CSC 3210 Computer organization and programming

Appendix A && Chapter 3

Chunlan Gao

Georgia State University

Appendix A

The **Basics** of Logic Design

Gates, Truth Tables, and Logic Equations

Digital electronics operate with two level of valtage:

High voltage (represents logic **1**, "true", or "asserted").

Low voltage (represents logic **0**, "false", or "deasserted")

The values 0 and 1 are called complements or inverses of one another.

We use 1/0 in our logic instructions:

AND, OR, NOR …..

Logic blocks

- Logic blocks are categorized as one of two types, depending on whether they contain memory

➢ combinational: Blocks without memory are called combinational; the output of a combinational block depends only on the current input.

➢ state : In blocks with memory, the outputs can depend on both the inputs and the value stored in memory, which is called the state of the logic block.

Truth Table Example

Truth table is a table that represents the output of a logical expression for all possible input combinations.

Consider a logic function with three inputs, A, B, and C, and three outputs, D, E, and F. The function is defined as follows: D is true if at least one input is true, E is true if exactly two inputs are true, and F is true only if all three inputs are true. Show the truth table for this function.

The truth table will contain $2^3 = 8$ entries. Here it is:

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Boolean Algebra

Another approach is to express the logic function with logic equations all the variables have the values 0 or 1 and, in typical formulations, there are three operators:

- OR +, as in A + B. The OR operation is also called a logical sum,
- AND · , as in A · B. The AND operator is also called logical product,
- The unary operator NOT is written as $\overline{A}$

Boolean Algebra

- ■ Identity law: A + 0 = A and A · 1 = A
- ■ Zero and One laws: A + 1 = 1 and A · 0 = 0
- ■ Inverse laws: A + $\overline{A}$ =1 and A · $\overline{A}$ =0
- ■ Commutative laws: A + B = B + A and A · B = B · A
- ■ Associative laws: A + (B + C) = (A + B) + C and A · (B · C) = (A · B) · C
- ■ Distributive laws: A · (B + C) = (A · B) + (A · C) and
- A + (B · C) = (A + B) · (A + C)

Example

The truth table will contain $2^3 = 8$ entries. Here it is:

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

D =A+ B+ C

F = A• B •C

E= (A• B •$\bar{C}$)+(A• C •$\bar{B}$)+(B• C •$\bar{A}$)

Gates

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

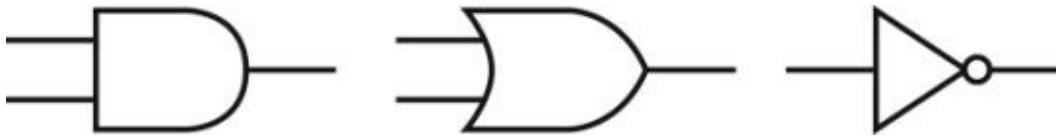


Figure A.2.1 Standard drawing for an AND gate, OR gate, and an inverter, shown from left to right. The signals to the left of each symbol are the inputs, while the output appears on the right. The AND and OR gates both have two inputs. Inverters have a single input.

The inputs or outputs of a gate to cause the logic value on that input line or output line to be inverted.

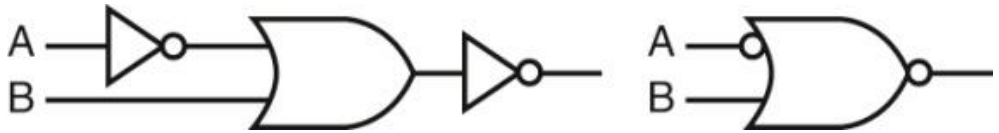
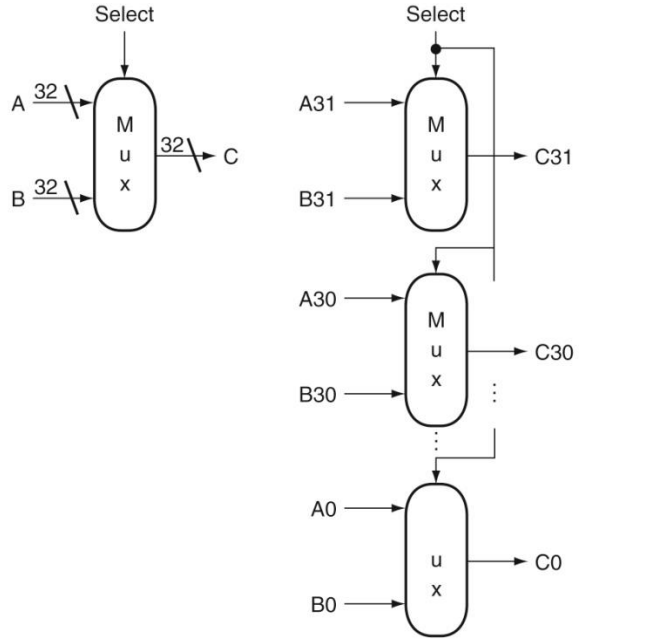


Figure A.2.2 Logic gate implementation of *A* + *B* *using explicit inverts on the left and* bubbled inputs and outputs on the right. This logic function can be simplified to *A B*

Multiplexors

A **Multiplexer (MUX)** is a **combinational logic circuit** that selects **one input from multiple inputs** and forwards it to the output. It functions like an **electronic switch**, where the selection is controlled by **selector (control) signals**.



a. A 32-bit wide 2-to-1 multiplexor

b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors

$$C = (A \cdot S)+(B \cdot S)$$

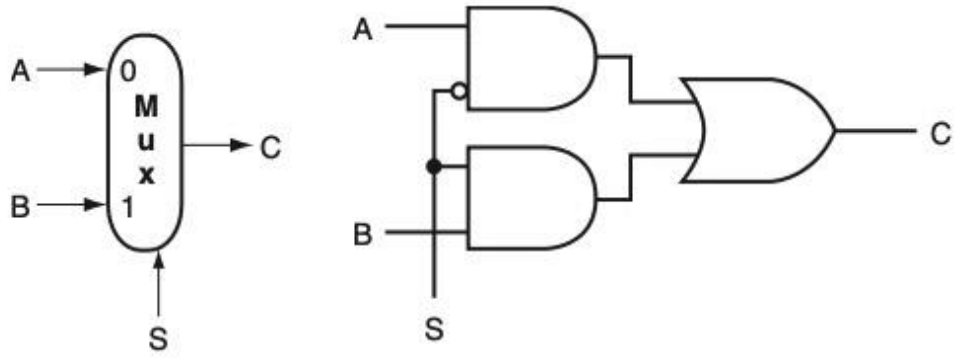


FIGURE A.3.2 A two-input multiplexor on the left and its implementation with gates on the right. The multiplexor has two data inputs (*A* and *B*), which are labeled *0* and *1*, and one selector input (*S*), as well as an output *C*. Implementing multiplexors in Verilog requires a little more work,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

Two-Level Logic and PLAs

Sum of products A form of logical representation that employs a logical sum (OR) of products (terms joined using the AND operator).

Show the sum-of-products representation for the following truth table for D

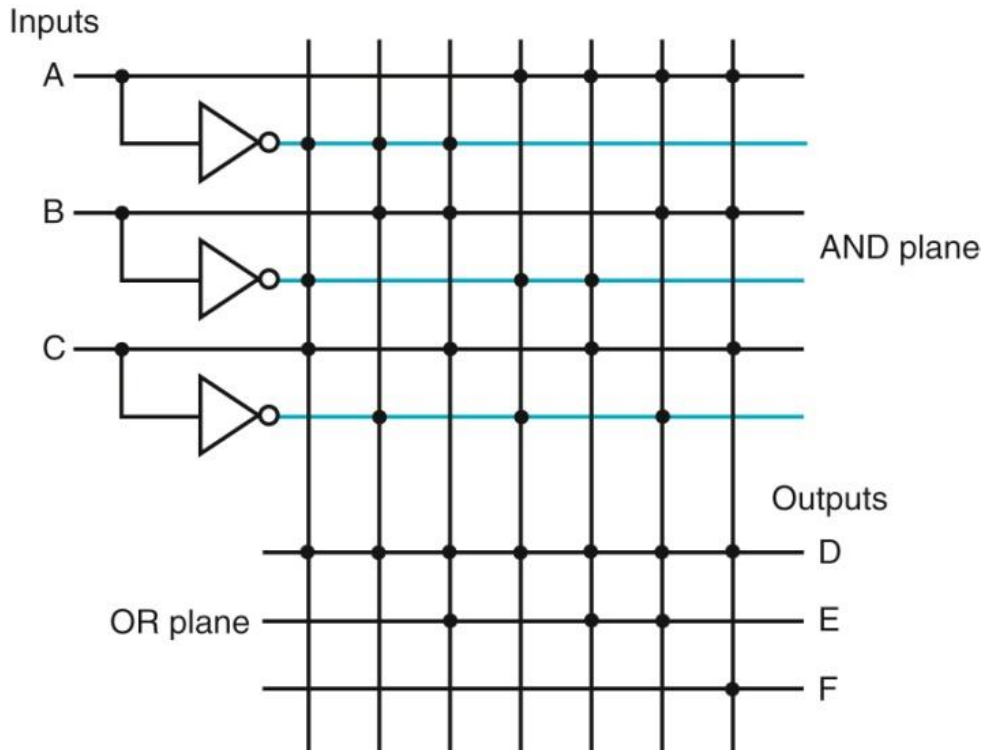
Inputs		Outputs	
A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

When D = 1 we have combinations :

$$\bar{A} \cdot \bar{B} \cdot C$$
$$\bar{A} \cdot B \cdot C$$
$$A \cdot \bar{B} \cdot \bar{C}$$
$$A \cdot B \cdot C$$

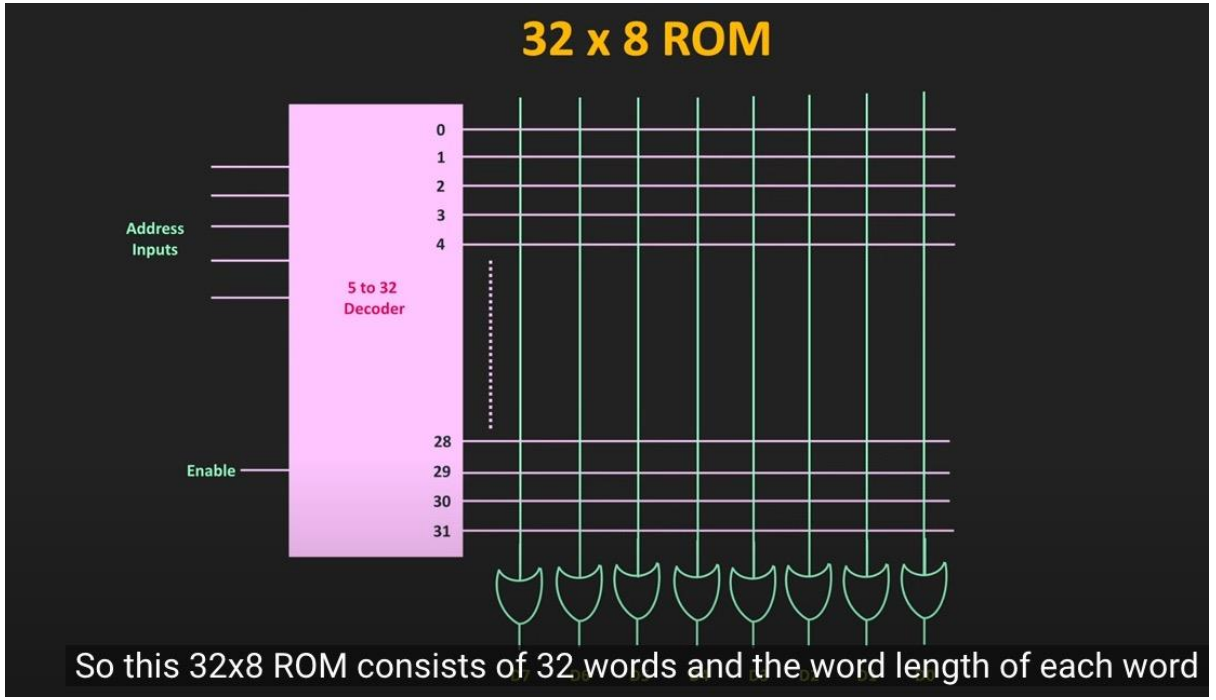
$$D = (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot \bar{C}) + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot B \cdot C)$$

PLA (Programmable Logic Array)



ROMs

- Another form of structured logic that can be used to implement a set of logic functions is a read-only memory (ROM).



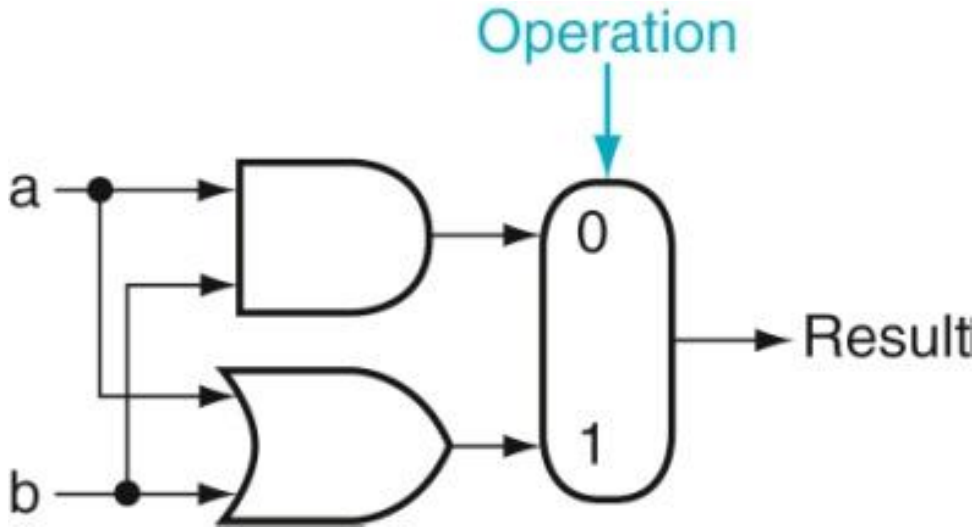
Constructing a Basic Arithmetic Logic Unit

The arithmetic logic unit (ALU) is the brawn of the computer, the device that performs the arithmetic operations like addition and subtraction or logical operations like AND and OR.

- This section constructs an ALU from four hardware building blocks (AND and OR gates, inverters, and multiplexors) and illustrates how combinational logic works.

1-bit logical unit

Figure A.5.1 The 1-bit logical unit for AND and OR



Adder

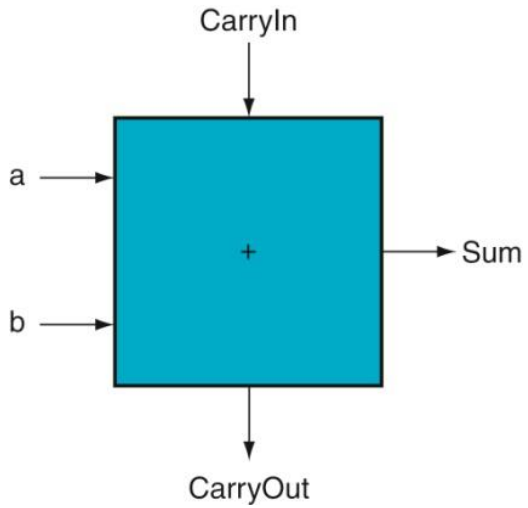


Figure A.5.2 A 1-bit adder. This adder is called a full adder; it is also called a (3,2) adder because it
has three inputs and two outputs. An adder with only the a and b inputs is called a (2,2) adder or half-adder.

Inputs			Outputs		
a	b	CarryIn	CarryOut	Sum	Comments
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

FIGURE A.5.3 Input and output specification for a 1-bit adder.

Adder hardware for the CarryOut signal

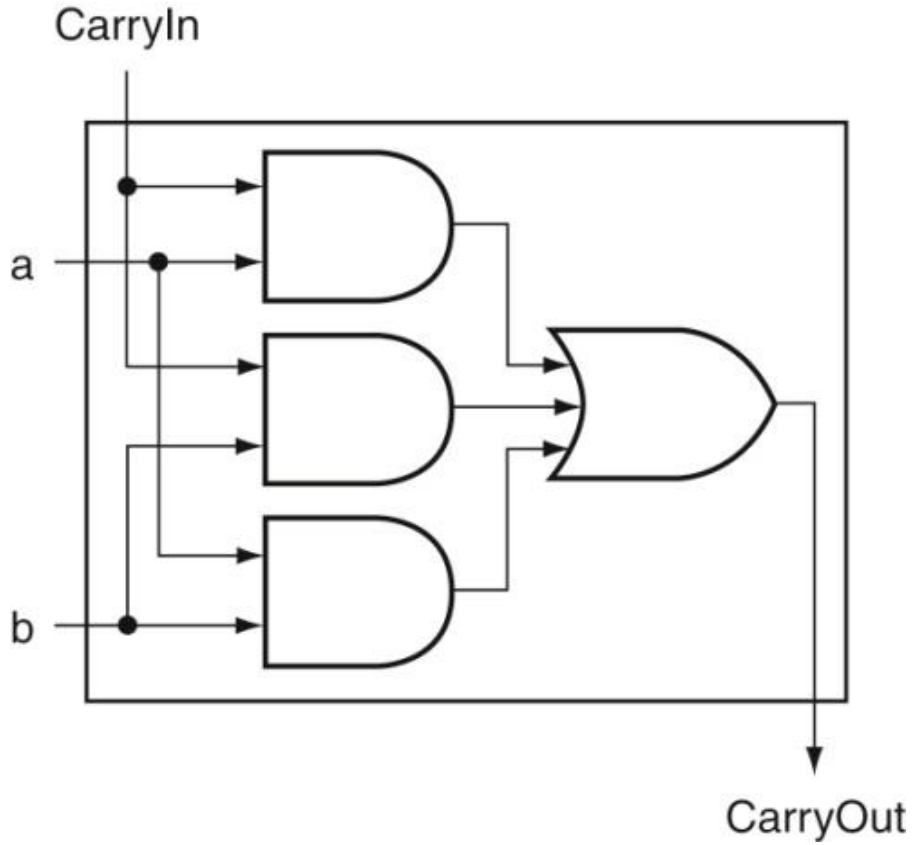


Figure A.5.5 Adder hardware for the CarryOut signal. The rest of the adder hardware is the logic for the Sum output given in the equation on this page.

1-Bit ALU

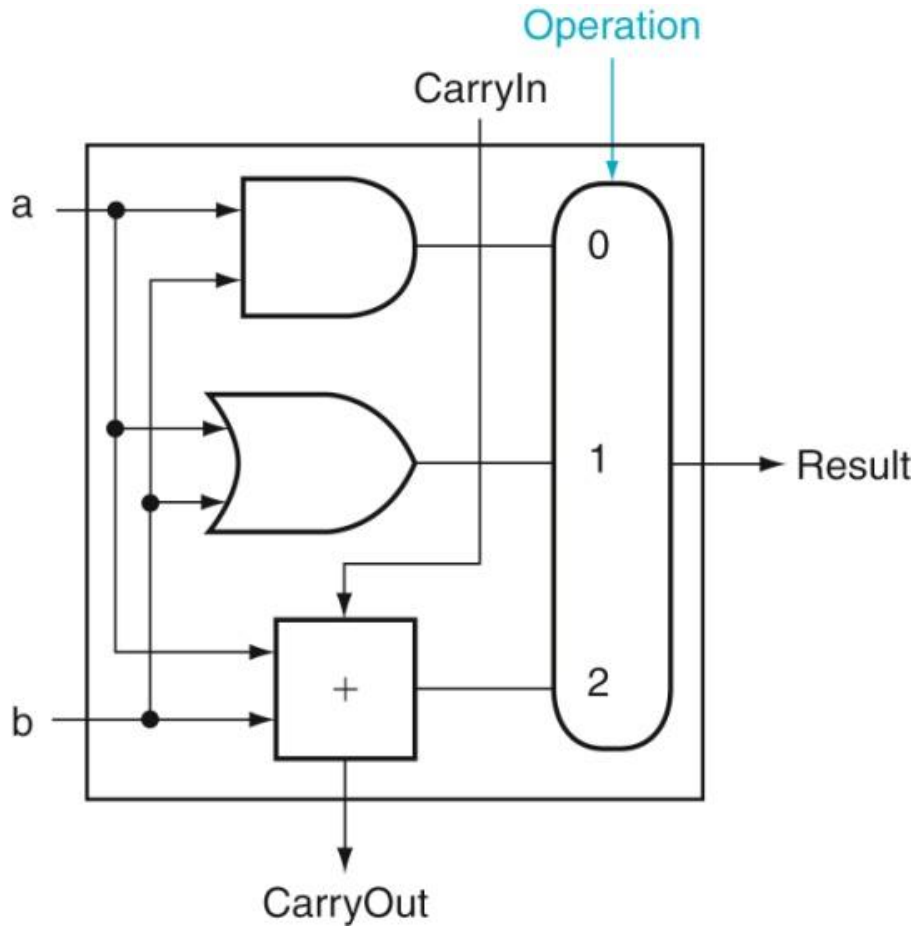


Figure A.5.6 A 1-bit ALU that performs AND, OR, and addition

A 32-bit ALU

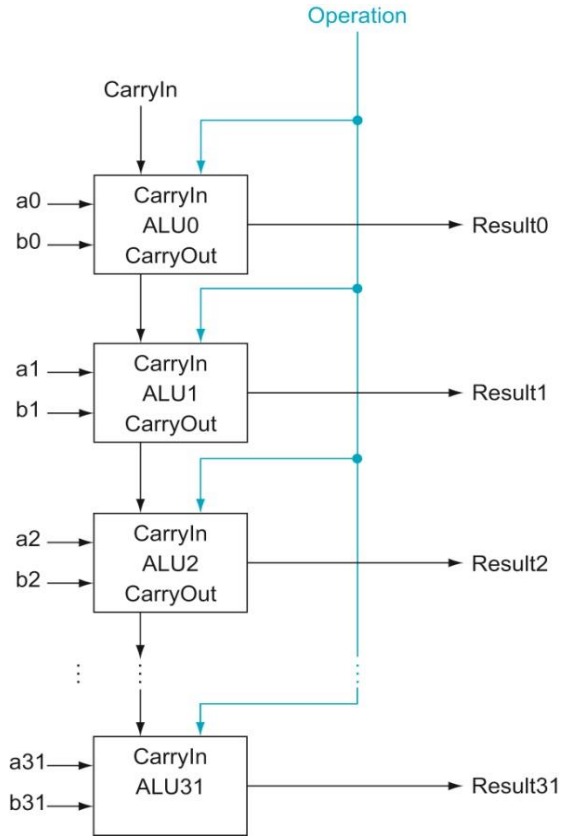


Figure A.5.7 A 32-bit ALU constructed from 32 1-bit ALUs. CarryOut of the less significant bit is connected to the CarryIn of the more significant bit. This organization is called ripple carry.

Clock

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.

Edge-triggered clocking A clocking scheme in which all state changes occur on a clock edge.

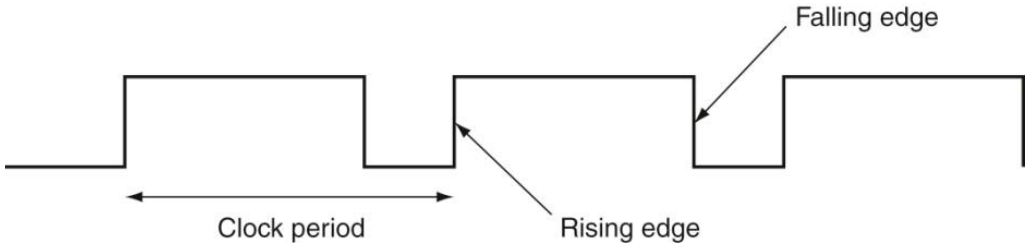


Figure A.7.1 A clock signal oscillates between high and low values. The clock period is the time for one full cycle. In an edge-triggered design, either the rising or falling edge of the clock is active and causes state to be changed.

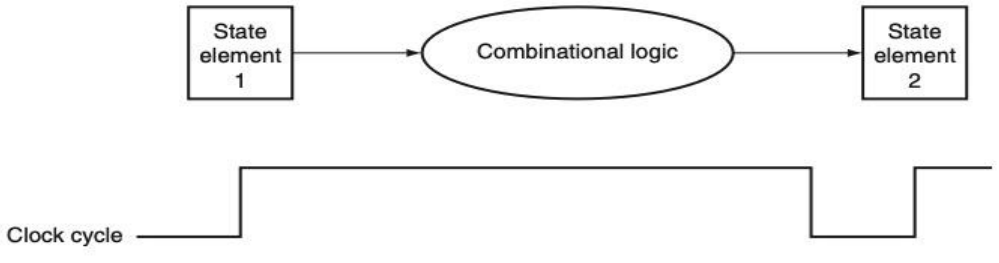


FIGURE A.7.2 The inputs to a combinational logic block come from a state element, and the outputs are written into a state element. The clock edge determines when the contents of the state elements are updated.

Register Files

- A register file consists of a set of registers that can be read and written by supplying a register number to be accessed.
- A register file can be implemented with a decoder for each read or write port and an array of registers built from D flip-flops.

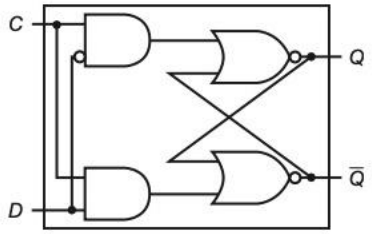
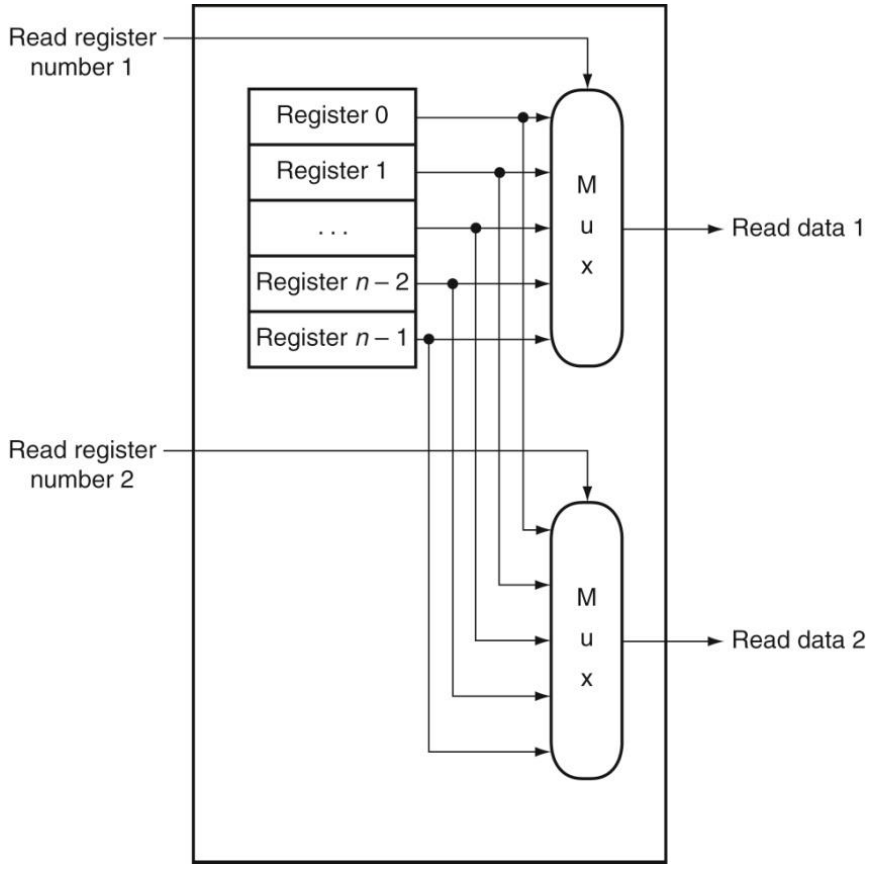


FIGURE A.8.2 A D latch implemented with NOR gates. A NOR gate acts as an inverter if the other input is 0. Thus, the cross-coupled pair of NOR gates acts to store the state value unless the clock input, *C*, is asserted, in which case the value of input *D* replaces the value of *Q* and is stored. The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

A D latch has two inputs and two outputs. The inputs are the data value to be stored (called D) and a clock signal (called C) that indicates when the latch should read the value on the D input and store it. The outputs are simply the value of the internal state (Q)

Two read ports



Registers (Register 0 to Register n−1n-1n−1)

•The register file consists of nnn registers.

•Each register stores **data** (e.g., 32-bit values in a typical CPU).

Read Register Selection

•**Read register number 1**: Specifies which register's data should be sent to **Read data 1**.

•**Read register number 2**: Specifies which register's data should be sent to **Read data 2**.

Multiplexers (MUX)

•Two **multiplexers (MUXes)** select the outputs.

•The first MUX selects **one register's value** for **Read data 1**.

•The second MUX selects **one register's value** for **Read data 2**.

Figure A.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 A.8.9 shows how the write port is implemented

Write to the register

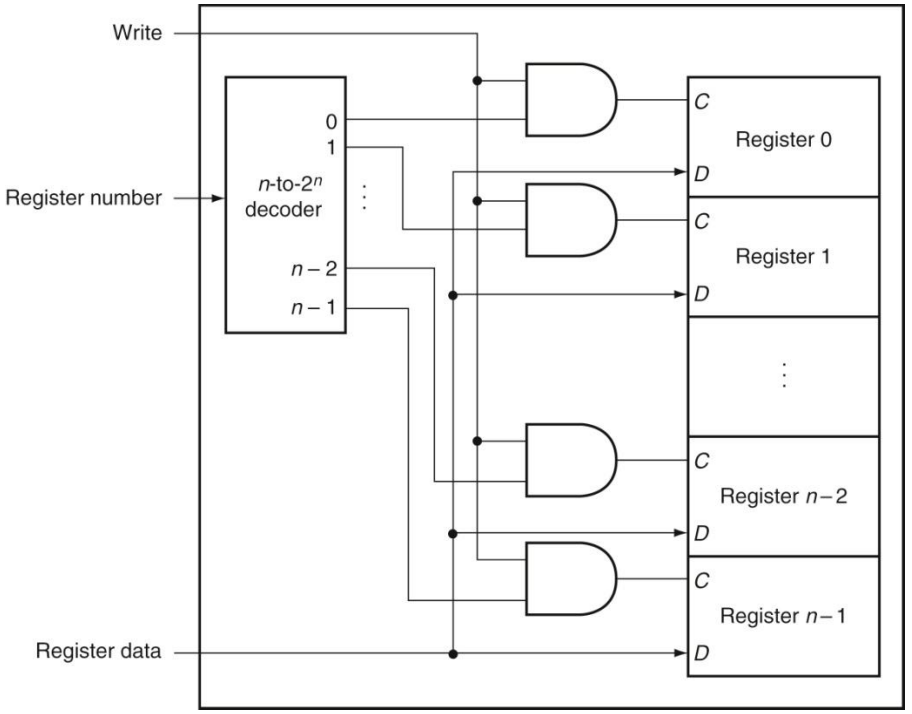


Figure A.8.9 The write port for a register file is implemented with a decoder that is used with the write signal to generate the C input to the registers. All three inputs (the register number, the data, and the write signal) will have setup and hold-time constraints that ensure that the correct data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

Finite State Machine

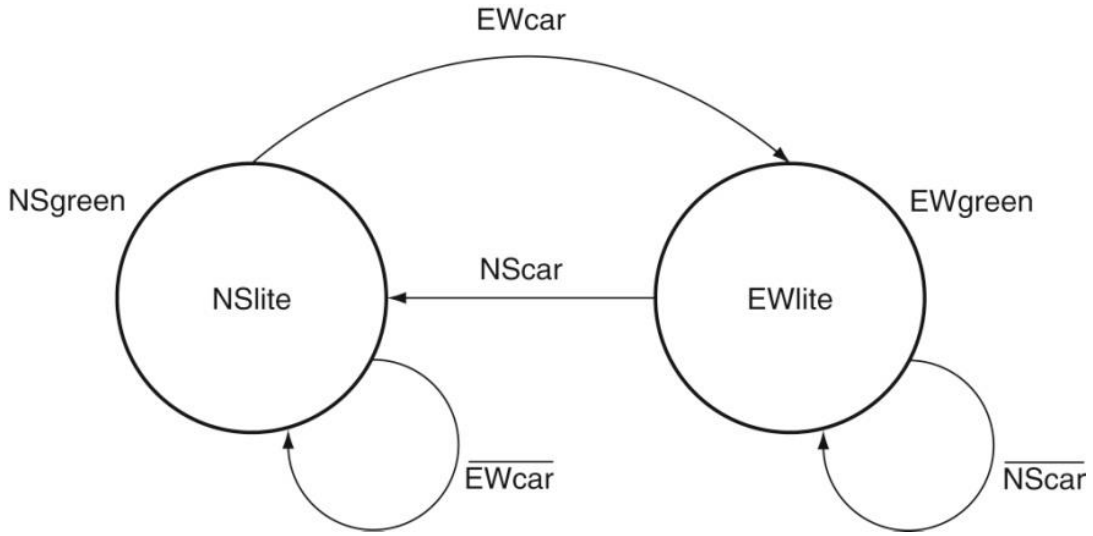


Figure A.10.2 The graphical representation of the two-state traffic light controller. We simplified the logic functions on the state transitions. For example, the transition from NSgreen to Ewgreen in the next-state table is (NScar EWcar) (NScar EWcar), which is equivalent to EWcar.