Digital Design and Computer Architecture

Danny Camenisch (dcamenisch)

March 15, 2026

Contents

Chapter 1

Introduction

1.1 Description

The class provides a first introduction to the design of digital circuits and computer architecture. It covers technical foundations of how a computing platform is designed from the bottom up. It introduces various execution paradigms, hardware description languages, and principles in digital design and computer architecture. The focus is on fundamental techniques employed in the design of modern microprocessors and their hardware/software interface.

Overview

This class provides a first approach to Computer Architecture. The students learn the design of digital circuits in order to:

- understand the basics
- understand the principles of design
- understand the precedents in computer architecture

Based on such understanding, the students are expected to:

- learn how a modern computer works underneath, from the bottom up
- evaluate tradeoffs of different designs and ideas
- implement a principled design (a simple microprocessor)
- learn to systematically debug increasingly complex systems
- hopefully be prepared to develop novel, out-of-the-box designs

The focus is on basics, principles, precedents, and how to use them to create/implement good designs.

Chapter 2

Basics

2.1 Definitions

In this script we will use the terms '1', TRUE and HIGH synonymously. Similarly, we will use '0', FALSE and LOW. Further we will use two's complement for signed binary numbers.

2.2 Design and Choices

Microprocessors where and still are a fundamental building block for the technological progess in the past three decades. Without them things like the Internet and cell phones would not exist. But for the development of new chips performance is not always the main priority. While performance is important, there is always a tradeoff between many different other aspects, including power consumption, temperature price and size, only to mention some of them.

Systems like a microprocessor can get really complicated to understand on a low level. To get a better understanding of such a system, it is necessary to use different techniques:

- Abstraction: hiding details when they are not important
- Disciplin: intentionally restricting design choices so that you can work more productively at a higher level of abstraction
- Hierarchy, Modularity, Regularity: defining a clear structure dividing a system into well-defined modules and reusing them among multiple points

Microarchitecture links the logic and architecture levels of abstraction. This will be the main focuse of this course.

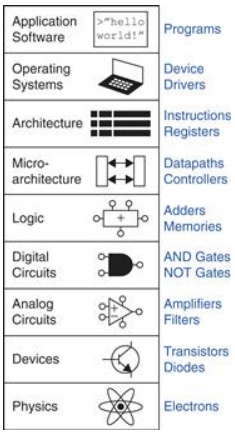


Figure 2.1: Levels of abstraction for a electronic computing system

2.3 Representing Numbers

There are three commonly used systems to represent numbers. We normally use decimal numbers, but in digital systems binary and hexadecimal numbers are more convienient. It is assumed that the reader is familiary with using these number systems and knows how to performe simple operations in them. When dealing with addition one has to be carefull to check for overflow of the carry bit.

Important Terms

- **Bit**: A single 0 or 1
- **Byte**: A group of eight bits, can be represented by two hexadecimal digit
- **Nibble**: Half a byte, can be represented by one hexadecimal digit
- **Word**: Data chunks a microprocessor handels, today 64 or 32 bit
- **LSB**: least significant bit, right-most bit in a group of bits
- **MSB**: most significant bit, left-most bit in a group of bits

There are multiple ways to represent a negative number in the binary system. The two most common ones are called sign/magnitude and two's complement.

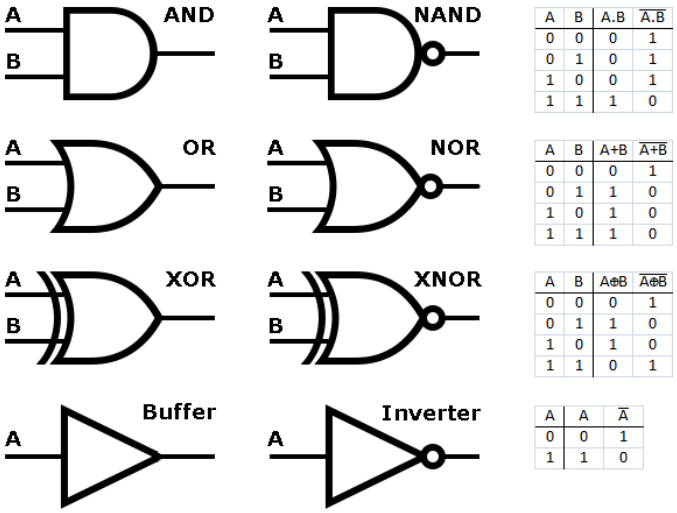
Sign/Magnitude: We use the MSB as a sign bit, this causes the problems that addition will no longer work and there exists both $+0$ and -0. This covers the range $[-2^{N-1}+1, 2^{N-1}-1]$.

Two's Complement: Same system as a unsigned binary number except the MSB has a weight of -2^{N-1}. Here addition works fine and there is only one representation for 0. The process of creating a negative number consists of inverting all the bits of the number and adding 1 to the LSB. This covers the range $[-2^{N-1}, 2^{N-1}-1]$.

2.4 Logic Gates

Logic Gates are simple digital circuits that take one or more binary inputs (denoted by letters from the beginning of the alphabet) and produce a binary output (commonly denoted Y). The most common logic gates are:

Note that a N-input XOR produces a TRUE output when an odd number of inputs are TRUE. From a logical point of view, a buffer might seem useless. However, from



A	B	A.B	$\overline{A.B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

A	B	A+B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

A	B	A⊕B	$\overline{A \oplus B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

A	A	$\overline{A}$
0	0	1
1	1	0

Figure 2.2: Most important logic gates

the analog point of view, the buffer has multiple, desirable characteristics such as the ability to quickly send its output to many gates. The logic operations can be denoted as follows:

BUFFER	$Y = A$
NOT	$Y = \overline{A}$
AND	$Y = A \cdot B = AB = A \cap B$
OR	$Y = A + B = A \cup B$
XOR	$Y = A \oplus B$
NAND	$Y = \overline{AB}$
NOR	$Y = \overline{A + B}$
XNOR	$Y = \overline{A \oplus B}$

There also exist logic gates with more than two inputs.

2.5 Beneath the Abstraction

Supply Voltage: In reality, binary signals are represented with a voltage on a wire. The lowest voltage in a system (0 V) is called the ground GND and the highest is denotes as V_{DD} (between 1.2 and 3.3 V).

Noise: Since in reality this signal is analog and could theoretically have every value between GND and V_{DD} it could be that we encounter noise. We can use a trick to filter such noise and get a clear signal. The noise margin is the amount of noise we can add to a signal and it can still be correctly interpreted. In this example we look at two simple logic gates, a driver and receiver. The noise margin is calculated $NM_L = V_{IL} - V_{OL}$, $NM_H = V_{OH} - V_{IH}$.

CMOS Transistors: We skip the detailed explanation of how a transistor works and look at what it does. A transistor can be viewed as a electrically controlled switch that turns ON or OFF when a voltage is applied to a controll terminal. The most common transistors are MOS transistors (or MOSFET). There are two types nMOS and pMOS.

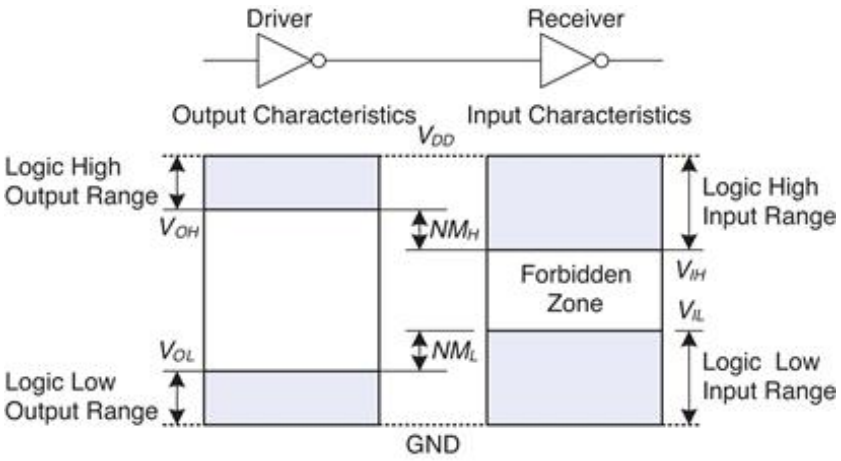


Figure 2.3: Noise characteristics

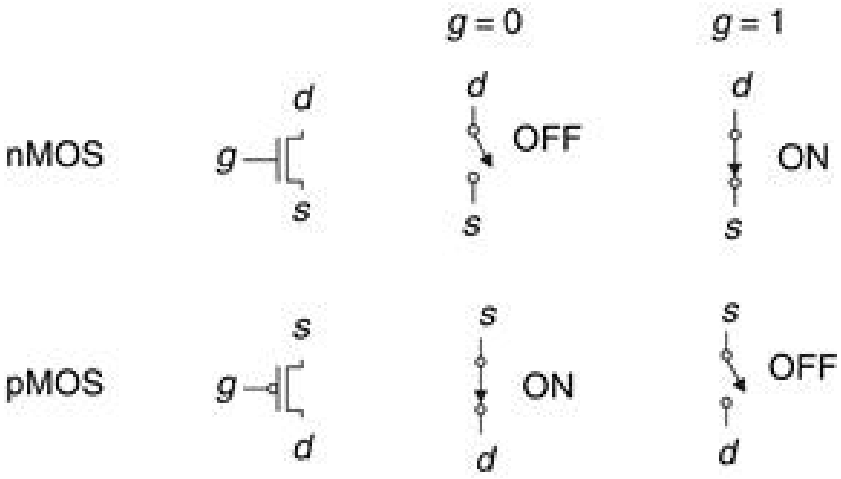


Figure 2.4: MOS transistors

Logic Gates: We can use transistor to build logic gates, as an example we will have a look at how a AND gate is built. A AND gate consists of a NAND gate and a NOT gate, that are made up by transistors as can be seen in the figure.

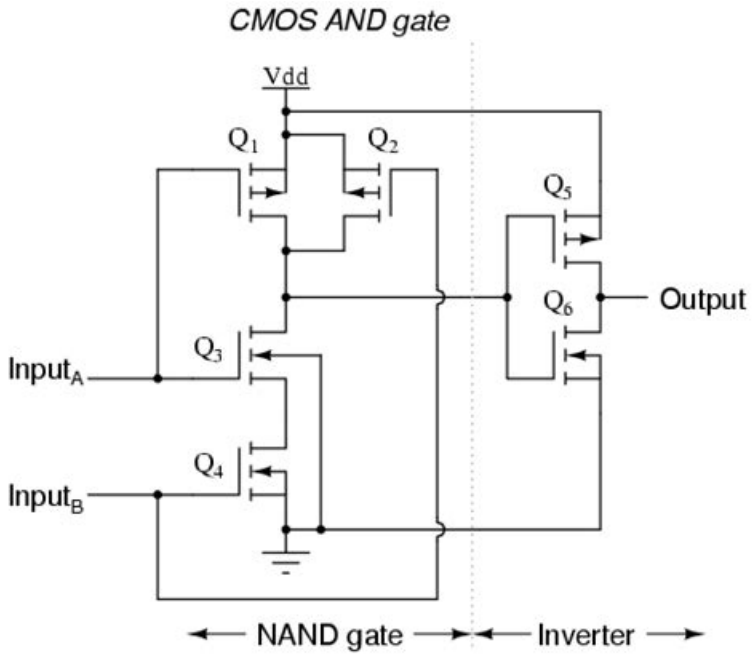


Figure 2.5: MOS transistors

Chapter 3

Combination Logic Design

3.1 Introduction

A **circuit** is a network that processes distinct-value variables. It has:

- one or more input terminals
- one or more output terminals
- a functional specification (expressed by a truth table)
- a timing specification

Digital circuits are classified as **sequential** or **combinational** (this chapter will discuss the later one). The difference between these two is, that a combinational circuit combines the current input values to compute the output, whereas in a sequential circuit, the output depends on the current input and on the previous input values. In another way, a combinational circuit is **memoryless** and a sequential circuit has **memory**.

Combination circuit therefore have to satisfie the following requirements:

- every circuit (element) is itself combinational
- every wire (node) of the circuit is either designated as input or output
- the circuit contains no cyclic paths

3.2 Boolean Algebra

Definition

- **Complement** - The inverse of a variable, $\bar{A}$
- **Literal** - A variable in a equation
- **Product** - The AND of two or more literals
- **Minterm** - The product of all inputs (or their complement) of a function
- **Sum** - The OR of two or more literals
- **Maxterm** - The sum of all inverse of inputs (or their complement) of a function
- **Precedence** - The order of operations: NOT ¿ AND ¿ OR
- **Sum of product form (SOP)** - The sum of all minterms for which a function is true
- **Product of sum form (POS)** - The product of all maxterms for which a function is false
- **Prime implication** - An implicant that cannot be combined with other implicants to for a new implicant with fewer literals

3.2.1 Axioms and Theorems

	Axiom		Dual	Name
A1	$B=0$ if $B\neq 1$	A1′	$B=1$ if $B\neq 0$	Binary field
A2	$\overline{0}=1$	A2′	$\overline{1}=0$	NOT
A3	$0 \bullet 0=0$	A3′	$1+1=1$	AND/OR
A4	$1 \bullet 1=1$	A4′	$0+0=0$	AND/OR
A5	$0 \bullet 1=1 \bullet 0=0$	A5′	$1+0=0+1=1$	AND/OR

Figure 3.1: Axioms

	Theorem		Dual	Name
T1	$B \bullet 1=B$	T1′	$B+0=B$	Identity
T2	$B \bullet 0=0$	T2′	$B+1=1$	Null Element
T3	$B \bullet B=B$	T3′	$B+B=B$	Idempotency
T4		$\overline{\overline{B}}=B$		Involution
T5	$B \bullet \overline{B}=0$	T5′	$B+\overline{B}=1$	Complements

Figure 3.2: Theorems

	Theorem		Dual	Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + (B \bullet D) = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots}$ $= (\overline{B}_0 + \overline{B}_1 + \overline{B}_2 \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots}$ $= (\overline{B}_0 \bullet \overline{B}_1 \bullet \overline{B}_2 \ldots)$	De Morgan's Theorem

Figure 3.3: Theorems for multiple variables

We can use these helpful theorems to reduce boolean equations into prime implicants.

3.3 From Logic to Gates

When drawing a **schemantic** we generally follow these rules:

- Inputs are on the left / top side
- Ouputs are on the right / bottom side
- Gates should flow from left to right
- Straight wires are better
- Wires always connect at a T Junction
- Wires crossing without a dot make no connection

To draw such a schemantic it can be usefull to first draw a truth table. In a truth table a X marks a don't care.

3.4 Multilevel Combinational Logic

In this section we will shortly mention bubble pushing as a helpful way to redraw combinational circuits so that bubbles cancle out and the function can be more easily determined. (More details can be found in the book)

3.5 Illegal and Floating Values

The symbol X indicates that the circuit node has an **unknown** or **illegal** value, this commonly happens if a node is being driven both by a 0 and a 1 at the same time. This situation is also called **contention**. Be carefull to not mistake this X with a don't care in truth tables.

The symbol Z denotes that a node is being driven neither by HIGH nor LOW. The node is said to be **floating, heigh impedance** or **high Z**. In reality, a floating node might be 0 or 1 or something in between.

3.6 Karnaugh Maps

Karnaugh Maps are a usefull graphical method for simplifying Boolean equations. They work well for equations with up to four variables.

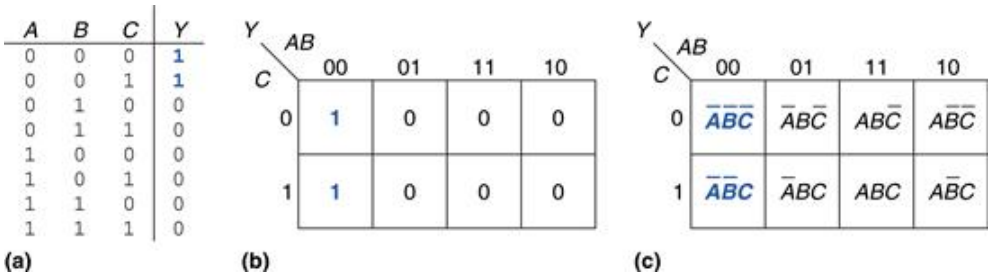


Figure 3.4: A Karnaugh Maps

Note that the AB combinations are in Gray code.

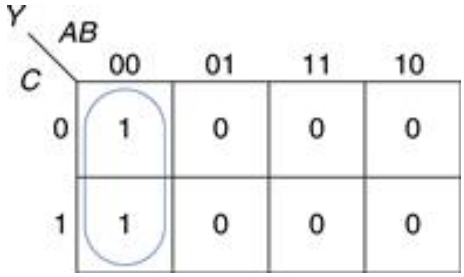


Figure 3.5: A Karnaugh Maps

We use the following rules to minimize a expression using a K-map:

- Us the fewest circles necessary to cover all 1's
- A circle can only contain 1's
- Each circle must span a rectangular block that is a power of 2 in each direction
- Each circle should be as large as possible
- A circle may wrap around the edge of the K-map
- A 1 in a K-map may be circled multiple times

3.7 Combinational Building Blocks

Combinational Logic is often grouped into larger building blocks to build more complex systems. This is an application fo the principle of abstraction. Some of these building blocks are full adders, priority circuits and seven-segment display decoders. In this section we will look at another two building blocks.

3.7.1 Multiplexer (mux)

Multiplexers (short mux) are among the most commonly used combinational circuits. They choose an output among several possible inputs based on the value of a select signal. A 2:1 mux chooses between 2 different input signals.

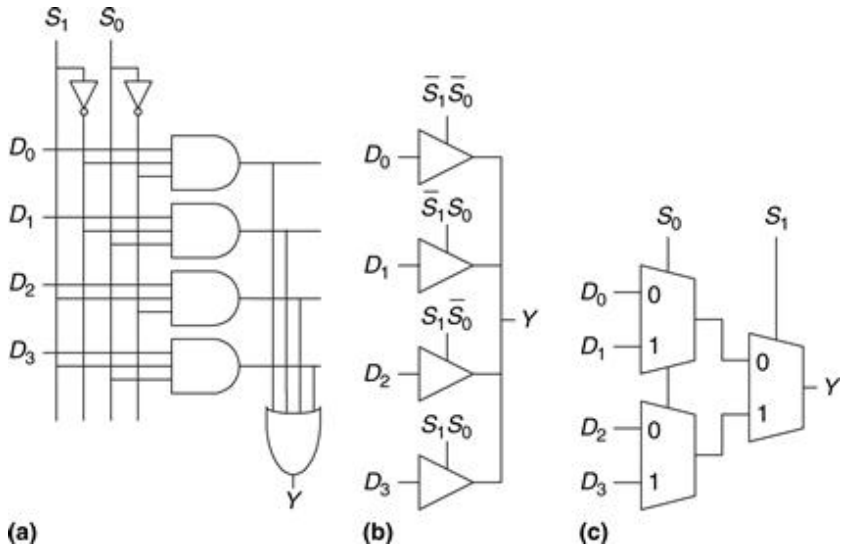


Figure 3.6: 4:1 mux implementations

3.7.2 Decoders

A decoder has n inputs and 2^n outputs. It asserts exactly one of its outputs depending on the input combination. The output of a decoder are called **one-hot**, because exactly one of them is hot at a given time.

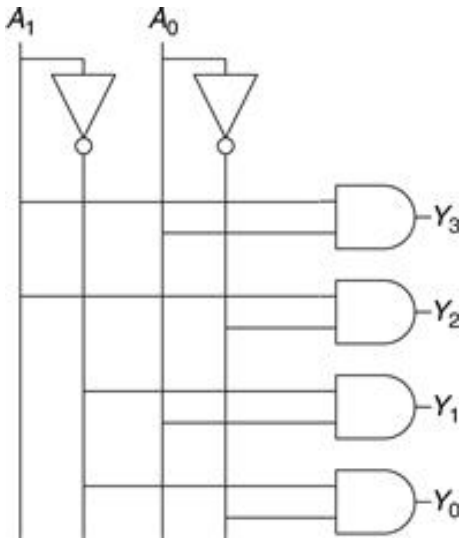


Figure 3.7: 2:4 decoder

3.8 Timing

An output takes time to change in response to a change of input. We can draw a **timing diagram** to visialize the transient response of a combinational circuit. The transition from LOW to HIGH is called the **rising edge** and from HIGH to LOW it's called the **falling edge**.

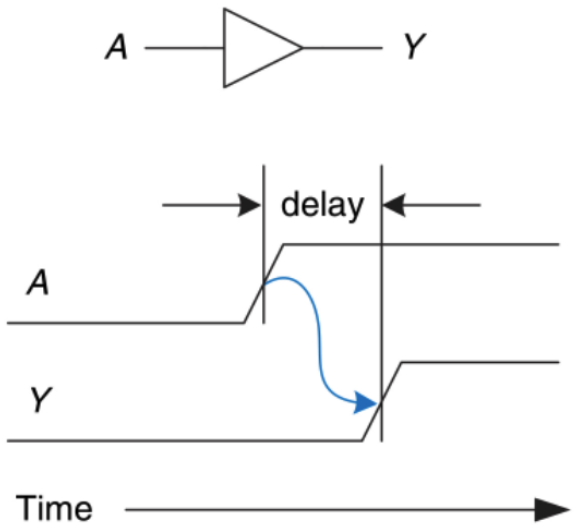


Figure 3.8: timing diagram of a buffer

Combinational logic is characterized by its **propagation delay** t_{pd}, the maximum time from when a input changes until the output reaches their final state, and the **contamination delay** t_{cd}, the minimum time from an input change until the output starts changing.

Since t_{pd} and t_{cd} are determined by signal paths in a combinational circuit, we introduce the following two definitions:

- **Critical path** - The longest and therefore slowest path in a circuit
- **Shortest path** - The shortest and therefore fastest path through a circuit

We can now calculate the t_{pd} by adding up the propagation time for each element along the critical path, and similarly calculate the t_{cp} by adding all propagation times along the short path.

Chapter 4

Sequential Logic

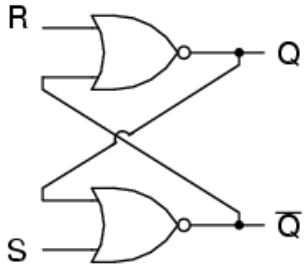
4.1 Introduction

We will now analyze **sequential logic**. In addition to being dependend on the current input, sequential logic is also dependend on the prior input (past state). We therefore say, sequential logic has **memory**.

4.2 Latches and Flip-Flops

4.2.1 SR Latch

One of the simplest sequential circuits is the SR Latch, it can be built from either NOR or NAND gates. The state of the SR Latch can be controlled through the S (set) and the R (reset) input.



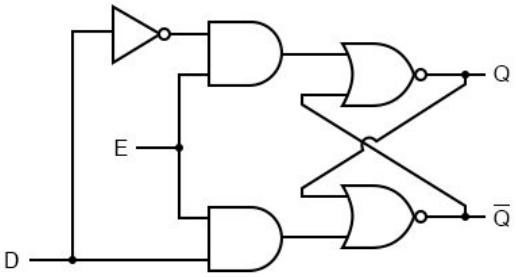
S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

Figure 4.1: SR Latch

The SR Latch has the simple but powerful functionality, that if both inputs R and S are 0, Q will keep the previous values. The circuit has **memory**.

4.2.2 D Latch

The limitation of the SR Latch is, that by giving an input on S and R not do we change the state, we also decide when the state should change. We want to seperate these two actions. In addition we also want to remove the state where both set and reset are active. The D Latch does exactly that.



E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

Figure 4.2: D Latch

A D Latch has two inputs, one data input D and a clock input E or CLK. Observe that when $E = 0$ the output will never change, only when $E = 1$ a new value is accepted from D. When $E = 1$, we say that the latch is **transparent** and **opaque** otherwise.

4.2.3 D Flip-Flop

We notice that whenever E is set to 1, a new input is accepted. This is not always desirable, we often want a D Latch to only accept a input at the rising edge of the clock. We can combine two D Latches to form a D Flip-Flop and achieve this desired behaviour.

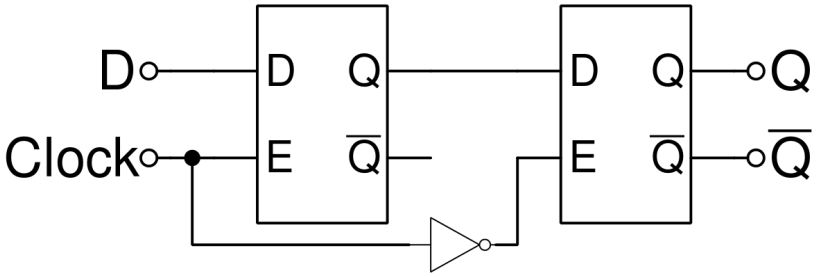


Figure 4.3: D Flip-Flop

The first D Latche is calles the master and the second one is the slave. The functionality of the D Flip-Flop is essential for many digital designs and should be known by heart.

D Flip-Flop

A D Flip-Flop copies D to Q on the rising edge of the clock, and remembers its state at all other times.

4.2.4 Register

An N-bit register is a bank of N flip-flops that share a common clock signal. Registers are used to save multiple bits of data and are a key building block of most sequential circuits.

4.2.5 Enabled Flip-Flop

An enabled flip-flop adds another input called EN to determine whether data is loaded on the clock edge or not. When EN is true, it behaves like a normal D Flip-Flop otherwise it just retains its current state.

4.2.6 Resettable Flip-Flop

A resettable flip-flop also adds another input called $RESET$. When $RESET$ is active, the resettable flip-flop ignores D and sets the output to 0. We make a difference between synchronously and asynchronously resettable flip-flops, the first only resets on the rising edge of the clock. Resettable flip-flops are usefull, since at system startup we don't know in which state a flip-flop will be.

4.3 Synchronous Logic Design

In general, sequential circuits include all circuits that are not combinational, that is, those whos outputs cannot be determined simply by looking at the current inputs.

A sequential circuit has a finite set of discrete states. A synchronous sequential circuit has a clock input, whose rising edges indicate a sequence of times at which state transitions occure. The rules of synchronous sequential circuit compositions are:

- Every circuit element is either a register or a combinational circuit
- At least one circuit element is a register
- All registers receive the same clock signal
- Every cyclic path contains at least one register

Similarly, if a circuit is not synchronous, it is said to be asynchronous.

4.4 Finite State Machines

Finite state machines are synchronous sequential circuit with k registers and 2^k possible states. An FSM consists of two blocks of combinational logic, next state logic and output logic, and a register that stores the state.

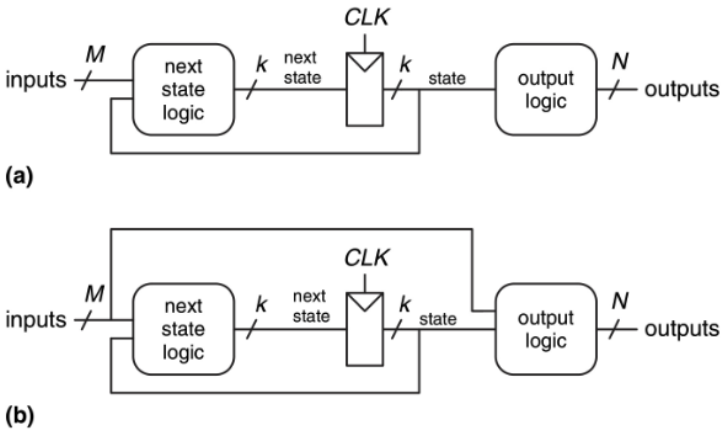


Figure 4.4: a) Moore Machine, b) Mealy Machine

There are two general classes of FSM, Moore Machines, where the output only depends on the current state, and Mealy Machines, where the output depends on both the current state and the current inputs.

4.4.1 Designing an FSM

The design process for an FSM involves several different steps. Those steps are usually the same for each FSM. Here we will look at this process for a Moore Machine:

- Identify the inputs and outputs
- Sketch a state transition diagram
- Write a state transistion table
- Write a outpute table
- Select state encodings
- Write boolean equations for the next state and output logic
- Sketch the circuit schemantics

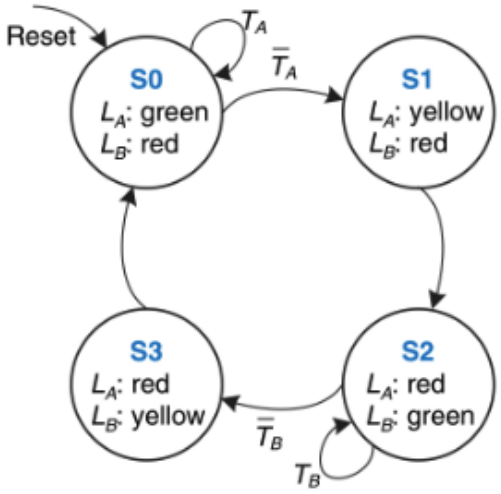


Figure 4.5: State transition diagram of a stop light

4.5 Timing of Sequential Logic

When a clock rises, the output may start to change after the contamination delay and must definitely settle to the final value within the propagation delay. For the circuit to sample its input correctly, the input must be stable at least some time before the rising edge, setup time, and remain stable for at least some time after the rising edge, hold time. The sum of setup and hold time is called the aperture time of a circuit, during this time, the input must remain stable. The dynamic discipline states that inputs of a synchronous circuit must be stable during the aperture time.

4.5.1 Metastability

As noted earlier, it is not always possible to guarantee that the input to a sequential circuit is stable during the aperture time. When a flip-flop samples an input that is changing during its aperture, the output Q may momentarily take on a voltage between 0 and V_{DD} that is in the forbidden zone. This is called a metastable state.

4.6 Parallelism

The speed of a system is characterized by the latency and throughput of information moving thought it. We define a **token** to be a group of inputs that are processed to produce a group of outputs. The **latency** of a system is the time required for one token to pass through the system from start to end. The **throughput** is the number of tokens that can be produced per unit time.

The throughput can be improved by processing several tokens at the same time. This is called parallelism, and it comes in two forms: spatial and temporal. With spatial parallelism, multiple copies of the hardware are provided so that multiple tasks can be done at the same time. With temporal parallelism, a task is broken into stages, like an assembly line. Those taks can be spread across stages, and if spread correctly, a different task will be in each stage at any given time so multiple tasks can overlap. Temporal parallelism is commonly refered to as pipelining.