Chapter 9

Asynchronous SRAM Memory Controller

The purpose of this laboratory is to learn how to use an asynchronous SRAM memory and create a synchronous memory interface for this device. One of the course learning outcomes for this course is the "ability to read and interpret a data sheet." You will achieve this learning outcome by reading and interpreting the data sheet for the memory on the Nexys2 board.

Learning Objectives

After completing this laboratory you should be able to:

- Read and interpret the memory data sheet

Exercises

Exercise 1 – Memory Data Sheet

The Nexys2 board that we are using in the lab contains one Micron 128Mb DRAM, MT45W8MW16BG (called a CellularRAM by Micron). This is a complex memory and provides many modes of operation. It is essential that you understand how this memory functions and how to perform asynchronous read and write operations. Obtain a copy of the datasheet and scan through the document. Keep this document handy as you will need to refer to it frequently during this lab.

Download: Micron 128Mb DRAM, MT45W8MW16BG Data Sheeet

Carefully read the overview pages 5-8 of the data sheet and answer the following questions:

Question: How many bits are available in a single device?

Question: How many bits are in each word of this device?

Question: What is the purpose of the Bus Configuration Register (BCR)?

Question: What is the purpose of the LB and UB signals?

The functional description and bus operating modes of this part are described in pages 10-16. For this lab, we will be using the "Asynchronous" read and write operations and the page mode read operation. Answer the following questions after reading these pages of the datasheet:

Question: How long does it take to complete the "Power-Up Initialization" of this device? How many 50 MHz clock cycles is this power-up initialization?

Question: What value should be applied on the CE# signal during power-up initialization?

Question: Summarize the difference between the Asynchronous Read Mode and the Page Mode Read operation.

Asynchronous Read Mode

The Asynchronous Read Mode is summarized on pages 10-11. Detailed timing diagrams of the asynchronous read mode (including the page read mode) are shown in Figures 30-32 of the data sheet. The timing requirements for the asynchronous read mode are listed in Table 14. You will need to refer to these figures and tables to answer the following questions about performing asynchronous read operations. For all your answers regarding timing, use the 70 ns speed grade numbers.

Question: Determine the values of CE, WE, OE, ADV, LB, and UB needed to perform a read in Asynchronous Mode on both the upper and lower bytes.

Question: What is the minimum read cycle time (T_{RC}) for the device used on our board?

Question: How many 50 MHz clock cycles will be needed to complete a read operation on the Nexys2 FPGA board?

Question: What is the purpose of the parameter T_{AA}? This parameter happens to be the same as T_{RC} and they have the same value. Why is T_{AA} in the "maximum" column and T_{RC} is in the "minimum" column?

Question: What is the difference between the timing parameter t_{OE} and the timing parameter t_{OLZ}?

Question: In page mode, determine the maximum time it will take for the data to be available once the address has changed (T_{AA} in Figure 7)?

Question: In page mode, determine the maximum time it will take for the second piece of data to be available once the address has changed (T_{APA} in Figure 7)?

Write Mode

Detailed timing diagrams of the asynchronous write mode are shown in Figures 39-42 of the data sheet. There are several different timing diagrams that indicate which input signal "controls" the write operation. The timing requirements for the asynchronous write mode are listed in Table 16. You will need to refer to these figures and tables to answer the following questions about performing asynchronous write operations.

Question: Determine the values of CE, WE, OE, ADV, LB, and UB needed to perform a write in Asynchronous Mode on both the upper and lower bytes.

Question: What is the difference between a WE# controlled write (page 53) and a CE# controlled write (page 51)?

Question: What is the minimum asynchronous write cycle time (T_{WC}) for the device used on our board?

Question: How many clock cycles should the WE# pulse be held low for a write operation (see t_{WP})?

Question: How long must the data be present on the memory data pins for a write (see t_{DW})?

Question: How long does the data need to be held after WE is de-asserted (see t_{DH})?

Exercise 2 – Memory Controller Architecture

We will use the "asynchronous" memory transfer modes. This means that there is no clock that synchronizes the operations of the memory with the internal FPGA circuit. Because the circuits you are creating in the FPGA are synchronous, you need to create a synchronous memory controller that allows us to safely use the memory with our synchronous system. The first design exercise is to create a synchronous memory controller for the memory.

The process for creating this memory controller is very similar to the approach described in Section 12.3 of the textbook. Reviewing Section 12.3 and the associated notes from the SRAM lecture will help you complete this lab. The diagram in Figure 9.1 provides an overview of the memory controller architecture. We will use the same

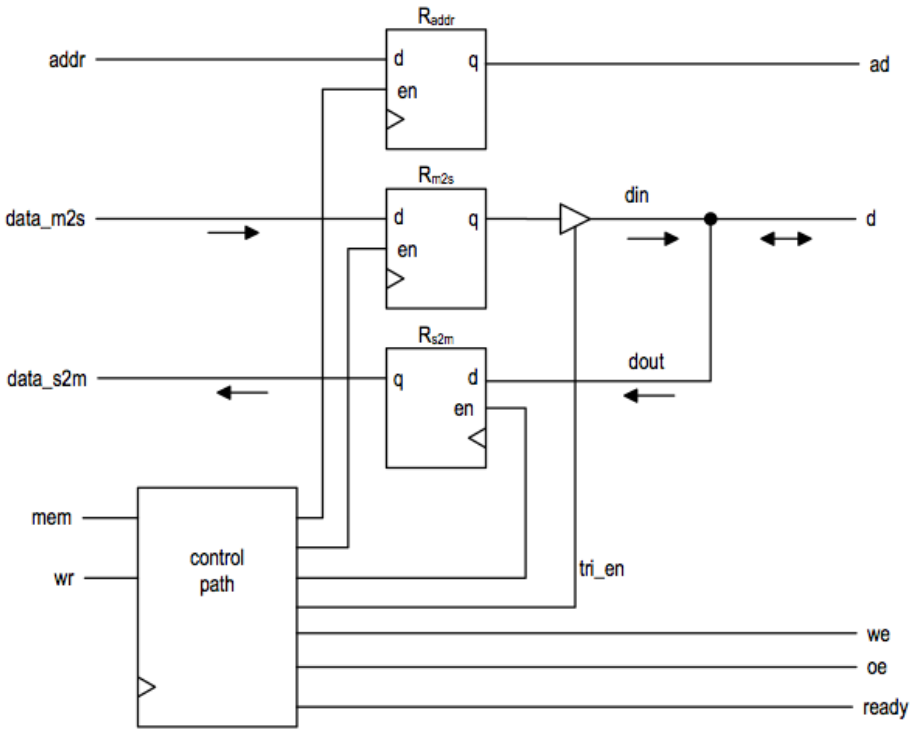

Figure 12.8 Block diagram of an SRAM controller.

Figure 9.1: SRAM Datapath Block Diagram (Figure 12.8 from the textbook).

architecture for this lab. Review the textbook to understand all components and I/O signals in this architecture. One of the challenges of this memory controller is creating the proper I/O signals to the SRAM. You will need to create the registers and tri-state driver as shown Figure 9.1.

The SRAM datapath diagram of Figure 9.1 involves several ports. Four of the ports represent connections to the external memory and are shown on the right side of the diagram. These ports include `ad`, (the memory address bus memory), `d`, (tri-state memory data bus), and the control signals, `we` and `oe`. The other ports are connections to the circuit within the FPGA that communicates with the memory. These ports include the address (`addr`), the data to write into the memory (`data_m2s`), the data to read from the memory (`data_s2m`), the state machine control inputs (`wr` and `mem`), and the state machine output (`ready`). You will include these these ports as well as other related ports in your SRAM controller entity.

Begin your memory controller by creating an entity named "`sramController`" that contains the ports listed in Table 9.1. Note that the names of the ports that are connected to the FPGA pins are different from the names used in Figure 9.1. These names are chosen to match the signal names in the default UCF file for the Nexys2 board. In addition, the memory we are using requires more signals than those shown in Figure 9.1 (i.e., CS, LB, UB, CLK, ADV, and CRE). You will need to determine the logic for each of these signals. The logic of each of these signals will be discussed later in the laboratory manual.

The logic value for some of these signals is constant and can be set with simple statements in your architecture. These constants are summarized below:

- **RamCLK**. Because your memory controller is only supporting the asynchronous memory read and write modes, the CLK signal (RamCLK output) should always be asserted low. Add a concurrent statement that assigns this output port the value '0'.
- **RamADV**. The ADV (RamADV output) is not used for asynchronous memory accesses. Again, assert the RamADV output port low at all times.
- **RamCRE**. The RamCRE signal is not used and it should also be asserted low at all times.

Generic	Type		Purpose
CLK_RATE	NATURAL		Indicates the frequency of the input clock.
Port Name	**Direction**	**Width**	**Purpose**
clk	Input	1	50 MHz clock
rst	Input	1	Asynchronous reset
addr	Input	23	Memory address
data_m2s	Input	16	The data from the memory controller that will be written into the memory (used for write operations)
mem	Input	1	Control signal to initiate a memory operation (mem='0' initiates a memory cycle)
rw	Input	1	Control signal to indicate whether the memory operation is a read (rw='1') or a write (rw='0')
data_s2m	Output	16	The data from the memory to the memory controller (obtained from a read operation)
data_valid	Output	1	Indicates that the data on the 'data_s2m' port is valid
ready	Output	1	Indicates whether the memory controller is ready for a memory operation
MemAdr	Output	23	The address pins to the memory device
MemOE	Output	1	Output enable pin to memory device
MemWR	Output	1	Write enable pin to memory device
RamCS	Output	1	Memory chip select (CE#)
RamLB	Output	1	Memory 'lower byte' control signal
RamUB	Output	1	Memory 'upper byte' control signal
RamCLK	Output	1	Memory CLK signal
RamADV	Output	1	Memory ADV signal
RamCRE	Output	1	Memory CRE signal
MemDB	Inout	16	Data pins to memory

Table 9.1: Entity Interface for the Asynchronous SRAM Controller.

Each of the important components shwon in Figure 9.1 must be included in your VHDL controller.

- **Control Path**. This represents the state machine to control the datapath elements. The state machine can be described by an ASMD diagram will described in more detail below.
- **Address register (Raddr)**. This register holds the value of the address used to drive the address pins on the SRAM.
- **Data out register (Rm2s)**. This register stores the value of the data to be written into the SRAM (m2s stands for 'M'emory controller 'to' 'S'ram).
- **Data in register (Rs2m)**. This register stores the value of the data read from the SRAM. This register is available to other parts of the circuit that want to use the SRAM data.
- **Tri-state driver**. This tri-state driver allows the data in the "data out register" to be driven onto the SRAM data bus during a write cycle. This tri-state driver will drive the d bus when the tri_en signal is asserted. The description of tri-state drivers in VHDL is described on page 135 of the text book. The following example demonstrates how the tri-state driver can be used in this lab.

```
d <= Rm2s when tri_en = '1' else (others => 'Z');
```

Update your VHDL architecture to include the address register, Raddr, the data out register, Rm2s, the data in register, Rs2m, and the tri-state driver. Later in the lab, you will create a state machine outputs at that control the loading of these three memory registers (i.e., the enable inputs). Attach the outputs of the address register to the memory address port and the tri-state driver to the memory data port.

State Machine

After creating the I/O registers and tri-state drivers, the next step is to create a state machine that sequences through all of the steps involved in a memory write and a memory read. You will need to create a state machine similar to (but not the same as) the state machine shown in Figure 12.10 or 12.13 in the textbook. Figure 12.13 from the textbook has been copied below as Figure 9.2 for your reference.

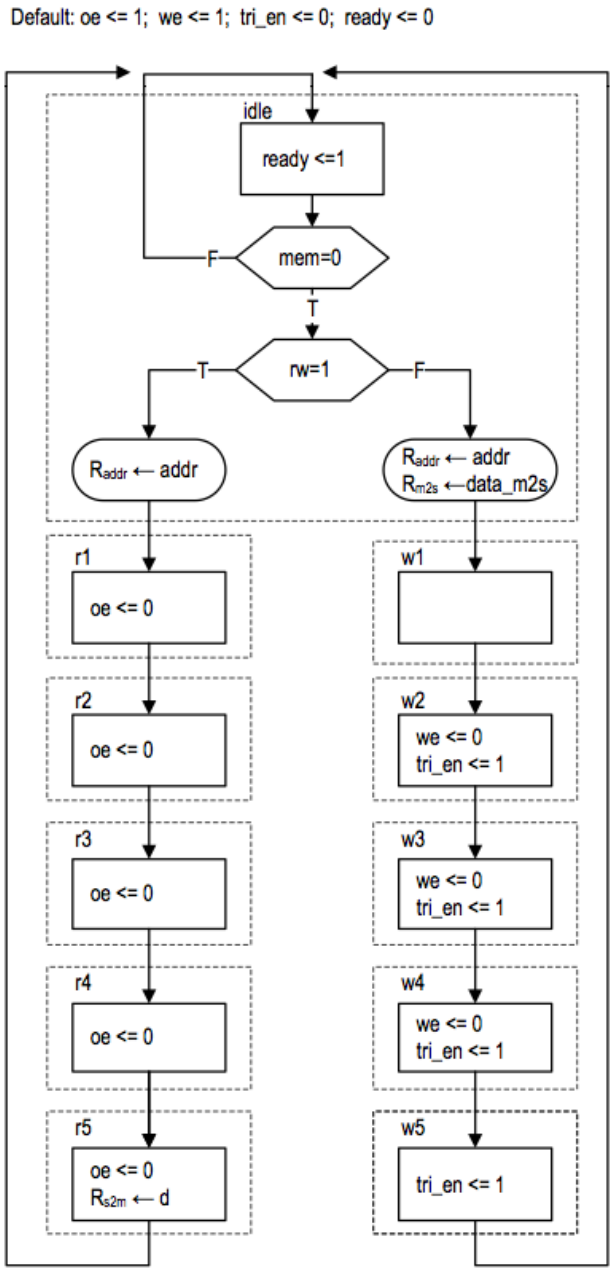

Figure 9.2: SRAM controller state machine. Note that we is RamWR and ce is RamCS. (Figure 12.13 from the textbook)

Your state machine will be similar to the one shown above—you will have an idle state, a read sequence, and a write sequence. However, there are a number of important differences for the state machine in this lab. Carefully read the following instructions to understand these differences. You will need to design your own controller and your controller must implement all of the functionality described below.

Power Up

The first important difference is that you must implement a "power-up" state. As discussed earlier during the data sheet section, this memory device requires a minimum amount of time to power-up the memory device. To support this power-up, create an initial "power-up" state that waits this minimum power-up time. You will need to compute how many clock cycles you must wait in this state using the CLK_RATE generic.

During this state, the CE# signal should be set to '1' (de-asserted). After completing the power-up wait time, proceed to the idle state. The CE# signal should be asserted (set to '0') in all states but the power-up state. In addition, the "ready" output signal should be de-asserted (set to '0') during the power-up state indicating that the controller is not yet ready to process memory transactions.

Read Sequence

When the mem signal is asserted (mem = '0') and the rw signal indicates a read (rw = '1') then you should proceed from the Idle state down the "Read" branch of the state machine. As shown in Figure 9.2 above, you will need to load the Raddr register when you take the transition from the "Idle" state to the first read state. This will initiate a new read function during the next clock (OE# controlled asynchronous write).

The memory read begins at the start of your first read state when the address pins change. Unlike the figure shown above, Figure 30 of the data sheet indicates that you do not need to assert the OE# signal when the read operation begins. When using this memory, you should NOT assert the OE# signal during the first memory cycle.

After this first state, you should implement one or more states in which the OE# signal IS asserted. You will need to refer to the data sheet to determine how many states the OE# signal should be asserted. During the final read state, you should load the value that you read from the memory into the Rs2m register for use by the internal FPGA circuit (as shown by the RTL statement in state r5 of Figure 9.2).

When you are done reading the memory, you will need to assert the 'data_valid' signal for one cycle to indicate that the data in the Rs2m register is valid. Create an RTL statement in the last read cycle of your ASM diagram that sets the data_valid signal. This signal should be zero at all other times.

Write Sequence

The write sequence also requires several states. Make sure you add each of the following steps in your state machine (you will need to determine how many clock cycles each steps will take). When the mem signal is asserted and the rw signal is de-asserted (i.e. rw = '0'), the state machine should initiate a write sequence. At the end of this clock cycle, the address that is on the addr lines is loaded into the address output pin registers (Raddr in Figure 12.8). In addition, the data to be written into the memory (data_m2s) is loaded into the data out register (Rm2s in Figure 12.8).

During the first write step, the address in the address register drives the memory. The WE# signal should be held high (i.e. de-asserted) to guarantee that the address setup time is met. During the next step, the tri-state enable signal must be asserted to drive the data values from the FPGA and into the memory. During the next step, the WE# should be asserted to initiate the write cycle. The address should not change and the data should continue to be asserted on the FPGA pins. During the final step, the WE# signal is de-asserted but the data bus is still asserted to meet the data hold time.

For the purposes of this lab, the signals ce, ub, and lb can be held constant during operation (you will need to determine what value they are constant). Carefully draw an ASMD diagram for your memory controller that implements all of the states necessary for power-up, reading, and writing.

Upload: Submit a scanned copy of your ASMD diagram.

Exercise 3 – Testbench Simulation

After reviewing your ASMD diagram with the TA, complete your VHDL architecture by implementing the state machine and datapath described in your ASMD diagram. This translation between ASMD diagram and VHDL code should be simple if you follow the guidelines described in the textbook.

An important considertation to make when creating your HDL is to generate glitch-free OE# and WE# signals. Consider using Moore output buffering to provide buffering on your OE# and WE# signals.

After you have created the VHDL for your memory controller and removed all syntax errors, you need to carefully simulate the operation of your controller. Like previous labs, you will be given a testbench to test your memory controller. For this lab, you will also be given a behavior model of the memory device you are using. This memory

model is written in Verilog and simulates the functionality of the memory in a timing accurate manner. This memory model is an invaluable way to demonstrate that you have correctly designed your sram controller circuit. To simulate your memory controller, download the lab testbench and all of the files associated with the memory model.

Download: tb_sramController.vhd

Download: cellram.v

Download: cellram_parameters.vh

The testbench will instance your memory controller as well as the memory simulation model and properly connect them. The testbench will exercise your memory controller by implementing a number of memory operations (i.e., reads and writes). If your memory controller is designed correctly, the simulation model will demonstrate proper reading and writing of the memory.

The memory model is designed to print a message when operations are being performed on the memory. Do not be surprised if you see a number of messages during the simulation of the memory controller. If your memory controller interacts with the memory incorrectly, the memory simulation model will print out an error message. These error messages indicate that you have violated some timing or operational constraint of the memory.

Simulate the testbench until you get the "DONE!" message. Note that you may need to simulate a long time to simulate the power-up sequence of your memory controller. Once the power-up sequence has completed, the testbench will simulate a few memory transactions and then print the "DONE!" message.

Upload: Copy and paste your working memory controller.

Exercise 4 – Top-Level Design

The final design exercise is to create a top-level design that incorporates your memory controller. This simple circuit will be used to verify your memory controller works with the actual SRAM memory on the board. Instance your memory controller into the top-level design and interface this controller as follows:

- Attach all top-level I/O signals from the memory controller to the top-level ports of your design (i.e., all SRAM address, data, and control signals)
- Create an 8-bit address register (AR) that is loaded with the value of the switches when button 0 is pressed. Use this 8 bit register to create an 23-bit address by concatenating 15 '0' bits to this address. This address is used as the input to your memory controller
- Create an 8 bit input data register (IDR) that is loaded with the value of the switches when button 1 is pressed. Use this 8-bit register to create the 16 bit data_m2s signal by copying your IDR in both the low and high byte of the sixteen bit value
- Instance your seven-segment display and drive the value of the display with the data_s2m signal generated by your memory controller. Your seven segment display controller will display the current value of this signal and will change when memory reads are performed
- Create the logic that will drive the mem and rw signals as described below. Make sure that you properly debounce these buttons to avoid initiating more than one read or write operation.
  - When Button 2 is pressed, initiate a write operation.
  - When Button 3 is pressed, initiate a read operation
- Issue a global reset when all four buttons are pressed simultaneously

Simulate your top-level design by performing a few read and write operations. Once you are confident that your design operates correctly, create a .ucf file that defines the location of every top-level FPGA pin. The default .ucf file contains all of the pins used for the memory. In addition to the pin locations, add your top-level timing constraint that indicates the clock frequency of your input clock. This constraint will make sure that the circuit operates within the limitations of the input clock[1].

[1] It will also generate an error if your circuit does not meet this constraint.

```
# Define a new timing constraint indicating a 50 MHz clock period
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 20 ns HIGH 50 %
```

After creating the .ucf file, synthesize your design and generate a bitfile. Test your design in hardware by writing a few values into memory using the buttons and the switches. After writing the values, read the values to verify that the values were properly written.

Upload: Copy and paste your top-level design.

Question: Summarize and justify all of the synthesis warnings you received when synthesizing this circuit.

Personal Exploration

For your personal exploration, modify the top-level design to add some additional features. Ideas include:

- Modify your state machine so you auto increment the address when reading from the memory (i.e., to quickly sequence through the memory locations)
- Provide an initial set of states that initializes the memory to some known value
- Create a memory tester state machine that writes and then reads from every location and indicates the status on the LEDs/seven segment display
- Implement the page read mode on your state machine

Pass Off

Demonstrate the following to a TA to pass off your lab:

- Show your memory controller simulation
- Demonstrate a working design to the TA by demonstrating the ability to read and write to memory.