Chapter 11: Modeling Memory

This chapter covers how to model memory arrays in VHDL. These models are technology independent, meaning that they can be ultimately synthesized into a wide range of semiconductor memory devices.

Learning Outcomes—After completing this chapter, you will be able to:

11.1 Describe the basic architecture and terminology for semiconductor-based memory systems.
11.2 Design a VHDL model for a read-only memory array.
11.3 Design a VHDL model for a read/write memory array.

11.1 Memory Architecture and Terminology

The term *memory* is used to describe a system with the ability to store digital information. The term *semiconductor memory* refers to systems that are implemented using integrated circuit technology. These types of systems store the digital information using transistors, fuses, and/or capacitors on a single semiconductor substrate. Memory can also be implemented using technology other than semiconductors. Disk drives store information by altering the polarity of magnetic fields on a circular substrate. The two magnetic polarities (north and south) are used to represent different logic values (i.e., 0 or 1). Optical disks use lasers to burn pits into reflective substrates. The binary information is represented by light either being reflected (no pit) or not reflected (pit present). Semiconductor memory does not have any moving parts, so it is called *solid state memory* and can hold more information per unit area than disk memory. Regardless of the technology used to store the binary data, all memory has common attributes and terminology that are discussed in this chapter.

11.1.1 Memory Map Model

The information stored in memory is called the **data**. When information is placed into memory, it is called a **write**. When information is retrieved from memory, it is called a **read**. In order to access data in memory, an **address** is used. While data can be accessed as individual bits, in order to reduce the number of address locations needed, data is typically grouped into *N-bit words*. If a memory system has $N = 8$, this means that 8-bits of data are stored at each address. The number of address locations is described using the variable M. The overall size of the memory is typically stated by saying "$M \times N$." For example, if we had a 16×8 memory system, that means that there are 16 address locations, each capable of storing a byte of data. This memory would have a **capacity** of $16 \times 8 = 128$ bits. Since the address is implemented as a binary code, the number of lines in the address bus (n) will dictate the number of address locations that the memory system will have ($M = 2^n$). Figure 11.1 shows a graphical depiction of how data resides in memory. This type of graphic is called a *memory map model*.

B. J. LaMeres, *Quick Start Guide to VHDL*, https://doi.org/10.1007/978-3-030-04516-6_11

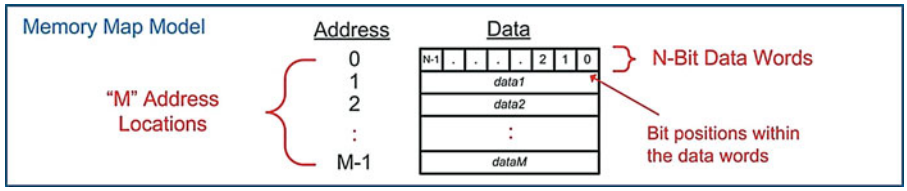


Fig. 11.1
Memory map model

11.1.2 Volatile vs. Nonvolatile Memory

Memory is classified into two categories depending on whether it can store information when power is removed or not. The term **nonvolatile** is used to describe memory that *holds* information when the power is removed, while the term **volatile** is used to describe memory that loses its information when power is removed. Historically, volatile memory is able to run at faster speeds compared to nonvolatile memory, so it is used as the primary storage mechanism, while a digital system is running. Nonvolatile memory is necessary in order to hold critical operation information for a digital system such as start-up instructions, operations systems, and applications.

11.1.3 Read-Only vs. Read/Write Memory

Memory can also be classified into two categories with respect to how data is accessed. **Read-only memory (ROM)** is a device that cannot be written to during normal operation. This type of memory is useful for holding critical system information or programs that should not be altered, while the system is running. **Read/write** memory refers to memory that can be read and written to during normal operation and is used to hold temporary data and variables.

11.1.4 Random Access vs. Sequential Access

Random access memory (RAM) describes memory in which any location in the system can be accessed at any time. The opposite of this is **sequential access** memory, in which not all address locations are immediately available. An example of a sequential access memory system is a tape drive. In order to access the desired address in this system, the tape spool must be spun until the address is in a position that can be observed. Most semiconductor memory in modern systems is random access. The terms RAM and ROM have been adopted, somewhat inaccurately, to also describe groups of memory with particular behavior. While the term ROM technically describes a system that cannot be written to, it has taken on the additional association of being the term to describe nonvolatile memory. While the term RAM technically describes how data is accessed, it has taken on the additional association of being the term to describe volatile memory. When describing modern memory systems, the terms RAM and ROM are used most commonly to describe the characteristics of the memory being used; however, modern memory systems can be both read/write and nonvolatile, and the majority of memory is random access.

Concept Check

CC11.1 An 8-bit wide memory has eight address lines. What is its capacity in bits?

(A) 64 (B) 256 (C) 1024 (D) 2048

11.2 Modeling Read-Only Memory

Modeling of memory in VHDL is accomplished using the *array* data type. Recall the syntax for declaring a new array type below:

```
type name is array (<range>) of <element_type>;
```

To create the ROM memory array, a new type is declared (e.g., ROM_type) that is an array. The *range* represents the addressing of the memory array and is provided as an integer. The *element_type* of the array specifies the data type to be stored at each address and represents the data in the memory array. The type of the element should be std_logic_vector with a width of *N*. To define a 4 × 4 array of memory, we would use the following syntax.

Example:

```
type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);
```

Notice that the address is provided as an integer (0 to 3). This will require two address bits. Also notice that this defines 4-bit data words. Next, we define a new constant of type ROM_type. When defining a constant, we provide the contents at each address.

Example:

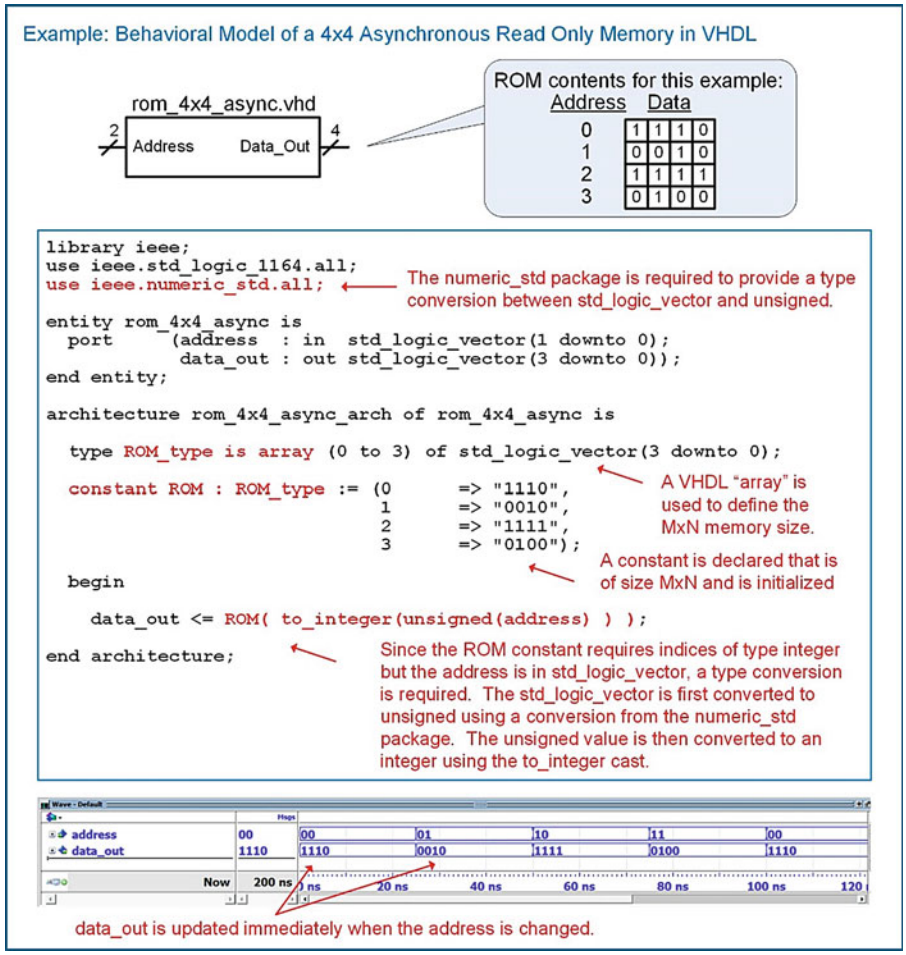
```
constant ROM : ROM_type := (0 => "1110",
                            1 => "0010",
                            2 => "1111",
                            3 => "0100");
```

At this point, the ROM array is declared and initialized. In order to model the read behavior, a concurrent signal assignment is used. The assignment will be made to the output data_out based on the incoming address. The assignment to data_out will be the contents of the constant ROM at a particular address. Since the index of a VHDL array needs to be provided as an integer (e.g., 0, 1, 2, 3) and the address of the memory system is provided as a std_logic_vector, a type conversion is required. Since there is not a direct conversion from type std_logic_vector to integer, two conversions are required. The first step is to convert the address from std_logic_vector to unsigned using the *unsigned* type conversion. This conversion exists within the numeric_std package. The second step is to convert the address from unsigned to integer using the *to_integer* conversion. The final assignment is as follows:

Example:

```
data_out <= ROM(to_integer(unsigned(address)));
```

Example 11.1 shows the entire VHDL model for this memory system and the simulation waveform. In the simulation, each possible address is provided (i.e., "00," "01," "10," and "11"). For each address, the corresponding information appears on the data_out port. Since this is an asynchronous memory system, the data appears immediately upon receiving a new address.



Example 11.1
Behavioral model of a 4 × 4 asynchronous read-only memory in VHDL

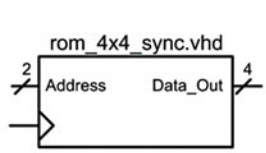
Latency can be modeled in memory systems by using delayed signal assignments. In the above example, if the memory system had a latency of 5 ns, this could be modeled using the following approach:

Example:

```
data_out <= ROM(to_integer(unsigned(address))) after 5 ns;
```

A synchronous ROM can be created in a similar manner. In this approach, a clock edge is used to trigger when the data_out port is updated. A sensitivity list is used that contains only the signal clock to trigger the assignment. A rising edge condition is then used in an if/then statement to make the assignment only on a rising edge. Example 11.2 shows the VHDL model and simulation waveform for this system. Notice that prior to the first clock edge, the simulator does not know what to assign to data_out, so it lists the value as *uninitialized*.

Example: Behavioral Model of a 4x4 Synchronous Read Only Memory in VHDL



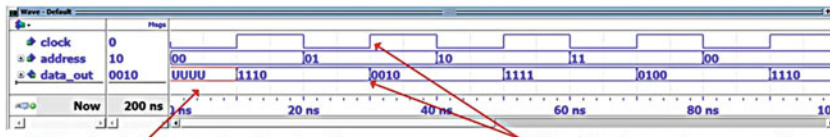
ROM contents for this example:

Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom_4x4_sync is
  port     (clock    : in  std_logic;
            address  : in  std_logic_vector(1 downto 0);
            data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rom_4x4_sync_arch of rom_4x4_sync is

  type ROM_type is array (0 to 3) of std_logic_vector(3 downto 0);

  constant ROM : ROM_type := (0      => "1110",
                              1      => "0010",
                              2      => "1111",
                              3      => "0100");
  begin

   MEMORY : process (clock)
     begin
       if (clock'event and clock='1') then
          data_out <= ROM( to_integer(unsigned(address) ) );
      end if;
   end process;

end architecture;
```

To model synchronous behavior, the clock is placed in the sensitivity list, and a rising edge condition is used to trigger the assignment.

When there is not a clock edge, the memory will hold its last output on data_out.



Before the first clock edge, the simulator doesn't know what the output should be.

The data does not appear on the output until a rising edge of clock.

Example 11.2
Behavioral model of a 4 × 4 synchronous read-only memory in VHDL

Concept Check

CC11.2 Explain the advantage of modeling memory in VHDL without going into the details of the storage cell operation.

(A) It allows the details of the storage cell to be abstracted from the functional operation of the memory system.

(B) It is too difficult to model the analog behavior of the storage cell.

(C) There are too many cells to model so the simulation would take too long.

(D) It lets both ROM and R/W memory to be modeled in a similar manner.

11.3 Modeling Read/Write Memory

In a read/write memory model, a new type is created using a VHDL *array* (e.g., RW_type) that defines the size of the storage system. To create the memory, a new signal is declared with the array type.

Example:

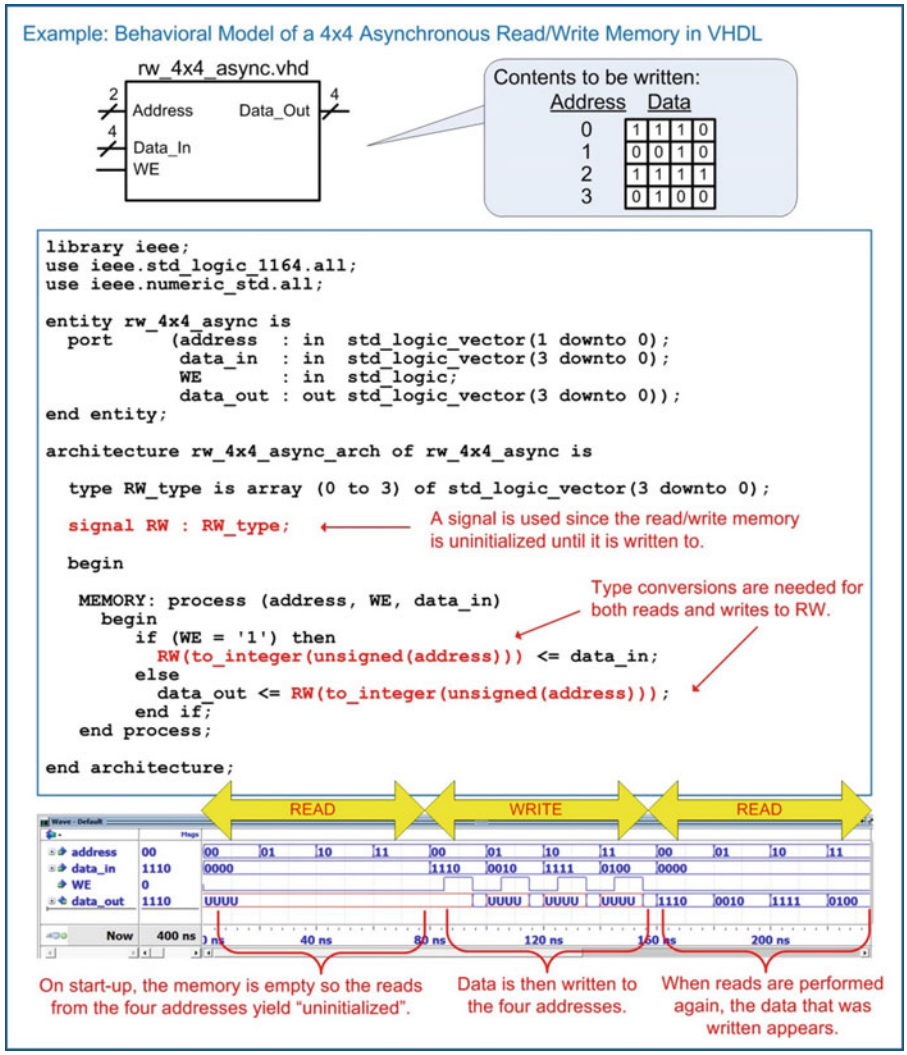
```
type RW_type is array (0 to 3) std_logic_vector(3 downto 0);
signal RW : RW_type;
```

Note that a signal is used in a read/write system as opposed to a constant as in the read-only memory system. This is because a read/write system is uninitialized until it is written to. A process is then used to model the behavior of the memory system. Since this is an asynchronous system, all inputs are listed in the sensitivity list (i.e., address, WE, and data_in). The process first checks whether the write enable line is asserted (WE = 1), which indicates a write cycle is being performed. If it is, then it makes an assignment to the RW signal at the location provided by the address input with the data provided by the data_in input. Since the RW array is indexed using integers, type conversions are required to convert the address from std_logic_vector to integer. When WE is not asserted (WE = 0), a read cycle is being performed. In this case, the process makes an assignment to data_out with the contents stored at the provided address. This assignment also requires type conversions to change the address from std_logic_vector to integer. The following syntax implements this behavior:

Example:

```
MEMORY: process (address, WE, data_in)
  begin
     if (WE = '1') then
        RW(to_integer(unsigned(address))) <= data_in;
     else
        data_out <= RW(to_integer(unsigned(address)));
     end if;
end process;
```

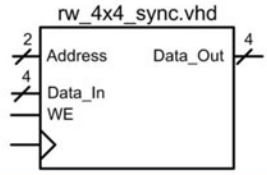
A read/write memory does not contain information until its storage locations are written to. As a result, if the memory is read from before it has been written to, the simulation will return *uninitialized*. Example 11.3 shows the entire VHDL model for an asynchronous read/write memory and the simulation waveform showing read/write cycles.



Example 11.3
Behavioral model of a 4 × 4 asynchronous read/write memory in VHDL

A synchronous read/write memory is made in a similar manner with the exception that a clock is used to trigger the signal assignments in the sensitivity list. The WE signal acts as a synchronous control signal indicating whether assignments are read from or written to the RW array. Example 11.4 shows the entire VHDL model for a synchronous read/write memory and the simulation waveform showing both read and write cycles.

Example: Behavioral Model of a 4x4 Synchronous Read/Write Memory in VHDL



Contents to be written:

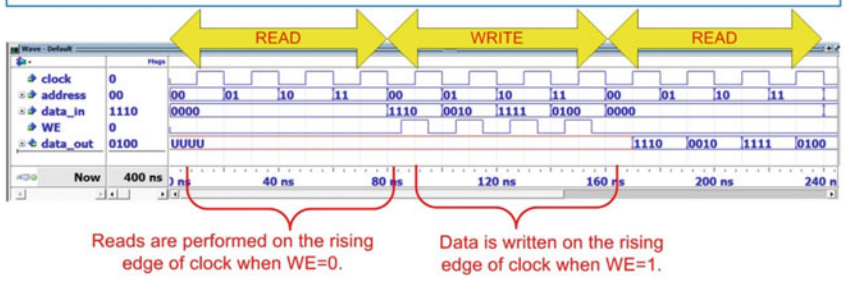
Address	Data
0	1 1 1 0
1	0 0 1 0
2	1 1 1 1
3	0 1 0 0

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rw_4x4_sync is
  port      (clock    : in  std_logic;
             address  : in  std_logic_vector(1 downto 0);
             data_in  : in  std_logic_vector(3 downto 0);
             WE       : in  std_logic;
             data_out : out std_logic_vector(3 downto 0));
end entity;

architecture rw_4x4_sync_arch of rw_4x4_sync is

  type RW_type is array (0 to 3) of std_logic_vector(3 downto 0);

  signal RW : RW_type;

  begin

   MEMORY : process (clock)
     begin
        if (clock'event and clock='1') then
           if (WE = '1') then
             RW(to_integer(unsigned(address))) <= data_in;
           else
             data_out <= RW(to_integer(unsigned(address)));
           end if;
        end if;
   end process;

end architecture;
```

Synchronous behavior is modeled by listing clock in the sensitivity list and using a rising edge condition.

The WE control signal dictates whether information is read or written to the RW array.

Type conversions are needed for both reads and writes to RW.



Reads are performed on the rising edge of clock when WE=0.

Data is written on the rising edge of clock when WE=1.

Example 11.4
Behavioral model of a 4 × 4 synchronous read/write memory in VHDL

Concept Check

CC11.3 Does modeling the R/W memory as an uninitialized array accurately describe the behavior of real R/W memory technology?

(A) Yes. Read/write memory is not initialized upon power up.

(B) No. Read/write memory should be initialized to all zeros to model the reset behavior found in memory.

Summary

- The term memory refers to large arrays of digital storage. The technology used in memory is typically optimized for storage density at the expense of control capability. This is different from a D-flip-flop, which is optimized for complete control at the bit level.
- A memory device always contains an address bus input. The number of bits in the address bus dictates how many storage locations can be accessed. An *n*-bit address bus can access 2^n (or *M*) storage locations.
- The width of each storage location (*N*) allows the density of the memory array to be increased by reading and writing vectors of data instead of individual bits.
- A memory map is a graphical depiction of a memory array. A memory map is useful to give an overview of the capacity of the array and how different address ranges of the array are used.
- A read is an operation in which data is retrieved from memory. A write is an operation in which data is stored to memory.
- An asynchronous memory array responds immediately to its control inputs. A synchronous memory array only responds on the triggering edge of clock.
- Volatile memory will lose its data when the power is removed. Nonvolatile memory will retain its data when the power is removed.
- Read-only memory (ROM) is a memory type that cannot be written to during normal operation. Read/write (R/W) memory is a memory type that can be written to during normal operation. Both ROM and R/W memory can be read from during normal operation.
- Random access memory (RAM) is a memory type in which any location in memory can be accessed at any time. In sequential access memory, the data can only be retrieved in a linear sequence. This means that in sequential memory the data cannot be accessed arbitrarily.
- Memory can be modeled in VHDL using the array data type.
- Read-only memory in VHDL is implemented as an array of constants.
- Read/write memory in VHDL is implemented an as array of signal vectors.

Exercise Problems

Section 11.1: Memory Architecture and Terminology

11.1.1 For a 512k × 32 memory system, how many unique address locations are there? Give the exact number.

11.1.2 For a 512k × 32 memory system, what is the data width at each address location?

11.1.3 For a 512k × 32 memory system, what is the *capacity* in bits?

11.1.4 For a 512k × 32-bit memory system, what is the *capacity* in bytes?

11.1.5 For a 512k × 32 memory system, how wide does the incoming address bus need to be in order to access every unique address location?

Section 11.2: Modeling Read-Only Memory

11.2.1 Design a VHDL model for the 16 × 8, asynchronous, read-only memory system shown in Fig. 11.2. The system should contain the information provided in the memory map. Create a test bench to simulate your model by reading from each of the 16 unique addresses and observing Data_Out to verify it contains the information in the memory map.

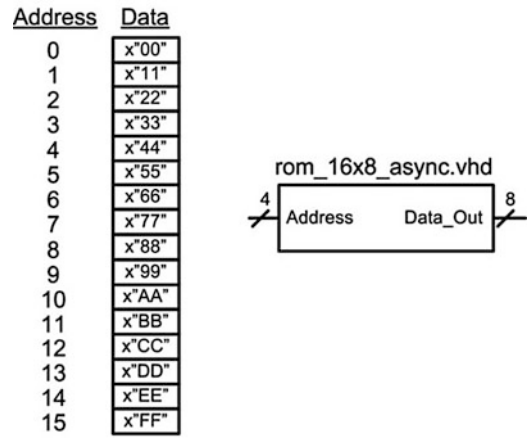


Fig. 11.2
16 × 8 asynchronous ROM block diagram

11.2.2 Design a VHDL model for the 16 × 8, synchronous, read-only memory system shown in Fig. 11.3. The system should contain the information provided in the memory map. Create a test bench to simulate your model by reading from each of the 16 unique addresses and

observing Data_Out to verify it contains the information in the memory map.

Address	Data
0	x"FF"
1	x"EE"
2	x"DD"
3	x"CC"
4	x"BB"
5	x"AA"
6	x"99"
7	x"88"
8	x"77"
9	x"66"
10	x"55"
11	x"44"
12	x"33"
13	x"22"
14	x"11"
15	x"00"

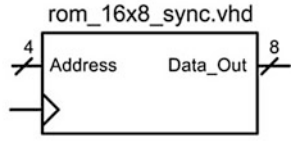


Fig. 11.3
16 × 8 synchronous ROM block diagram

Section 11.3: Modeling Read/Write Memory

11.3.1 Design a VHDL model for the 16 × 8, asynchronous, read/write memory system shown in Fig. 11.4. Create a test bench to simulate your model. Your test bench should first read from all of the address locations to verify they are uninitialized. Next, your test bench should write unique information to each of the address locations. Finally, your test bench should read from each address location to verify that the information that was written was stored and can be successfully retrieved.

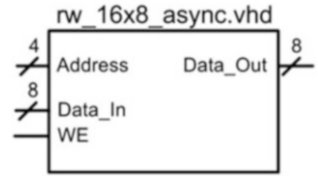


Fig. 11.4
16 × 8 asynchronous R/W block diagram

11.3.2 Design a VHDL model for the 16 × 8, synchronous, read/write memory system shown in Fig. 11.5. Create a test bench to simulate your model. Your test bench should first read from all of the address locations to verify they are uninitialized. Next, your test bench should write unique information to each of the address locations. Finally, your test bench should read from each address location to verify that the information that was written was stored and can be successfully retrieved.

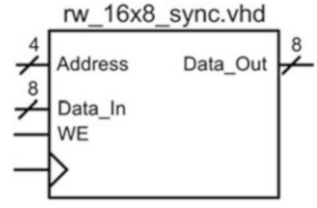


Fig. 11.5
16 × 8 synchronous R/W block diagram