International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

Memories using in Digital System Design

AUTHORS – NADIMPALLI RAJKUMAR[1], ARUNKUMAR S[2], PAVAN KUMAR E[3]

[1,2,3]*Department of Electronics and Communication Engineering, Sai Vidya Institute of Technology*

ABSTRACT : *Memory is a main component of the digital system. Memories are used to store the data within the system. Every system needs memory to store the data and process the data to complete its operation successfully. Memory can volatile or non-volatile, volatile memory stores the data until unless power is supplied. Non-volatile memory stores the data even after the power is removed. Random Access Memory (RAM) is of Volatile memory and Read Only Memory (ROM) is a Non- volatile memory. In this paper we learn the different types of RAM and ROM and their operation in detail. Speed of the system depends on type of the memory used in the system.*

INTRODUCTION

In this paper, we will discuss various types of memory provided by manufacturers, either as individual integrated circuits or as resources within ASIC or FPGA fabrics. We will discuss the distinguishing properties of each kind of memory, including their timing characteristics and costs, and describe how to model some of them in Verilog. We will distinguish between memory that can be both read and written, called random access memory (RAM), and memory that can only be read, called read-only memory (ROM). We use the term RAM instead of read/write memory largely for historical reasons. Memories in very early computers enforced sequential access, that is, access to locations in increasing order of address, due to the physical medium on which the data was stored. The invention of memories in which locations could be read and written with equal facility in any order was a significant milestone, and so the term RAM has stuck.

ASYNCHRONOUS STATIC RAM

One of the simplest forms of memory is asynchronous static RAM. It is asynchronous because it does not rely on a clock for its timing. The term static means that the stored data persists ndefinitely so long as power is applied to the memory component. Static RAM is volatile, meaning that it requires power to maintain the stored data, and loses data if power is removed. Since engineers are fond of abbreviations, the term static RAM is usually further shortened to SRAM. Asynchronous SRAM internally uses 1-bit storage cells that are similar to the D-latch circuit. Within the memory component, the address is decoded to select a particular group of cells that comprise one location. For largely historical reasons, most manufacturers use active-low logic for the control signals. Further, since asynchronous SRAMs are usually only available as packaged integrated circuits, and not as blocks in ASIC libraries or FPGAs, they usually have bidirectional tristate data input/output pins.

Given that the storage cells in an asynchronous SRAM are basically latches, it is not surprising that the timing is similar to that of a D-latch. The control section that sequences the datapath containing the memory must ensure that the address is stable before commencing the write operation and is held stable during the entire operation. Otherwise, locations other than the one to be updated may be affected. In isolation, we can also perform back-to-back read operations simply by changing the address value. The read operation is essentially a combinational operation, involving decoding the address and multiplexing the selected latch-cell's value onto the data outputs. Changing the address simply causes a different cell's value to appear on the outputs after a propagation delay.

Manufacturers of asynchronous SRAM chips publish the timing parameters for write and read operations in data sheets. The parameters typically include setup and hold times for address and data values, and delays for turning tristate drivers on and off.

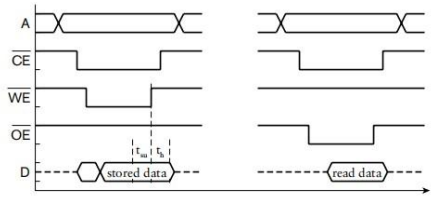

Fig1: Timing for write and read operations in an asynchronous SRAM.

Other performance-related parameters are the write cycle time and the read cycle time, which are the times taken to complete write and read operations, respectively. Manufacturers offer chips in different speed grades, with faster chips usually costing more. This allows us, as designers, to make cost/performance trade-offs in our designs.

While asynchronous SRAMs are conceptually simple and have simple timing behavior, the fact that they are asynchronous can make them difficult to use in clocked synchronous systems. The need to set up and hold

address and data values before and after activation of the control signals and to keep the values stable during the entire cycle means that we must either perform operations over multiple clock cycles, or use delay elements to ensure correct timing within a clock cycle. A asynchronous SRAMs are usually used only in systems with low performance requirements, where their low cost is a benefit.

SYNCHRONOUS STATIC RAM

Given the difficulties associated with asynchronous SRAMs, many memory component vendors and implementation fabrics provide synchronous SRAMs, otherwise known as SSRAMs. The internal storage cells of SSRAMs are the same as those of asynchronous SRAMs. However, the interface includes clocked registers for storing the address, input data and control signal values, and in some cases, output data. In this section, we will describe two forms of SSRAMs in general terms. The details of control signals and timing will vary between SSRAMs provided by different component vendors and implementation fabrics. As always, we need to read and understand the data sheets before using a component in a design.

The simplest kind of SSRAM is often called a flowthrough SSRAM. It includes registers on the inputs, but not on the data outputs. The term flow-through refers to the fact that data read from the memory cells flows through directly to the data outputs. Having registers on the inputs allows us to generate the address, data and control signal values according to our clocked synchronous design methodology, ensuring that they are stable in time for a clock edge.

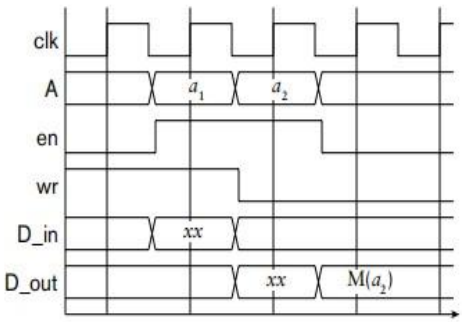

Fig2: Timing for a flow-through SSRAM.

Another method of SSRAM is called a pipelined SSRAM. It includes a register on the data output, as well as registers on the inputs. If there is no time in which to perform combinational operations on the read data before the next clock edge, it needs to be stored in an output register and used in the subsequent clock cycle. A pipelined SSRAM provides that output register. We declare a variable to represent the stored register value and assign a new value to it on a rising clock edge. We can promote this approach to model an SSRAM in Verilog. We need to declare a variable that represents all of the locations in the memory. The way to do this is to declare an array variable, which represents a collection of values, each with an index that corresponds to its location in the array. For example, to model a 4K*16-bit memory, we would write the following declaration: reg [15:0] data_RAM [0:4095];

The declaration specifies a variable named data_RAM that is an array with elements index from 0 to 4095. Each element is a 16-bit vector. Once we have declare the variable representing the storage, we declare an always block that performs the write and read operations. The block is similar in form to that for a register. For example, an always block of a model a flow-through SSRAM based on the

variable declaration above is

always @(posedge clk)

if (en) if (wr) begin data_RAM[a] <= d_in; d_out <= d_in; end else

d_out <= data_RAM[a];

Since there are many minor difference on the general concept of a pipelined SSRAM, it is hard to present a general template, especially one that can be recognized by synthesis tools. A common alternative approach is to use a CAD tool that generates a memory circuit and a Verilog model of that circuit. We can then instantiate the generated model as a component in a larger system.

MULTIPORT MEMORIES

A multiport memory usually consumes more circuit area more than a single-port memory with the same number of bits of storage, since it has separate address decoders and data multiplexers for each access port. Only the internal storage cells of the memory are shared between the multiple ports, though additional wiring is needed to connect the cells to the access ports. However, the cost of the extra circuit area is necessitate.

In some applications, such as high performance graphics processing and high-speed network connections. Suppose we have one subsystem producing data to store in the memory, and another subsystem accessing the data to process it in some way. If we use a single-port memory, we would need to multiplex the addresses and input data from the subsystems into the memory, and we would have to arrange the control sections of the subsystems so that they take turns to access the memory.

There are two potential problems here. First, if the combined rate at which the subsystems need to move

data in and out of the memory exceeds the rate at which a single access port can operate, the memory becomes a bottleneck. Second, even if the average rates don't exceed the capacity of a single access port, if the two subsystems need to access the memory at the same time, one must wait, possibly causing it to lose data. Having separate access ports for the subsystems obviates both of these problems. The only remaining difficulty is the case of both subsystems accessing the same memory location at the same time. If both accesses are reads, they can proceed. If one or both is a write, the effect depends on the a feature of the particular dual-port memory. In an asynchronous dual-port memory, a write operation performed concurrently with a read of the same location will result in the written data being reflected on the read port after some delay. Two write operations performed concurrently to the same location result in an unpredictable value being stored. In the case of a synchronous dual-port memory, the effect of concurrent write operations depends on when the operations are performed internally by the memory. We should advice from the data sheet for the memory component to understand the effect.

Some multiport memories, particularly those manufactured as packaged components, provide additional circuits that compare the addresses on the access ports and indicate when contention arises. They may also provide circuits to arbitrate between conflicting accesses, ensuring that one proceeds only after the other has completed. If we are using multiport memory components or circuit blocks that do not provide such features and our application may result in conflicting accesses, we need to include some form of arbitration as a separate part of the control section in our design. An alternative is to ensure that the subsystems accessing the memory through separate ports always access separate location.

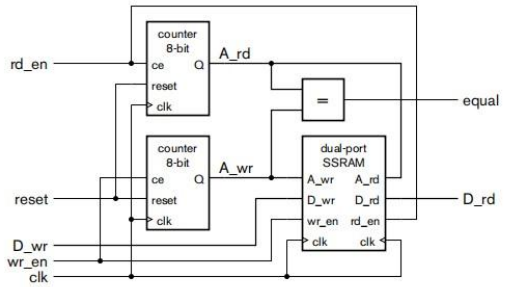

Fig3: Datapath for a FIFO using a dual-port memory.

One specialized form of dual-port memory is a firstin first-outmemory, or FIFO. It is used to queue data arriving from a source to be processed in order of arrival by another subsystem. The data that is first in to the FIFO is the first that comes out; hence, the name. The most common way of building a FIFO is to use a dual-port memory as a circular buffer for the data storage, with one port accepting data from the source and the other port reading data to give to the processing subsystem. Each port has an address counter to keep track of where data is written or read. Data written to the FIFO is stored in successive free locations. When the write-address counter reaches the last location, it wraps to location 0. As data is read, the read-address counter is advanced to the next available location, also wrapping to 0 when the last location is reached. If the write address wraps around and catches up with the read address, the FIFO is full and can accept no more data. If the read address catches up with the write address, the FIFO is empty and can provide no more data. FIFO can store a variable amount of data, depending on the rates of writing and reading data. The size of memory requied in a FIFO depends on the maximum amount by which reading of data lags writing. Determining the maximum size may be difficult to do. We may need to evaluate worst-case scenarios for our application using mathematical or statistical models of data rates or using simulation.

DYNAMIC RAM

Dynamic RAM (DRAM) is another form of volatile memory that uses a different form of storage cell for storing data. Static RAM uses storage cells that are similar to D-latches. In contrast, a storage cell for a dynamic RAM uses a single capacitor and a single transistor, The DRAM cells are thus much smaller than SRAM cells, so we can fit many more of them on a chip, making the cost as per bit of storage lower. However, the access times of DRAMs are longer than those of SRAMs, and the complexity of access and control is greater. Thus, there is a trade-off of cost, performance and complicated against memory capacity. DRAMs are most commonly used as the main memory in computer systems, since they desires of the need for high capacity with relatively low cost. However, they can also be used in other digital systems. The choice between SRAM and DRAM depends on the requirements and constraints of each application.

When the transistor is turned off, the capacitor is isolated from the bit line, thus storing the charge on the capacitor. To write to the cell, the DRAM control circuit pulls the bit line high or low and turns on the transistor, thus charging or discharging the capacitor. To read from the cell, the DRAM control circuit precharges the bit line to an intermediate level, then turns on the transistor. As the charges on the capacitor and the bit line equalize, the voltage on the bit line either increases slightly or decreases slightly, depending on whether the storage capacitor was charged or discharged.

A sensor detects and amplifies the change, thus determining whether the cell stored a 1 or a 0. Unfortunately, this process destroys the stored value in the cell, so the control circuit must then restore the value

by pulling the bit line high or low, as appropriate, before turning off the transistor. The time taken to complete the restoration is added to the access time, making the overall read cycle significantly longer than than that for an SRAM. Off, the capacitor is isolated from the bit line, thus storing the charge on the capacitor. To write to the cell, the DRAM control circuit pulls the bit line high or low and turns on the transistor, thus charging or discharging the capacitor. To read from the cell, the DRAM control circuit precharges the bit line to an intermediate level, then turns on the transistor. As the charges on the capacitor and the bit line equalize, the voltage on the bit line either increases slightly or decreases slightly, depending on whether the storage capacitor was charged or discharged. A sensor detects and amplifies the change, thus determining whether the cell stored a 1 or a 0. Unfortunately, this process destroys the stored value in the cell, so the control circuit must then restore the value by pulling the bit line high or low, as appropriate, before turning off the transistor. The time taken to complete the restoration is added to the access time, making the overall read cycle significantly longer than than that for an SRAM.

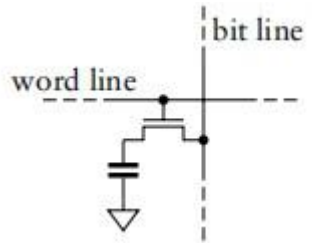

Fig4: A DRAM storage cell

Another property of a DRAM cell is that, while the transistor is turned off, charge leaks from the capacitor. This is the meaning of the term "dynamic" applied to DRAMs. To compensate, the control circuit must read and restore the value in each cell in the DRAM before the charge decays too much. This process is called refreshing the DRAM. DRAM manufacturers typically specify a period of 64ms between refreshes for each cell. The cells in a DRAM are typically organized into several rectangular arrays, called banks, and the DRAM control circuit is organized to refresh one row of each bank at a time. Since the DRAM cannot perform a normal write or read operation while it is refreshing a row, the refresh operations must be interleaved between writes and reads. Depending on the application, it may be possible to refresh all rows in a burst once every 64ms. Alternatively, we may have to refresh one row at a time between writes and reads, making sure that all rows are refreshed within 64ms. The important thing is to avoid scheduling a refresh when a write or read is required and cannot be deferred.

Historically, timing of DRAM control signals used to be asynchronous, and management of refreshing was performed by control circuits external to the DRAM chips. More recently, manufacturers changed to synchronous DRAMs (SDRAMs) that use registers on inputs to sample address, data and control signals on clock edges. This is analogous to the difference between asynchronous and synchronous SRAMs, and makes it easier to incorporate DRAMs into systems that use a clocked synchronous timing methodology. Manufacturers have also incorporated refresh control circuits into the DRAM chips, also making use of DRAMs easier. Since applications with very high data transfer rate requirements may be limited by the relatively slow access times of DRAMs, manufacturers have more recently incorporated further features to improve performance. These include the ability to access a burst of data from successive locations without having to provide the address for each, other than the first, and the ability to transfer on both rising and falling clock edges. These features are mainly motivated by the need to provide high-speed bursts of data in computer systems, but they can also be of benefit in non-computer digital systems. Because of the relative complicated of controlling DRAMs, we will not go into detail of the control signals required and their sequencing.

READ-ONLY MEMORIES

The memories that we have seen so far has a ability to do both read the stored data and update it autocratically. In contrast, a read-only memory, or ROM, has only ability to read the stored data. This is useful in cases where the data is constant, so there is no need to update it. The data is either incorporated into the circuit during its manufacture, or is programmed into the ROM subsequently. We will describe number of ROM that take one or other of these approaches.

Combinational ROMs

A simple ROM is a combinational circuit that points from an input address to a constant data value. We could specify the ROM satisfy in tabular form, with a row for each address and an entry showing the data value for that address. Such a table is essentially a truth table, so we could, in principle, implement the mapping using the combinational circuit design techniques. However, ROM circuit structures are generally much denser than arbitrary gate-based circuits, because each ROM cell needs at most one transistor. Indeed, for a complex combinational function with multiple outputs, it may be better to use a ROM to implement the function than a gatebased circuit.

In FPGA fabrics that will provide SSRAM blocks, we can use an SSRAM block as a ROM. We modify the always-block template for the memory to omit the part that updates the memory content. We could include a case statement to determine the data output. The content of the memory is loaded into the FPGA as part of its

programming when the system is turned on. Thereafter, since the data is not updated, it is constant. For large ROMs, writing the data directly in the Verilog code like this is very cumbersome. Values are read from the file into successive elements of the specified variable until either the end of the file is reached or all elements of the variable are loaded.

Programmable ROMs

ROMs in which the contents are manufactured into the memory are suitable for applications where the number of manufactured parts is high and where we are sure that the contents will not need to change over the lifetime of the product. In other applications, we would prefer to be able to revise the ROM contents from time to time, or to use a form of ROM with lower costs for low-volume production. A programmable ROM (PROM) meets these requirements. It is manufactured as a separately packaged chip with no content stored in its memory cells. The memory contents are programmed into the cells after manufacture, either using a special programming device before the chip is assembled into a system, or using special programming circuits when the chip is in the final system. There are a number of forms of PROMs. Early PROMs used fusible links to program the memory cells. Once a link was fused, it could not be replaced, so programming could only be done once. These devices are now largely obsolete. They were replaced by PROMs that could be erased, either with ultraviolet light so its called as EPROMs, or electrically using a higher-than-normal powersupply voltage so its called as electrically erasable PROMs, or EEPROMs.

Flash Memories

Most new designs use flash memory, which is a form of electrically erasable programmable ROM. It is organized so that blocks of storage can be erased at once, followed by programming of individual memory locations. A flash memory typically allows only a limited number of erasure and programming operations, typically hundreds of thousands, before the device “wears out.” Thus, flash memories are not a suitable replacement for RAMs. There are two kinds of flash memories, NOR and NAND flash, referring to the organization of the transistors that make up the memory cells. Both kinds are organized as blocks (commonly of 16, 64, 128, or 256 Kbytes) that must be erased in whole before being written. In a NOR flash memory, locations can then be written (once per erasure) and read (an arbitrary number of times) in random order. The IC has similar address, data and control signals to an SRAM and can read data with a comparable access time, making it suitable for use as a program memory for an embedded processor, for storing configuration parameters to be used to control system operation, and for storing configuration information for FPGAs.

In a NAND flash memory, on the other hand, locations are written and read one page at a time, a page being typically 2 Kbytes. Read access to a given location would require reading the page containing the location, followed by selection of the required data, taking several microseconds. If all of the locations in a page are required, however, sequential reading is much faster, comparable in time to SRAM. Erasing a block and writing a page of data are significantly slower than SRAM access times. For example, the data sheet for the Micron Technology MT29F16G08FAA 16G bit IC specifies a random read time of 25µs, a, sequential read time of 25ns, a block erase time of 1.5ms, and a page write time of 220µs. Given their different access behavior, NAND flash memories have a different interface than SRAMs, making control circuits more involved. The advantage of NAND flash memory is that the density of storage cells is greater than that of NOR flash. Thus, NAND flash chips are better suited to applications in which large amounts of data must be stored cheaply. One of the largest applications of NAND flash memories is in memory cards for consumer devices such as digital cameras. They are also used in USB memory sticks for general purpose computers.

ACKNOWLEDGMENT

I would like to express my sincere thanks to Prof.Pavan Kumar E (Sai Vidya Institute of Technology) for having guided us in this whole research and finally help us to complete this paper as a whole.

REFERENCES

[1] http://www.rgcetpdy.ac.in/Notes/IT/II%20YEAR/DIGITAL%20SYSTEM%20DESIGN/Unit%204.pdf

[2]https://en.wikipedia.org/wiki/Dynamic_randomaccess_memory4

[3]http://www.staroceans.org/kernel-anddriver/Digital%20Design%20-%20An%20Embedded%20Systems%20Approach%20Using%20Verilog.pdf

[4]https://www.researchgate.net/profile/Nikola_Zlatanov3/publication/295550090_Computer_Memory_Applications_and_Management/links/56cb9e3c08ae5488f0db1882/Computer-Memory-Applicationsand-Management.pdf