the expected anode and cathode values are displayed, compared to their simulated values. To determine the input switch settings you will have to observe the timing diagram at the appropriate value of the index i. An example mismatch message could be:

```
Match--loop index i:          17 Mismatch--loop index i:       18; anode, cathode
expected:
1011_1000000 0, received: 1011_0011001 4
Match--loop index i:          19
 anode                                        cathode    decoded cathode
```

In this example the decoded cathode is the digit that would be displayed on the seven segment decoder based on the cathode pattern.

Correct your mismatches before you implement the design in hardware.

Synthesis

Generate a bit file as you did in past labs (right click and Run on: lab8_top.qsf). For this lab it will be called: lab8_top.sof. Copy it to your sof file to load and run it on your DE10-Lite board.

Implementation

Using the Functional Table shown previously, select the switches and push buttons to enable the required operations. In particular, test your design by writing to the 8-words some random 4-bit numbers of your choosing, i.e., part of your TUID, part of your phone number, etc. Complete the table below with the data. Then select the read mode and see that you can read back all of the stored data – and notice how you can read two different locations simultaneously.

Memory Address	Data
0	
1	
2	
3	
4	
5	
6	
7	

The ability to read two different locations simultaneously in a single memory is a subtle but important concept to understand the low level details of how a basic processor works.