COE3DQ5 Lab #4
Embedded Memories and an External SRAM Interface

Objective

To understand how to build and simulate a digital system containing embedded memory blocks, such as read-only memory (ROM), single-port random access memory (RAM), and dual-port RAM. To gain experience with design verification using SystemVerilog testbenches and the ModelSim simulation environment. To design and implement a self-test engine for an external static RAM (SRAM) memory.

Preparation

- Revise the first three labs, especially finite state machines (FSMs) and PS/2 and VGA interfaces
- Read this document and get familiarized with the source code and the in-lab experiments

Experiment 1

The objective of this experiment is to get you familiarized with SystemVerilog for design verification.

Hardware description languages (HDLs), e.g., VHDL, Verilog, SystemVerilog, are employed for both describing hardware and verifying it. As shown in Figure 1, the aim of verification is to control the inputs of a design (using **drivers**) and to observe its internal states and outputs (using **monitors**) in order to ensure the correct behavior before the design is implemented. There are (System)Verilog language constructs which (as shown in the ***experiment1*** design) are used exclusively for verification. Note, most of these language constructs do not have an equivalent hardware circuit, as they are used primarily to drive and monitor the design behavior through simulation. They are used in special verification modules called “testbenches”. Verifying the design through simulation can be done at different levels of design abstraction. The (System)Verilog source code can be simulated “fast” by exploiting the event structures and the sensitivity lists (part of the language). This is called **behavioral** simulation and it is commonly at least one order of magnitude faster in terms of runtime than **timing** simulation. Timing simulation works on the implemented circuit and it uses accurate timing information, which is back-annotated from the technology library. A properly designed testbench will ensure that the design will work correctly as soon as it is implemented, e.g., programmed into a field-programmable gate array (FPGA) device.

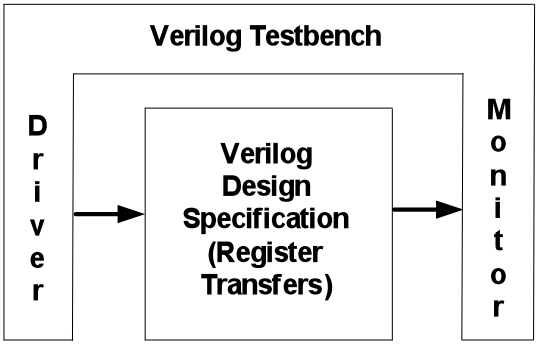


Figure 1 – Design verification using testbenches

You have to perform the following tasks in the lab for this experiment:

- understand the design verification methodology using SystemVerilog testbenches and ModelSim
- do behavioral and timing simulation for video signals for 1.5 ms; understand the runtime implications
- do behavioral simulation for one frame of video (in the 640x480 @ 60 frames per second VGA mode)

Experiment 2

The aim of this experiment is to understand the functionality of embedded RAM blocks.

Programmable logic devices contain embedded memory blocks that can be configured as ROMs or single-port RAMs or dual-port RAMs (or DP-RAMs). They are necessary for avoiding storing large amounts of state information in on-chip registers. They are suitable for implementing the first level of caches in memory systems. The difference between having a RAM on-chip vs. off-chip lies in the fast memory access time on-chip which can be in the range of a few of ns for high-speed devices. Besides, having multiple memory blocks facilitates concurrent access of data in all the embedded blocks, thus avoiding accessing a shared resource off-chip (this avoids memory access bottlenecks that can slow down the performance of computer systems). As shown in Figure 2, for single port RAMs the write enable signal (wren) decides whether the memory cycle is a READ or a WRITE. For the given RAM configuration, after the address is applied to the inputs, the output data will be available after the edge of the clock cycle. If wren is high then the data available on the input port will be written to the memory location determined by the address bits on the next edge of the clock cycle. For dual-port RAMs the two ports can read and write data concurrently, so long as the addresses are different. If a write to the same address is attempted on both ports a hazard will occur, which may push the system into meta-stability.

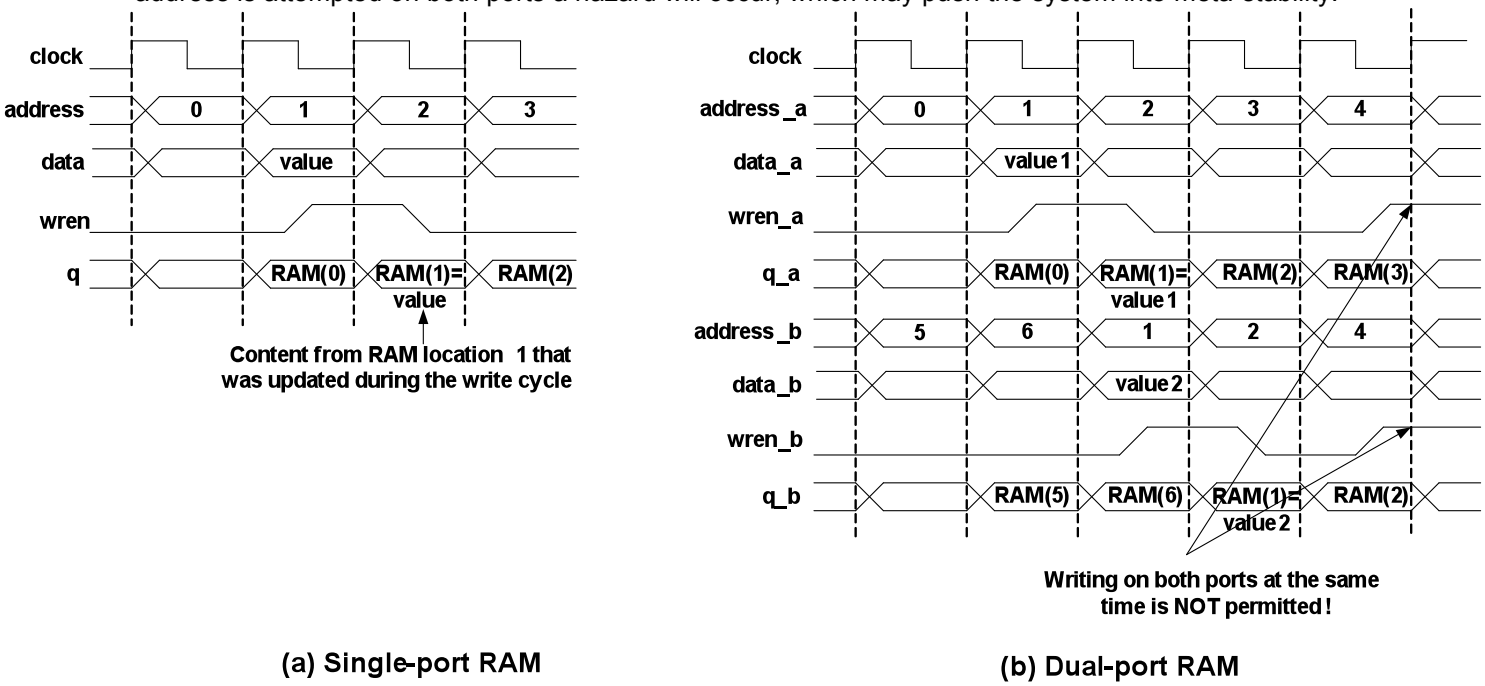


Figure 2 – The timing behavior of single-port and dual-port RAMs

To better understand the benefits of DP-RAMs this experiment shows how values from two different embedded RAM blocks can be read, added and subtracted and subsequently these results are stored back to the embedded RAMs in the same location from where they have been read. In Figure 3 an implementation with 2 single-port RAMs is given (***experiment2a***). The single-port RAMs share the same address register and write enable. The data is first read from a location determined by the address register and in the following clock cycle the address register is kept the same while the addition and the subtraction results are written back to the embedded RAMs by activating the write enable signal. This computation can be accelerated by using 2 dual-port RAMs (***experiment2b*** shown in Figure 4) of the same capacity. There will be two distinct address registers (one for reading and one for writing). Unlike the case where the single-port RAM is employed, the read address register is incremented every clock cycle. This register will be hooked up to a read port while the write address register (delayed by one clock cycle) is connected to a write port. Because (by design construction) the read address and write address registers will not conflict, a new result can be written back to the memory every single clock cycle.

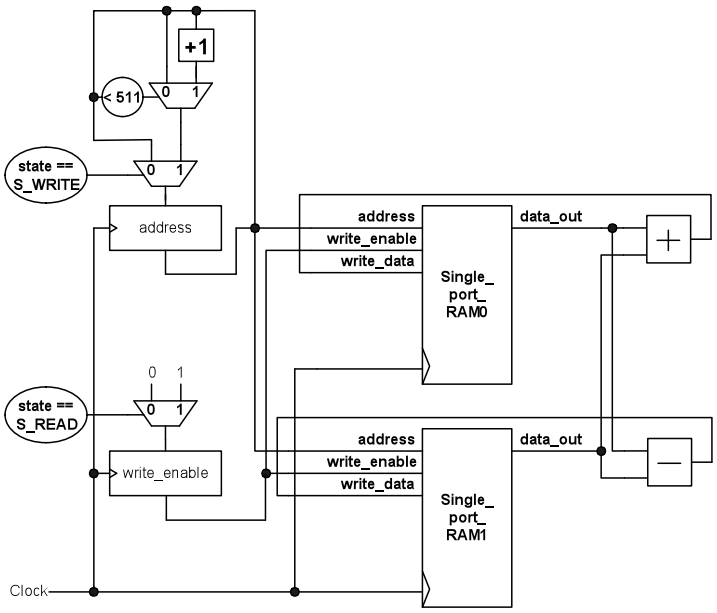


Figure 3 – Circuit from experiment2 implemented using 2 single-port RAMs

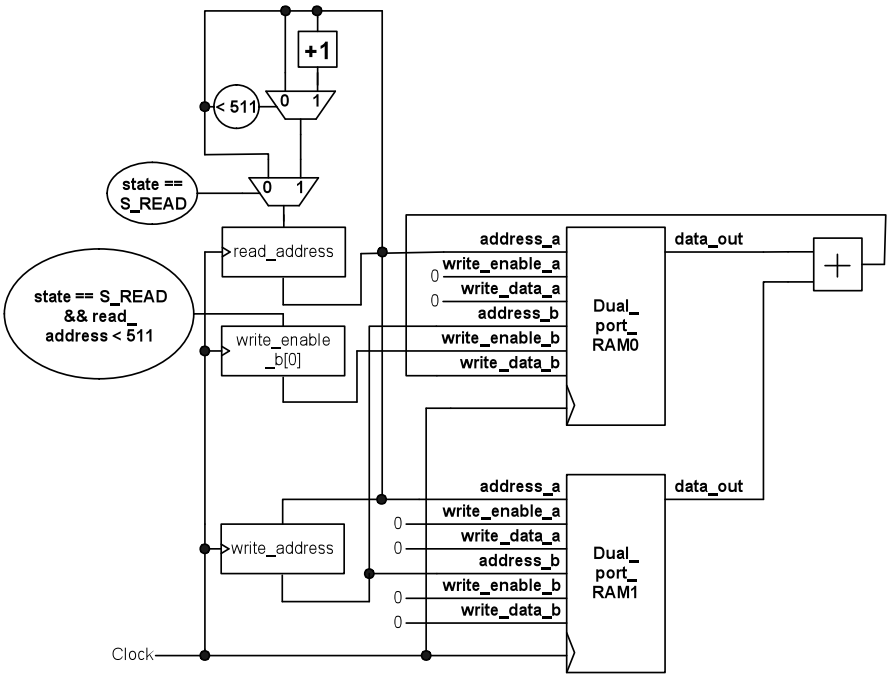


Figure 4 – Circuit from experiment2 implemented using 2 dual-port RAMs (to be completed)

You have to perform the following tasks in the lab for this experiment:

- simulate and understand why the design from ***experiment2a*** (based on single-port RAMs) takes more clock cycles to execute than the design from ***experiment2b*** (based on dual-port RAMs); note the memory initialization files are given in "HEX" format due to compatibility with ModelSim
- in the code from folder ***experiment2b*** the addition of the values read from the two RAMs is stored in the top RAM; extend the dual-port implementation (shown in Figure 4) in such way that the subtraction of the values read from the two RAMs is stored in the bottom RAM

Experiment 3

In this experiment you will learn how to use single and dual-port RAMs with the PS/2 and VGA interfaces.

In ***experiment3a*** shown in Figure 5 the PS/2 controller explored in the second lab is put together with the VGA controller from the third lab in order to create a small editor. Note, although we work in the VGA mode (640x480 @ 60 frames per second), the VGA controller has been re-designed such that ***all*** the registers and the embedded memories in the design are clocked at 50 MHz. The VGA controller is clocked at 50 MHz, however internally it uses one register that enables the update of its reference counters every other clock cycle. Therefore, pixel_X_pos and pixel_Y_pos are updated every second clock cycle at 50 MHz. The viewing area (or display area) is set in the middle of the screen in a 320x240 box. The design requires a ROM that converts the PS/2 make codes to the character ROM codes. A RAM is used to store the address of each character that is displayed in the viewing area. When the pixel_X_pos and pixel_Y_pos are in the viewing area, the character address will be looked up from the single-port RAM and it will be passed to the character ROM that will in turn drive the RGB signals. Since each character occupies an 8x8 square, there will be a total of (320/8)x(240/8) = (40 characters/line) x (30 lines) = 1,200 characters that can be displayed. Therefore, although the RAM is created with 2,048 locations, the design is implemented in such way that if the first 1,200 locations of the RAM are filled no further updates will be made to the RAM. If the user wishes to delete the viewing area he needs to use a "clean" mode where after the push button 0 is pressed the content of the character RAM is erased.

Using two switches you can explore the limitations of the single-port RAM for capturing the keys pressed on the PS/2 keyboard. If SWITCH_I[1] is set and we are not in the "clean" mode, the PS2_write_enable (determined by the values of the PS2_code_ready and PS2_make_code flags) will be set and the RAM address will be driven by the PS2_data_counter in order to store the new character address. If PS2_write_enable is asserted while pixel_X_pos and pixel_Y_pos are in the viewing area, the ROM output will be intermittently (and incorrectly) changed to the value which was being written, thus leading to flickering on the screen. This problem can be fixed by disabling any memory writes while pixel_X_pos and pixel_Y_pos are in the active period of V_SYNC or when they are in the display area on the screen (controlled by SWITCH_I[0]). Note, however in this case some of the typed keys will never be stored.

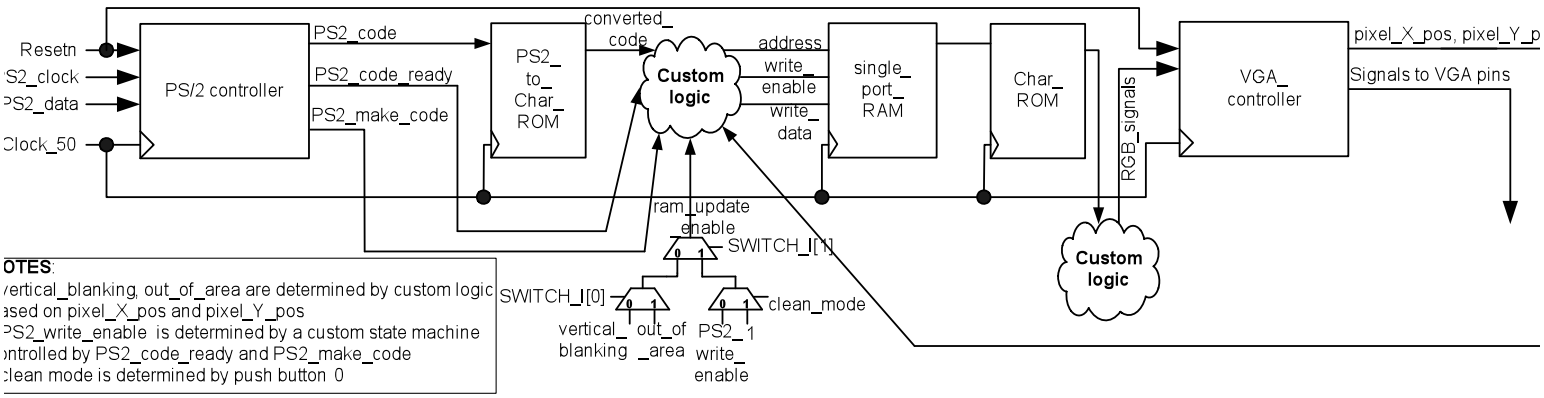


Figure 5 – Circuit from experiment3 implemented using a single-port RAM

Although there is no need to write any source code for this experiment, you have to perform the following tasks in the lab:

- understand why using a single-port RAM in ***experiment3a*** there are problems when capturing keys pressed on the PS/2 keyboard
- understand why using a dual-port RAM in ***experiment3b*** the limitations of ***experiment3a*** are removed

Experiment 4

The objective of this experiment is to introduce the external SRAM and to show how it can be modeled.

In this experiment the external SRAM device on the DE2 board is introduced. It has 18 address lines and 16 data lines (organization of 256k locations with 16 bits per location). Its capacity is approximately 10 times the on-chip memory available on the CycloneII device. The SRAM controller implemented for this external device has the memory access cycles shown in Figure 6. Note, the latency is 2 clock cycles.

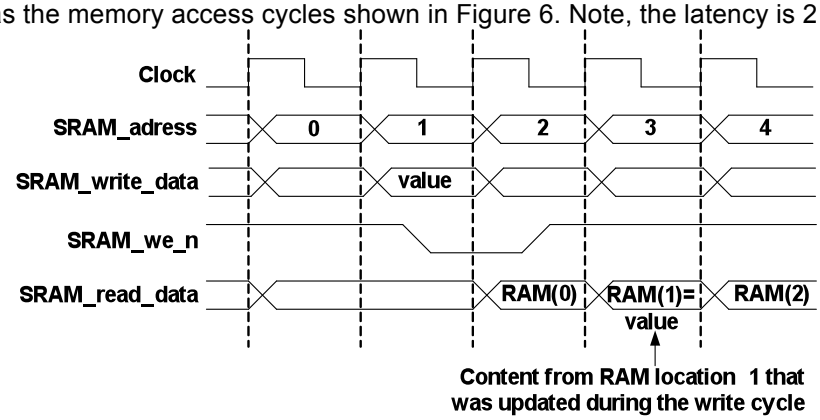
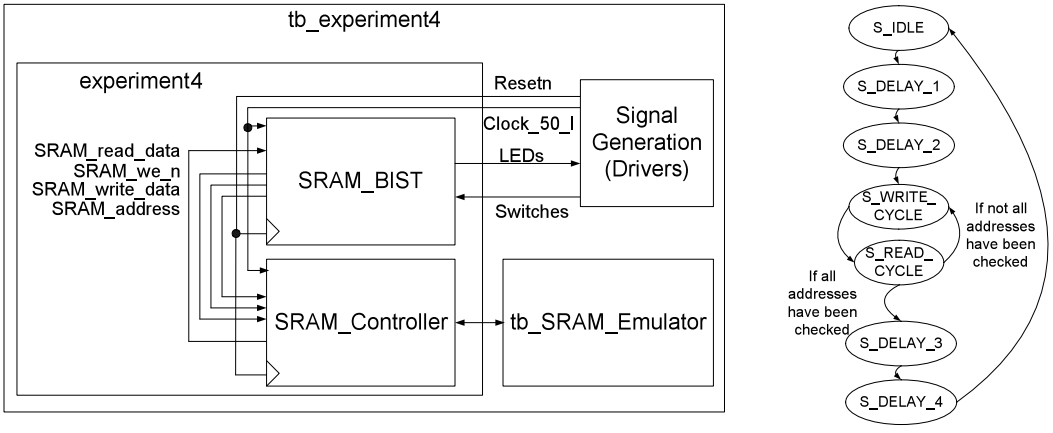


Figure 6 – Memory accesses to the external SRAM

In the ***experiment4*** folder, a simple built-in self-test (BIST) engine is designed for the SRAM device. BIST is an enabling technology that is essential for characterization, test and diagnosis of high-density memory devices. The circuit architecture and its testbench are shown in Figure 7(a), while the state machine for the BIST engine is given in Figure 7(b). The BIST engine employs a counter that iterates through the entire memory space of the SRAM and it reads back the value that has just been written. In the case of a mismatch an error flag will be set. The key point of this experiment is the use of a SRAM emulator for simulation purposes. Because the SRAM is an external reactive device (i.e., it responds to the requests given by our circuit) it is essential that its behavior is accurately modeled in the verification testbenches.



(a) Design architecture and testbench (b) BIST state machine

Figure 7 – Circuit for experiment4 (including its testbench)

You have to perform the following tasks in the lab for this experiment:

- understand how an emulator module can be used for simulating an external device, such as SRAM
- in the given source code the memory BIST is done by reading back the value which has just been written in the SRAM; modify the code in such way that all the values are written in the SRAM in a burst mode (the data value equals the lowest 16 bits of the address); then all the values from the memory should be read back and compared against the expected values generated on-chip; prove that your design works first through simulation and then by programming the FPGA device

Write-up Template

COE3DQ5 – Lab #4 Report
Group Number
Student names and IDs
McMaster email addresses
Date

There are 2 take-home exercises that you have to complete within a week. When the source files are requested, submit the Verilog and "QSF" files. Where applicable, submit the "MIF" or "HEX" files used to initialize the memories and the ModelSim "do" files. Label the top-level modules as exercise1 and exercise2. If, for any particular reason, you will add/remove/change the signals in the port list from the port names used in the design files from the in-lab experiments, make sure that these changes are properly documented in the source code.

Exercise 1 (2 marks) – Modify the design from ***experiment2b*** as follows. In the first and the second DP-RAMs there are two different arrays (called A and B respectively) of 8-bit signed integers. Each of these two arrays has 512 elements (initialized the same way as in the lab, i.e., using memory initialization files). Design the circuit that computes two arrays C and D defined as follows (for every i from 0 to 511): if A[i] is negative then C[i] = A[i] + B[i], otherwise C[i] = A[i] - B[511-i]; if B[i] is negative then D[i] = B[i] - A[i], otherwise D[i] = B[i] + A[511-i]. Each element C[i] should overwrite the corresponding element A[i] in the first DP-RAM (for every i from 0 to 511); likewise, each element D[i] should overwrite the corresponding element B[i] in the second DP-RAM. As in the lab experiment, if the arithmetic overflow occurs as a direct consequence of the additions, it is not necessary to detect it (or take any action on it). The above should be implemented in as few clock cycles as it can be facilitated by the DP-RAMs. Verify the correctness of your design using simulation only.

Submit your sources and in your report write at most a half-a-page paragraph describing your reasoning.

Exercise 2 (2 marks) – Modify the burst-mode implementation of the BIST engine from ***experiment4*** as follows. To verify the 256k locations of the SRAM, 2 bursts of writes and reads will be performed. In the first burst, 128k consecutive locations will be verified by first writing the value of the 16 least significant bits of the address (as done in the lab). While writing the data during the first burst, the address lines must change in the *decreasing* order (i.e., from 128k-1 down to 0). Then, the same 128k consecutive locations will be read to verify their content. Note, however, when reading the data during the first burst the address lines must change in the *increasing* order (i.e. from 0 to 128k-1). After the first burst is done, the BIST engine will move on to perform the same action on the second burst whose starting address is 128k. For this burst, when writing the data in the SRAM the address lines must change in the *increasing* order (i.e., from 128k to 256k-1). Then, the same 128k consecutive locations will be read to verify their content. Note, however, when reading during the second burst the address lines must change in the *decreasing* order (i.e. from 256k-1 down to 128k). Verify the correctness of your design using simulation only.

Submit your sources and in your report write at most a half-a-page paragraph describing your reasoning.

VERY IMPORTANT NOTE:

This lab has a weight of 4% of your final grade. The report has no value without the source files, where requested. Your report must be in "pdf" format and together with the requested source files it should be included in a directory called "coe3dq5_group_xx_takehome4" (where xx is your group number). Upload this directory onto the SVN server (in your group directory as described in the SVN guide posted on the course website) before noon on the day you are scheduled for lab 5. Late submissions will be penalized.