#5 : Basic Components

Computer Architecture 2021/2022

Ricardo Rocha

Computer Science Department, Faculty of Sciences, University of Porto

Slides based on the book

'Computer Organization and Design, The Hardware/Software Interface, 5th Edition

David Patterson and John Hennessy, Morgan Kaufmann'

Sections B.1 – B.9

Digital Electronics

The **electronics inside a modern computer are digital**. Digital electronics operate with only two voltage levels of interest: a high voltage and a low voltage. All other voltage values are temporary and occur while transitioning between the values.

To simplify, rather than refer to voltage levels, we will talk about **signals that are (logically) true (or 1)**, or **signals that are (logically) false (or 0)**.

The fact that computers are digital is also a key reason they use **binary numbers**, since a binary system matches the underlying abstraction inherent in the electronics.

Combinational and Sequential Logics

Logic blocks are categorized as one of two types:

- **Blocks without memory** are called **combinational** – the output depends only on the current input
- **Blocks with memory** are called **sequential** – the output can depend on both the inputs and the value stored in memory, which is called the **state of the block**

Because a combinational logic block contains no memory, it can be completely specified by defining the output values for each possible set of input values. Such a description is normally given as a **truth table**.

- For a logic block with N inputs, there are 2^N entries in the truth table, since each entry specifies the value of all the outputs for that particular input combination

Basic Logic Gates

Logic blocks are built from **gates that implement basic logic functions**. Any logical function can be constructed using **AND gates, OR gates, and inverters**.

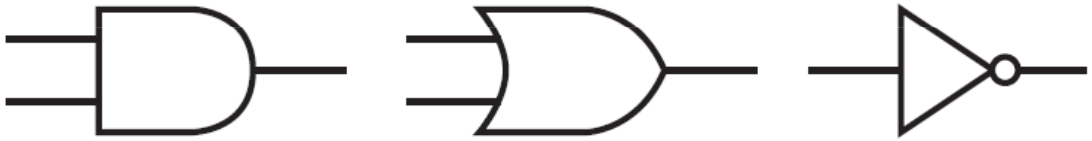


FIGURE B.2.1 Standard drawing for an AND gate, OR gate, and an inverter, shown from left to right. The signals to the left of each symbol are the inputs, while the output appears on the right. The AND and OR gates both have two inputs. Inverters have a single input.

Basic Logic Gates

Rather than draw inverters explicitly, a common practice is to **add bubbles to the inputs or outputs of a gate to cause the logic value on that input line or output line to be inverted**.

In fact, all logic functions can be constructed with only a single gate type, if that gate is inverting. The two common inverting gates are called NOR and NAND and correspond to inverted OR and AND gates, respectively. **NOR and NAND gates are called universal** since any logic function can be built using this one gate type.

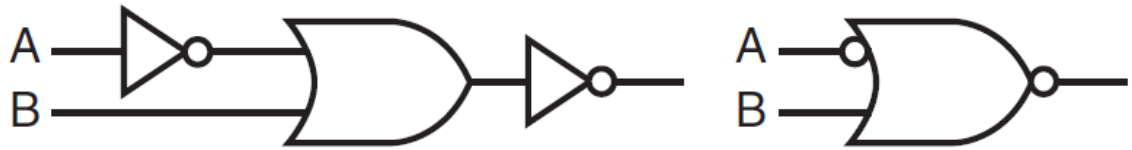
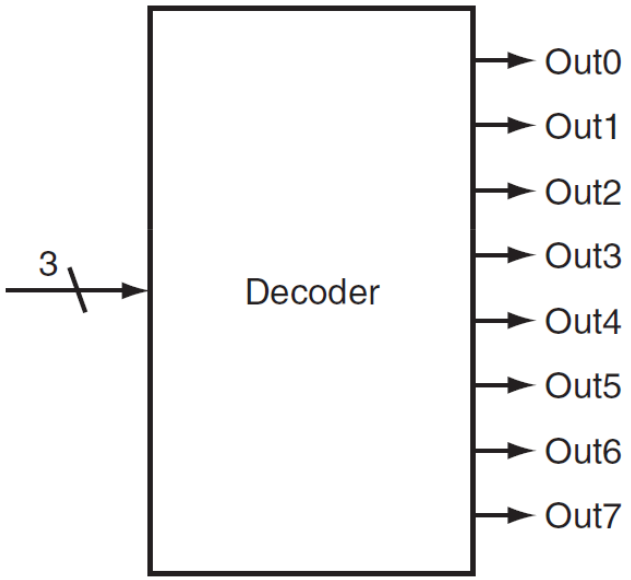


FIGURE B.2.2 Logic gate implementation of $\overline{\overline{A} + B}$ using explicit inverts on the left and bubbled inputs and outputs on the right. This logic function can be simplified to $A \cdot \overline{B}$

Decoder

A decoder is a logic block that has an N-bit input and 2^N outputs, where **only one output is asserted for each input combination**.



a. A 3-bit decoder

Inputs			Outputs							
12	11	10	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b. The truth table for a 3-bit decoder

FIGURE B.3.1 A 3-bit decoder has 3 inputs, called 12, 11, and 10, and $2^3 = 8$ outputs, called Out0 to Out7. Only the output corresponding to the binary value of the input is true, as shown in the truth table. The label 3 on the input to the decoder says that the input signal is 3 bits wide.

Multiplexor

A multiplexor is a logic block that has an N-bit input and 1 output, where the **output is one of the inputs that is selected accordingly to a control value**.

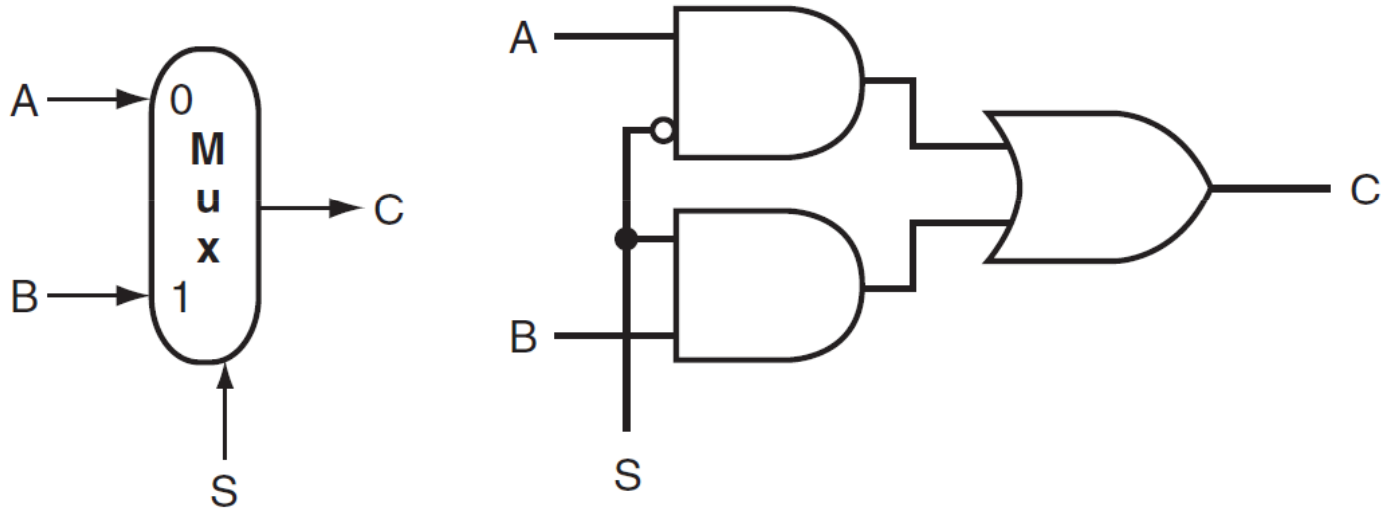


FIGURE B.3.2 A two-input multiplexor on the left and its implementation with gates on the right. The multiplexor has two data inputs (*A* and *B*), which are labeled *0* and *1*, and one selector input (*S*), as well as an output *C*. Implementing multiplexors in Verilog requires a little more work, especially when they are wider than two inputs. We show how to do this beginning on page B-23.

One-Bit Adder

An adder must have two inputs for the operands and a single-bit output for the sum. There must be a second output to pass on the carry, called **CarryOut**. Since the CarryOut from the neighbor adder must be included as an input, we need a third input called **CarryIn**.

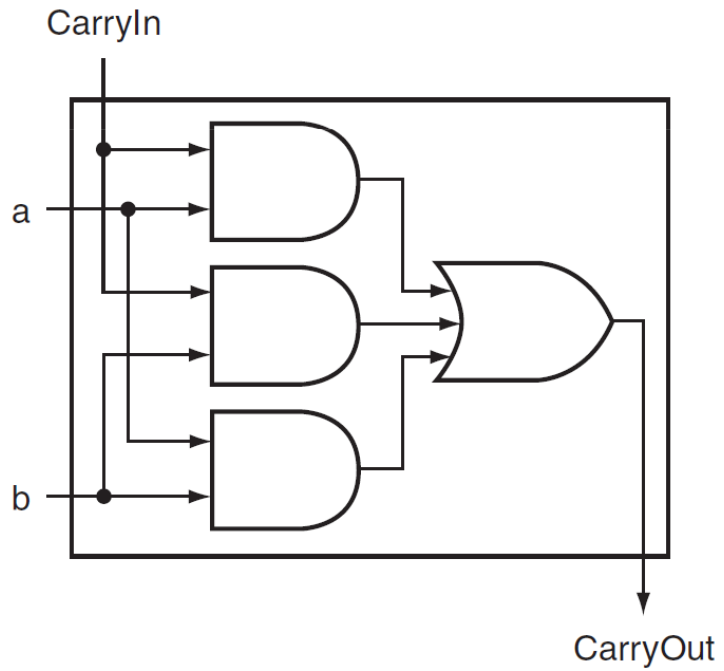


FIGURE B.5.5 Adder hardware for the CarryOut signal. The rest of the adder hardware is the logic for the Sum output given in the equation on this page.

One-Bit ALU

The device that performs the arithmetic operations like addition, subtraction or logical operations is the **arithmetic logic unit (ALU)**.

The 1-bit ALU for AND, OR and addition is implemented with a multiplexor that selects ‘a AND b’, ‘a OR b’ or ‘a + b’, depending on whether the value of Operation is 0, 1 or 2.

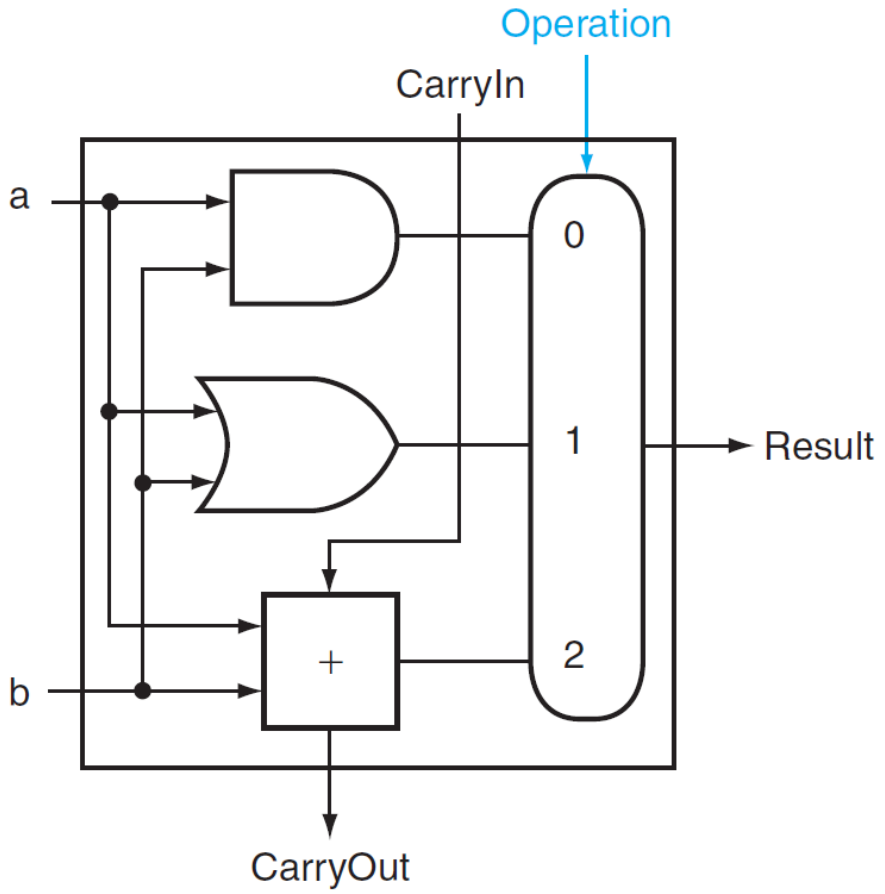


FIGURE B.5.6 A 1-bit ALU that performs AND, OR, and addition

32-Bit ALU

A 32-bit ALU is created by connecting 32 1-bit ALUs together and by propagating the CarryOut from 1-bit adder to the CarryIn of the next more significant 1-bit adder.

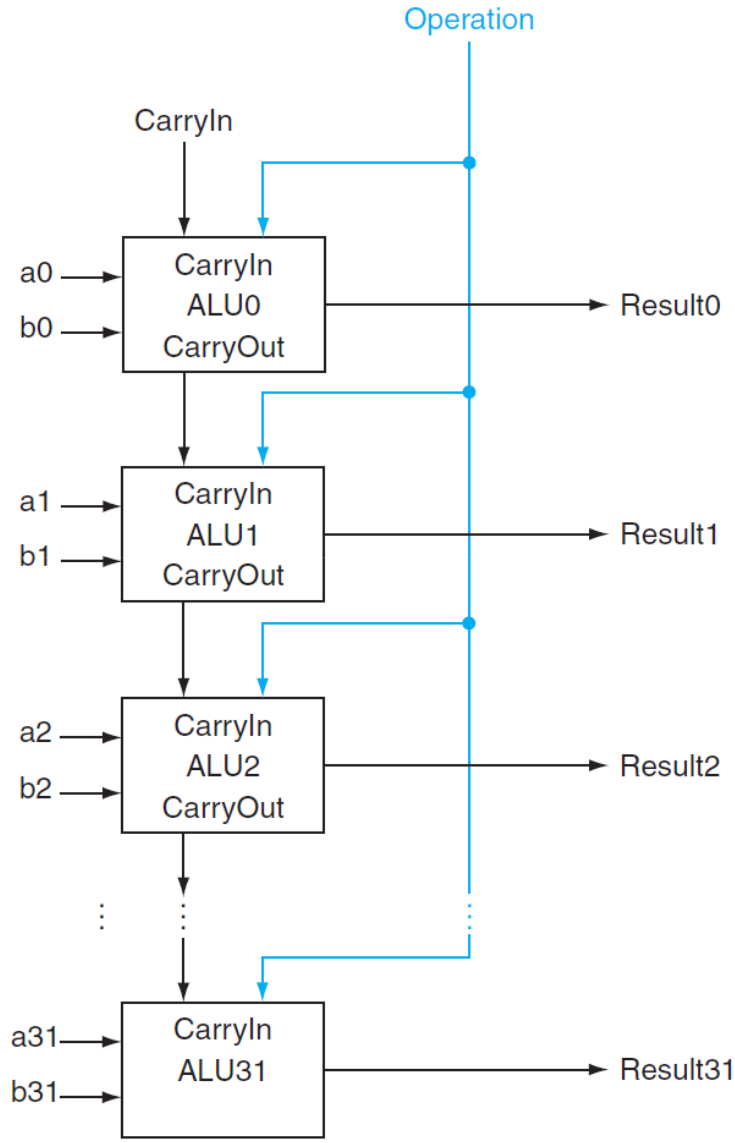


FIGURE B.5.7 A 32-bit ALU constructed from 32 1-bit ALUs. CarryOut of the less significant bit is connected to the CarryIn of the more significant bit. This organization is called ripple carry.

Edge-Triggered Clocking

Clocks are needed in sequential logic **to decide when an element that contains state should be updated**.

A clock is simply a **free-running signal with a fixed cycle time** (or clock period) divided into two portions: when the clock is high and when the clock is low. Here, we use only **edge-triggered clocking**, which means that all **state changes occur on a clock edge**.

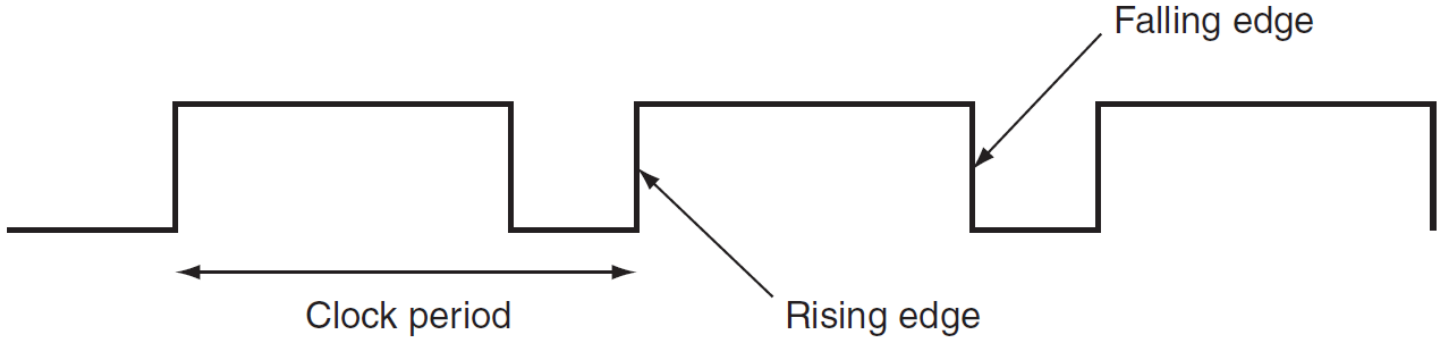


FIGURE B.7.1 A clock signal oscillates between high and low values. The clock period is the time for one full cycle. In an edge-triggered design, either the rising or falling edge of the clock is active and causes state to be changed.

Edge-Triggered Clocking

In an edge-triggered clocking, **either the rising or the falling edge of the clock is active** and causes the **state elements to only change on the active clock edge**. The choice of which edge is active is influenced by the implementation technology and does not affect the concepts involved in designing the logic.

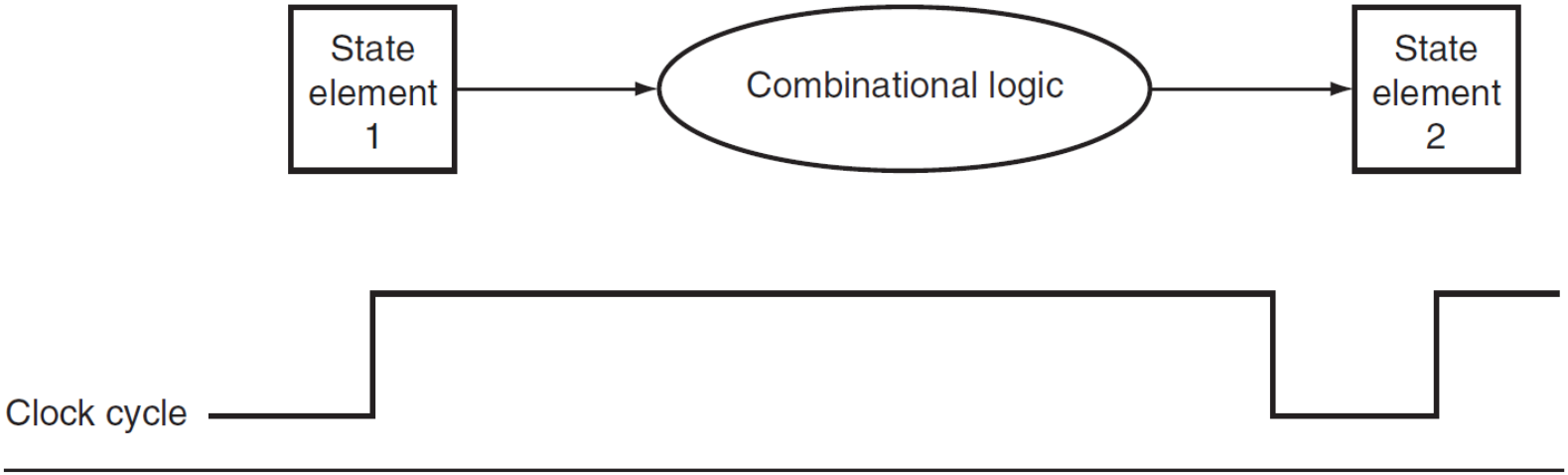


FIGURE B.7.2 The inputs to a combinational logic block come from a state element, and the outputs are written into a state element. The clock edge determines when the contents of the state elements are updated.

Edge-Triggered Clocking

To ensure that the values written into the state elements on the active clock edge are valid, the clock must have a **long enough period so that all the signals in the combinational logic block stabilize**, and then the clock edge samples those values **for storage in the state elements**. This constraint sets a lower bound on the length of the clock period, which must be long enough for all state element inputs to be valid.

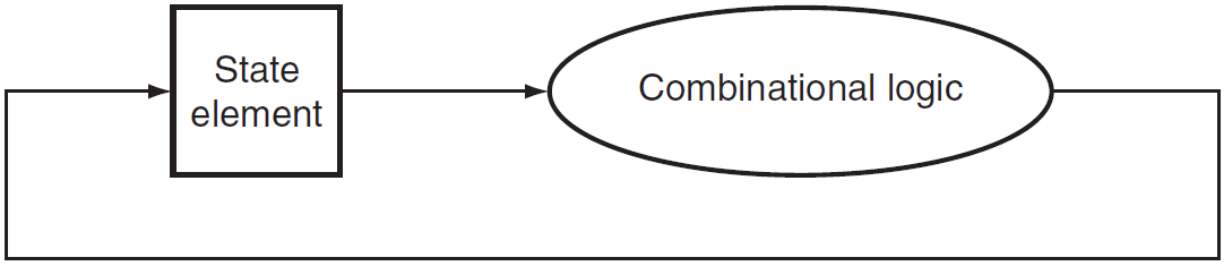


FIGURE B.7.3 An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could lead to undetermined data values. Of course, the clock cycle must still be long enough so that the input values are stable when the active clock edge occurs.

S-R Latch – Unclocked Memory Cell

The S-R (set-reset) latch is the simplest type of memory cell as it does **not have any clock input**. It is built from a pair of NOR gates where the outputs represent the value of the stored state (Q) and its complement. State changes when S or R are turned on and remains unaltered when both S and R are off. State may be undefined if both S and R are turned on simultaneously.

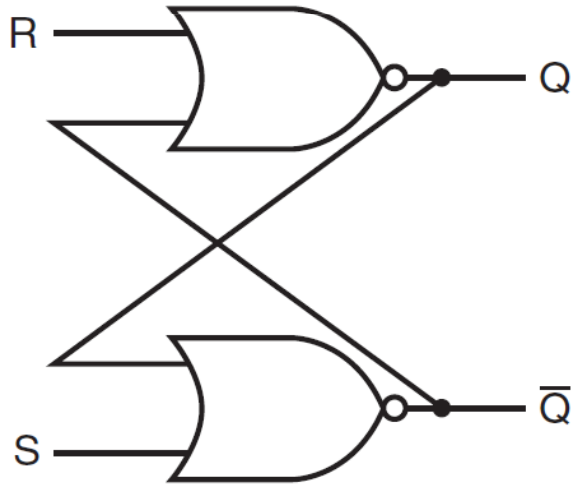


FIGURE B.8.1 A pair of cross-coupled NOR gates can store an internal value. The value stored on the output Q is recycled by inverting it to obtain $\overline{Q}$ and then inverting $\overline{Q}$ to obtain Q. If either R or $\overline{Q}$ is asserted, Q will be deasserted and vice versa.

D Latch – Transparent Clocked Memory Cell

In a D latch, the internal memory state is changed whenever the appropriate inputs change and the **clock is asserted**. The inputs are the data value to be stored (D) and a clock signal (C) that indicates when the latch should read the value on D and store it. The outputs are simply the value of the internal state (Q) and its complement.

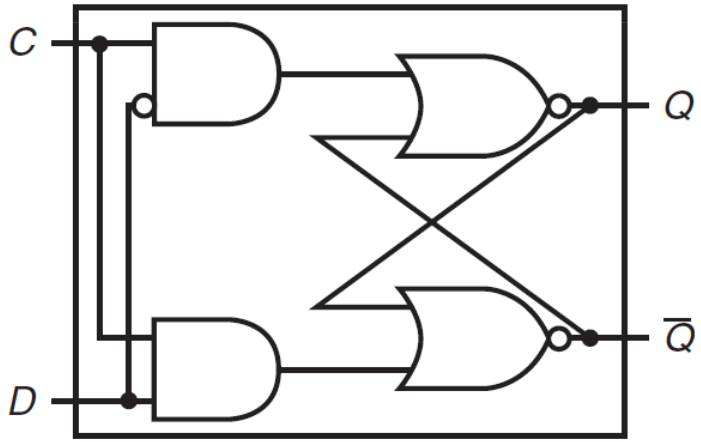


FIGURE B.8.2 A D latch implemented with NOR gates. A NOR gate acts as an inverter if the other input is 0. Thus, the cross-coupled pair of NOR gates acts to store the state value unless the clock input, *C*, is asserted, in which case the value of input *D* replaces the value of *Q* and is stored. The value of input *D* must be stable when the clock signal *C* changes from asserted to deasserted.

D Flip-Flop – Edge-Triggered Clocked Memory Cell

The D flip-flop is the basic building block for memory cells since its **output only changes on the clock edge**. A D flip-flop is constructed from a pair of D latches and can be built so that it triggers on either the rising or falling clock edge. The output is stored when the clock edge occurs.

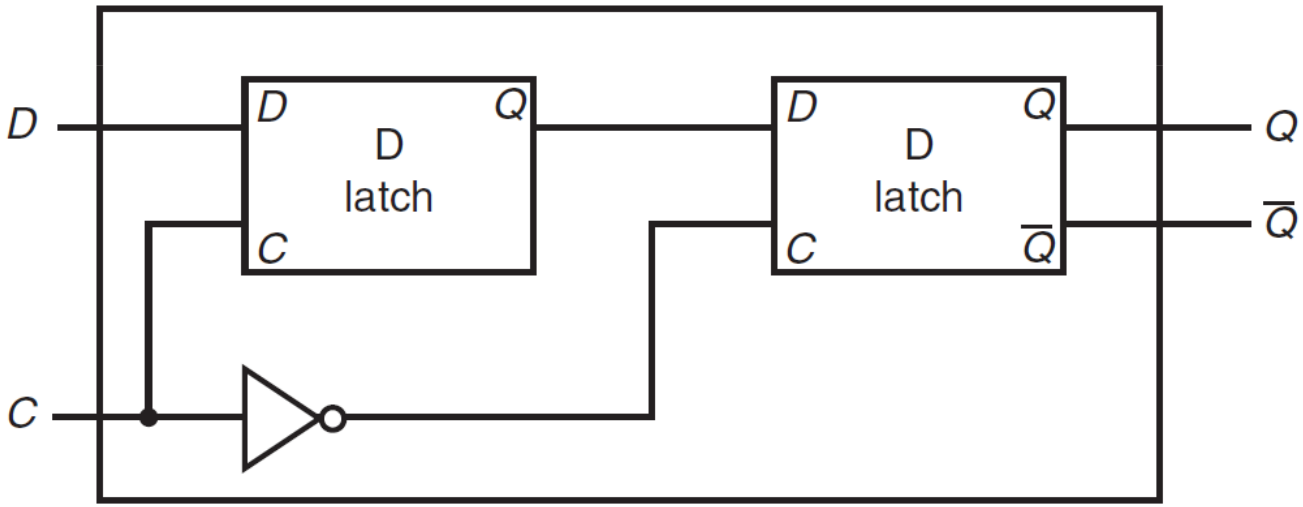


FIGURE B.8.4 A D flip-flop with a falling-edge trigger. The first latch, called the master, is open and follows the input *D* when the clock input, *C*, is asserted. When the clock input, *C*, falls, the first latch is closed, but the second latch, called the slave, is open and gets its input from the output of the master latch.

D Latch and D Flip-Flop Operations

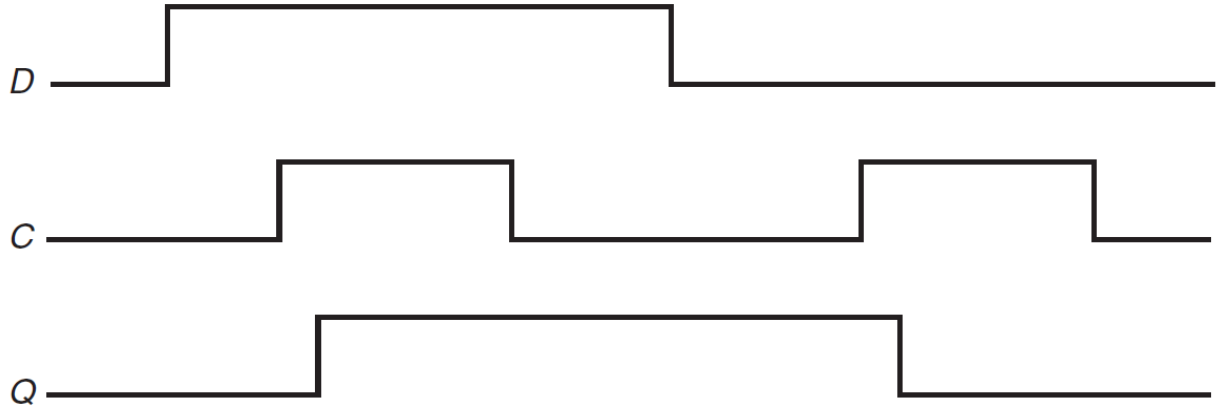


FIGURE B.8.3 Operation of a D latch, assuming the output is initially deasserted. When the clock, *C*, is asserted, the latch is open and the *Q* output immediately assumes the value of the *D* input.

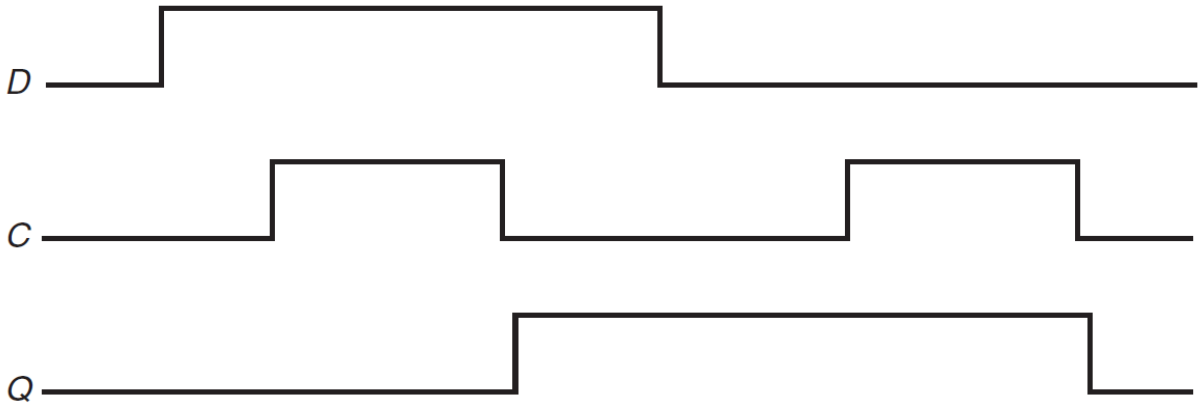


FIGURE B.8.5 Operation of a D flip-flop with a falling-edge trigger, assuming the output is initially deasserted. When the clock input (*C*) changes from asserted to deasserted, the *Q* output stores the value of the *D* input. Compare this behavior to that of the clocked D latch shown in Figure B.8.3. In a clocked latch, the stored value and the output, *Q*, both change whenever *C* is high, as opposed to only when *C* transitions.

Registers

We can use an **array of D flip-flops to build a register** that can hold a multibit datum, such as a byte or word.

A set of registers (or register file) can be then implemented with an **array of registers**, each built from an array of D flip-flops, and several **logic read/write ports**, one for each read/write operation.

Because reading a register does not change any state, we need only a register number as input and the output will be the data contained in that register. For writing a register we need a register number, the data to write, and a clock that controls the writing into the register.

Reading Registers

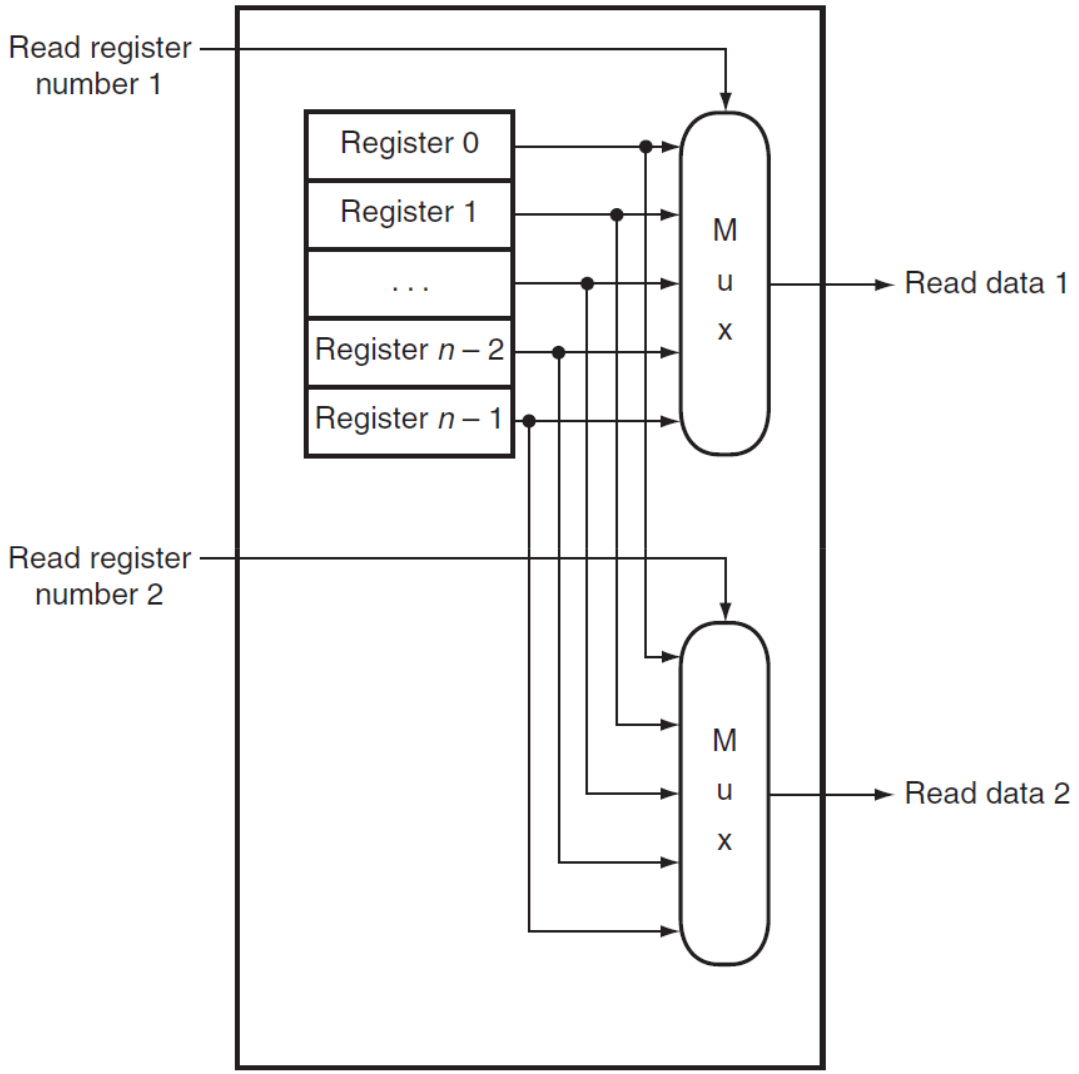


FIGURE B.8.8 The implementation of two read ports for a register file with *n* registers can be done with a pair of n-to-1 multiplexors, each 32 bits wide. The register read number signal is used as the multiplexor selector signal. Figure B.8.9 shows how the write port is implemented.

Writing Registers

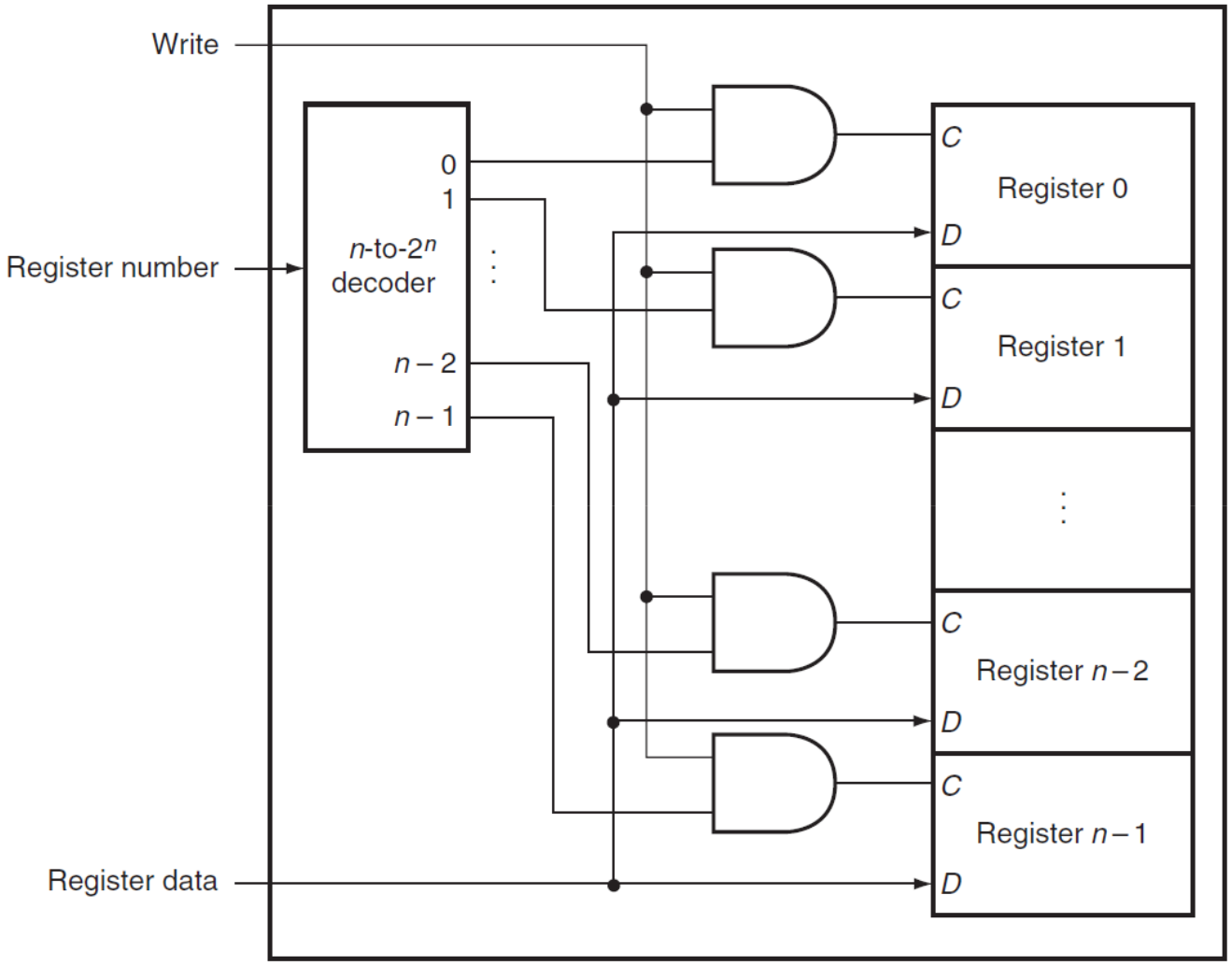


FIGURE B.8.9 The write port for a register file is implemented with a decoder that is used with the write signal to generate the C input to the registers. All three inputs (the register number, the data, and the write signal) will have setup and hold-time constraints that ensure that the correct data is written into the register file.

Register File

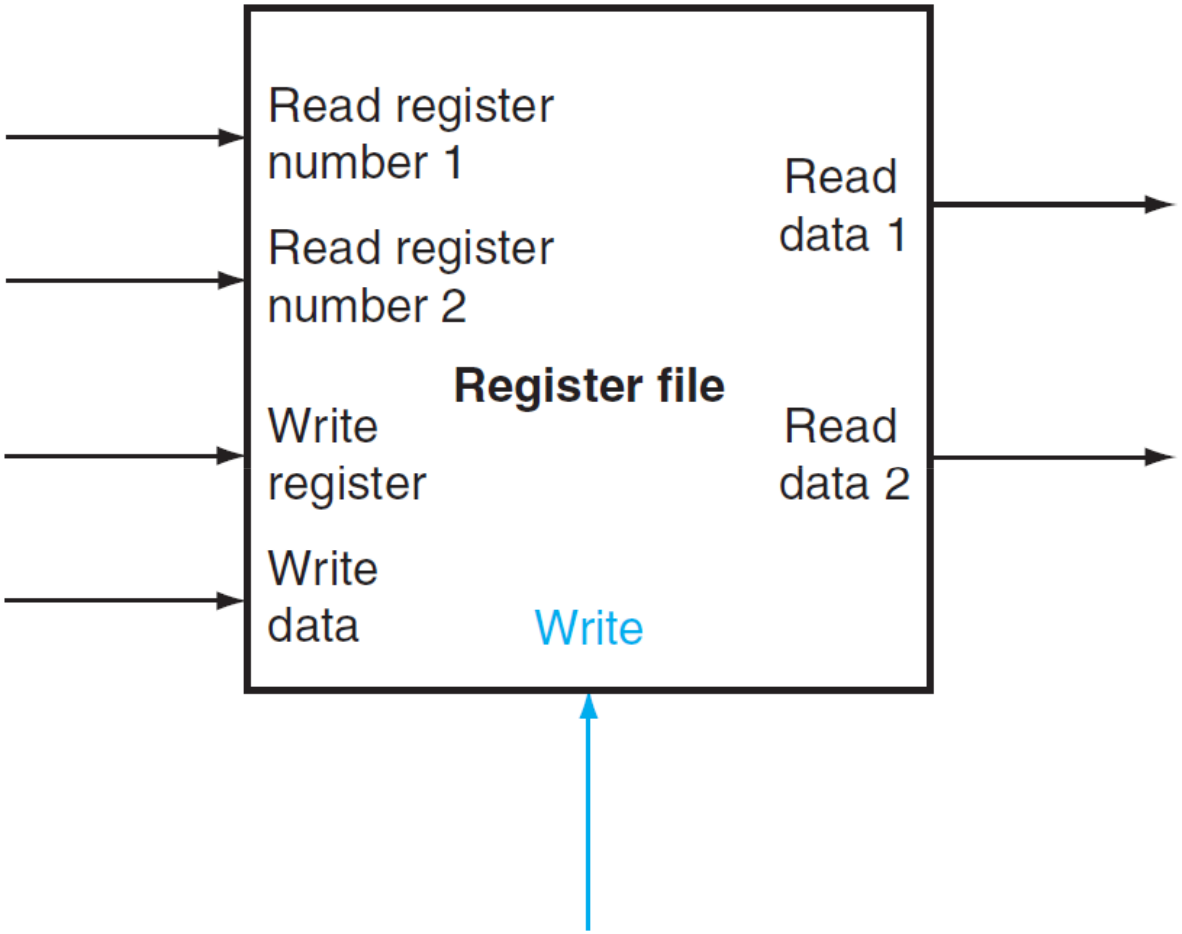
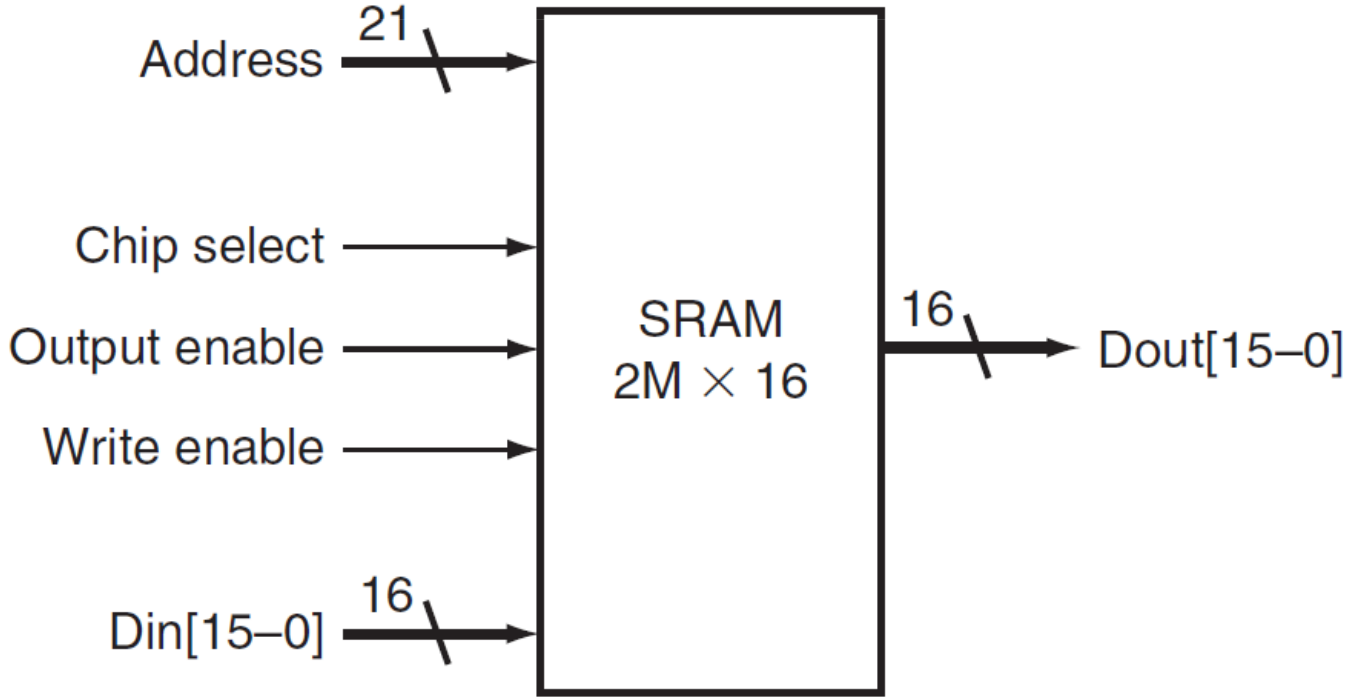


FIGURE B.8.7 A register file with two read ports and one write port has five inputs and two outputs. The control input Write is shown in color.

SRAM – Static Random Access Memory

SRAMs are simply **memory arrays integrated circuits**. An SRAM chip has a specific configuration in terms of the number of addressable locations, as well as the width of each addressable location.

- A 2M × 16 SRAM provides 2M entries, each of which is 16bits wide – it thus requires 21 address lines ($2M = 2^{21}$), a 16-bit data input line and a 16-bit output line



SRAM – Static Random Access Memory

Large SRAMs cannot be built in the same way as a register file because the usage of a giant multiplexor/decoder is totally impractical. Instead, large memories are implemented with **shared output lines**, which multiple memory cells in the memory array can assert.

For example, in a 4M × 8 SRAM, we would need a 22-to-4M decoder and 4M word lines (required to enable the individual flip-flops). To circumvent this problem, large memories are **organized as rectangular arrays** and use a **two-step decoding process**.

SRAM – Static Random Access Memory

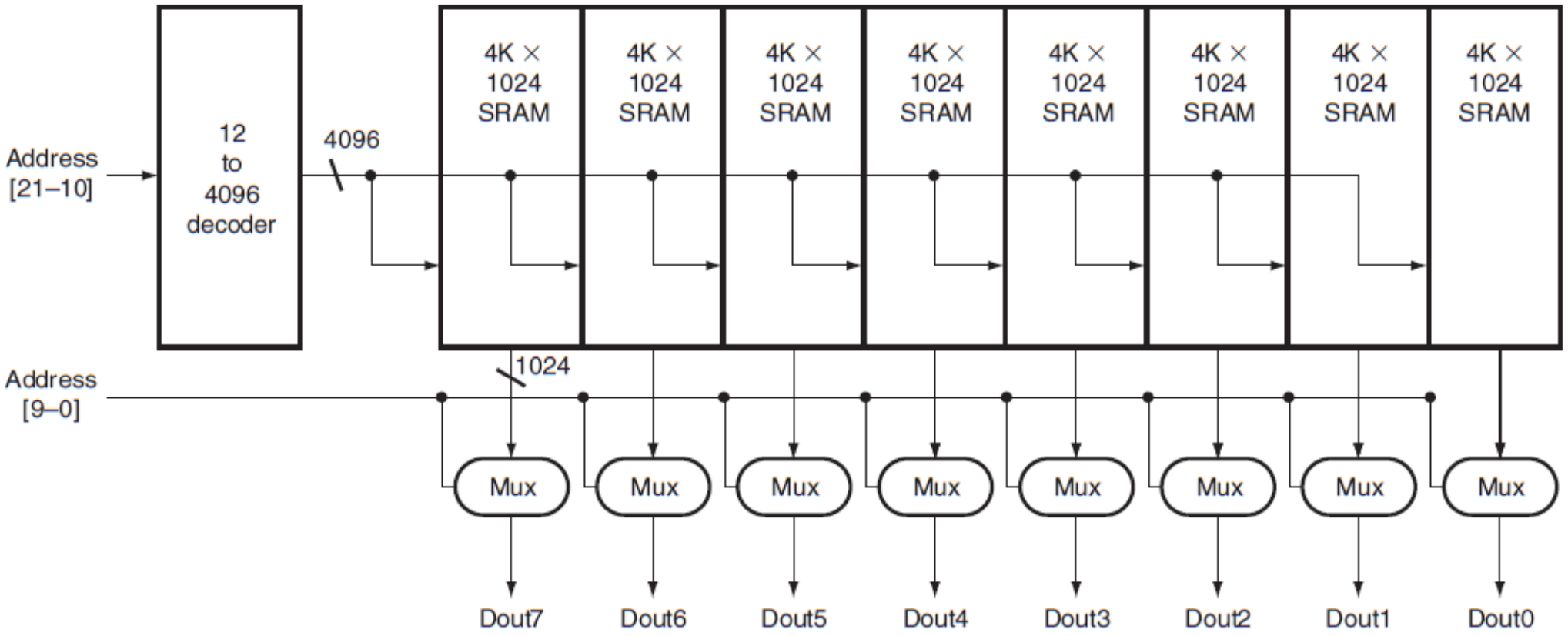


FIGURE B.9.4 Typical organization of a 4M × 8 SRAM as an array of 4K × 1024 arrays. The first decoder generates the addresses for eight 4K × 1024 arrays; then a set of multiplexors is used to select 1 bit from each 1024-bit-wide array. This is a much easier design than a single-level decode that would need either an enormous decoder or a gigantic multiplexor. In practice, a modern SRAM of this size would probably use an even larger number of blocks, each somewhat smaller.

DRAM – Dynamic Random Access Memory

In SRAM, the cell values can be kept indefinitely as long as power is applied. In DRAM, a cell value is stored as a **charge in a capacitor**.

Because DRAMs store the charge on a capacitor, it cannot be kept indefinitely and must **periodically be refreshed** (that is why this memory is called **dynamic**).

The charge can be kept for several milliseconds, which might correspond to close to a million clock cycles. Today, single-chip memory controllers often handle the refresh function independently of the processor.

DRAM – Dynamic Random Access Memory

A single transistor is then used to access the stored charge, either to read its value or to overwrite the charge stored there.

Because DRAMs use only a single transistor per bit of storage (SRAMs require four to six transistors per bit), they are much **denser and cheaper per bit**. On the other hand, the two-level addressing scheme, combined with the internal circuit, makes DRAM **access times much longer** (by a factor of 5–10) than SRAM access times.

The much lower cost per bit makes DRAM the choice for main memory, while the faster access time makes SRAM the choice for caches.

SSRAMs, SDRAMs and DDRRAMs – Synchronous RAMs

The key capability provided by **synchronous RAMs** is the ability to **transfer a burst of data from a series of sequential addresses within an array or row**. The burst is defined by a starting address, supplied in the usual fashion, and a burst length.

The speed advantage of synchronous RAMs comes from the ability to transfer the bits in the burst **without having to specify additional address bits**. Instead, a clock is used to transfer the successive bits in the burst, which **significantly improves the overall data transfer rate**. A refined form of SDRAMS are the DDRRAMs (Double Data Rate RAMs), which **transfer data on both the rising and falling edge of an externally supplied clock**.