Lab 5

By

Don Dang

Brigid Kelly

May 30th, 2018

Microprocess Designs

Ken Rabold

**Section B: Table of Contents**

Section A : Title………………………………………………………………………………………………………..……1

Section B : Table of Contents…………..………………………………………………………………………......2

Section C : A brief description of the design for both modules….…………………………………..3

Section D : An explanation of why the Address port for the RAM module is only 30

bits wide and not 32 bits………………………………………………………………………………………………….4

Section E : Screenshots from ModelSim showing your RAM module and Register

Bank working within the provided test bench…………………….……………………………………………5

Section F : Lab5 link to GitHub…………………………………………………………………………………..…….10

**Section C : A brief description of the design for both modules**

Memory Module:

The memory module for this lab was a simple procedural implementation using an array of 32-bit standard logic vectors, the included for-loop to reset the module, followed by a series of if-statements.

Utilizing the ‘Write Enable’ signal on a sensitivity list, the module is then able to determine when to accept incoming data.   The address being sent into to the memory module for writing is then converted to an unsigned integer and, as long as it is between 0 and 127, is used as an index to determine the element of the signal array “I\_ram” which to write.  The array contained within that index is then assigned the value being passed by the ‘DataIn’ signal until it is overwritten by new data.

Reading the memory works in a similar way, but activates on an active low sensitivity list from the ‘OE’ or ‘Output Enable’ signal.  The address input into the memory module is then used to access the element in the array corresponding to the input value, and the data at that element is then relayed to the DataOut signal.  If a signal beyond the range of 0-127 is attempted to be read, it will return a signal of high impedance, or ‘high Z’.

Register Bank:

           The register bank operates in a similar fashion to the memory module, but the implementation differs due to the use of concurrent components instead of a sequential process.  These concurrent components include the 8 and 32-bit registers created in previous labs, as well as the 1 bit flip-flop initially provided.

 For the design to be implemented, a 32-bit signal was created as an input line for each of the 8 registers needed to be included and the zero register.  Using the ‘Write Register’ signal, combined with the 5-bit binary address of the register within the bank, a de-multiplexer was created to determine which input signal should receive the ‘Write Data’ input.

For reading data, each register has a 32-bit output line connected to two multiplexers, which are implemented using two simple With-Select statements.  The ‘DataOut1’ and ‘DataOut2’ signals are assigned an output line based on the address selected by the ‘Read Regsiter’ signals (readreg1 and readreg2), which correspond to an address of one of the registers in the bank.

A port map for each register in the bank was built, in which the 32-bit read signal is hardwired to 1 and others are disabled, the 32-bit write command is mapped to the ‘WriteCMD’ signal with others disabled, and each 32-bit input/output signal  is mapped to its corresponding register.

**Section D : An explanation of why the Address port for the RAM module is only 30 bits wide and not 32 bits**

Because the memory module we have designed is only whole-word accessible (32-bits at a time) we are only required to have 2^30 addresses to access the same amount of data at an 8-bit addressable memory module. Since, in a normal (32-bit byte-addressable) memory module, each 32-bit word is stored across 4 addresses, it requires 4 times the amount of addresses as ours, which stores each 32-bit word in a single address. Essentially, the memory addressing method for our module does not require the accessing of the intermediary bytes, but it will still contain the same amount of data *and* addresses as its 32-bit addressable counterpart.

**Section E : Screenshots from ModelSim showing your RAM module and Register Bank working within the provided test bench**

**-- Wait 100 ns for global reset to finish**

wait for 100 ns;

-- Test RAM by filling in some values

reset <= '1'; -- Reset RAM module

wait for 5 ns;

reset <= '0';

wait for 5 ns;

oe <= '0';

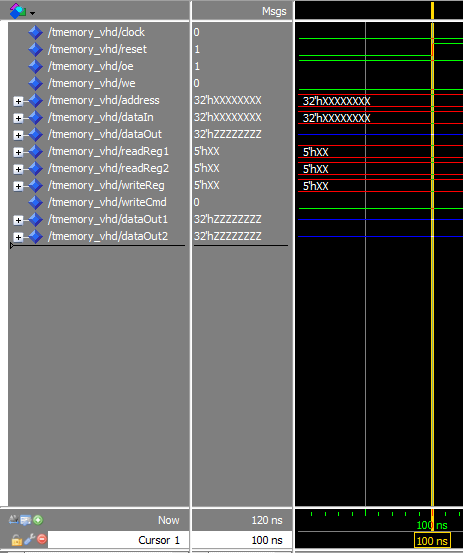


Figure 1 - First 100 ns

--Set dataIn = X"11111111

-- address = X"00000000

address <= X"00000000";

dataIn <= X"11111111";

we <= '1';

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

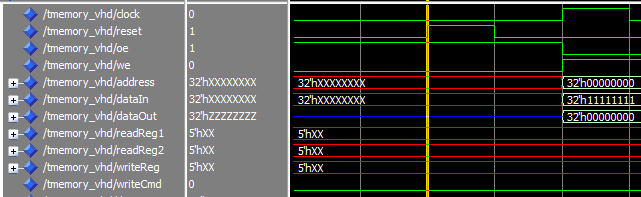


Figure 2 – 110 ns

-- Set dataIn = X"22222222

-- address = x"00000004

address <= X"00000004";

dataIn <= X"22222222";

we <= '1';

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

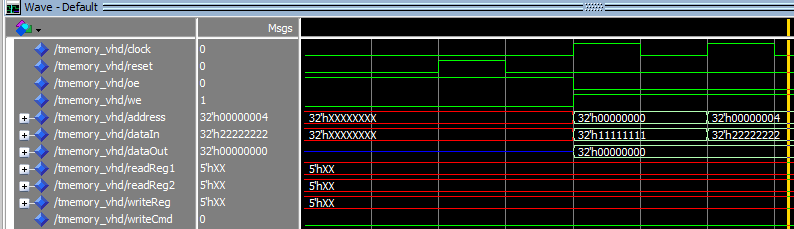


Figure 3 – 125 ns

**-- Continue writing data to the memory. Increment the memory address by 4.**

address <= X"00000008";

dataIn <= X"33333333";

we <= '1';

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

address <= X"0000000C";

dataIn <= X"44444444";

we <= '1';

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

address <= X"00000010";

dataIn <= X"55555555";

we <= '1';

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

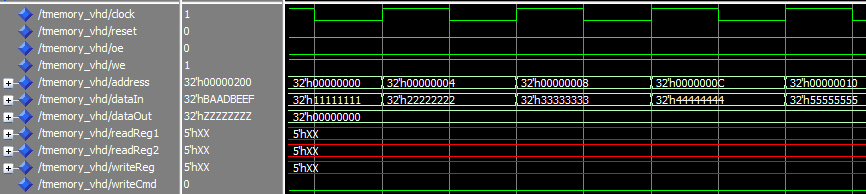


Figure 4 – 155 ns

**-- DataOut is high Z. The address is beyond 127 bits.**

address <= X"00000200";

dataIn <= X"BAADBEEF";

we <= '1';

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

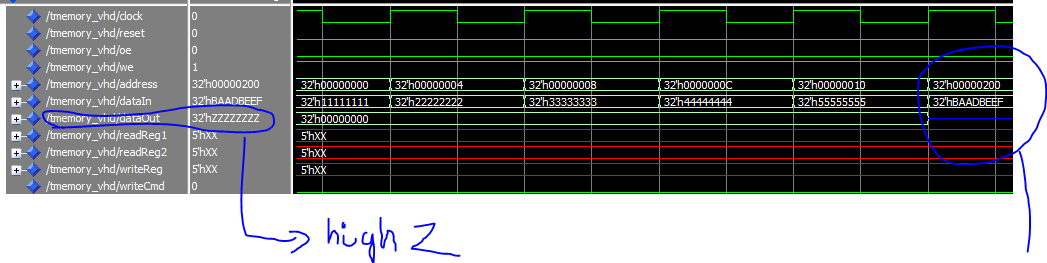


Figure 5 – Dataout high Z. Memory address is beyond 127 bits.

**--Reading back contents from the memory**

oe <= '0';

we <= '0';

address <= X"00000000";

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

address <= X"00000010";

clock <= '1';

wait for 5 ns;

clock <= '0';

wait for 5 ns;

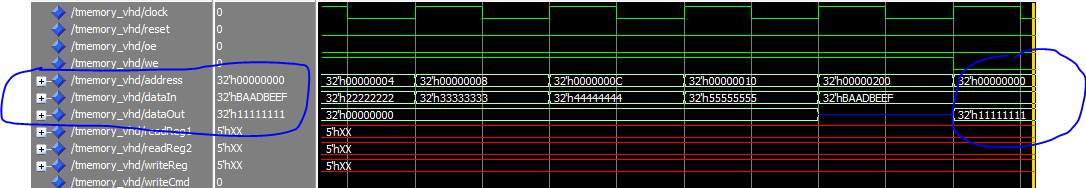


Figure 6 – Retrieve value from address X”00000000”. Datout = X”1111111”

**--Writing to the registers at specific address and values**

dataIn <= X"11111111";

writeReg <= "00001";

writeCmd <= '1';

wait for 5 ns;

writeCmd <= '0';

wait for 5 ns;

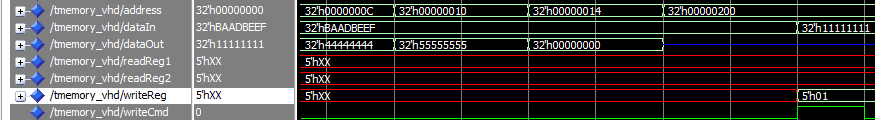


Figure 7 - Writing to the registers

**--Reading from the registers.**

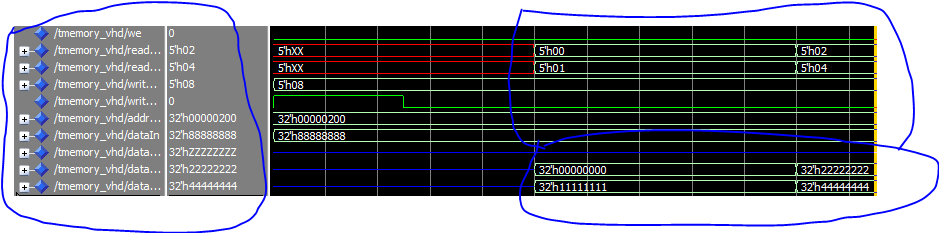


Figure 8 – Reading from the registers

**Section G: Lab5 link to GitHub**

<https://github.com/donjuanwu/Labs/tree/master/Lab%205>