「计算机系统概论」TA Session 2

2022-2023 短学期计算系统概论课程

By @HobbitQia

More about Session 1

Logical Operations

- 逻辑运算: AND, OR, NOT, XOR

$$X \cdot Y \Leftrightarrow X\ AND\ Y$$
$$X + Y \Leftrightarrow X\ OR\ Y$$
$$X \oplus Y \Leftrightarrow X\ XOR\ Y$$
$$\overline{X} \Leftrightarrow NOT\ X$$

- DeMorgan's Law
- $a\ AND\ 1 = a, a\ OR\ 0 = a$

Compiler and Assembler

- Compiler 编译器把高级语言直接转化为 ISA
- Assembler 汇编器把低级语言（汇编语言）转化为 ISA

也有人认为 Compiler 先把高级语言转为低级语言，再通过 Assembler 转化为 ISA。

Overflow

Overflow 的原因是因为计算机的位数有限，可能会出现计算结果超出了计算机位数能表示范围，就会得到意想不到的结果。
出现溢出只有这几种情况

- 非负数 + 非负数 = 负数
- 负数 + 负数 = 非负数
- 非负数 - 负数 = 负数（等价于情况 1）
- 负数 - 非负数 = 非负数（等价于情况 2）

在计算 n 位加法的时候，可能会出现 $n+1$ 位的进位，这并不代表溢出。

e.g. $n=4$

$1000(-8)+1000(-8)=10000(0)$ （进位 1 去掉）

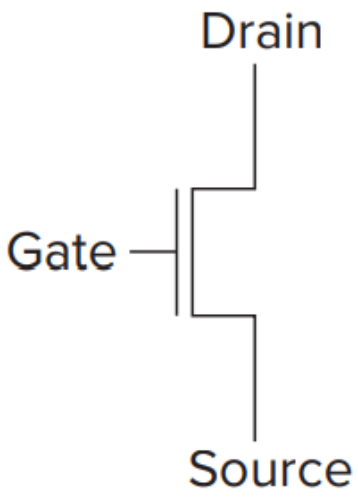
$0111(7)+0010(2)=1001(-7)$

Chapter 3 Digital Logic Structures

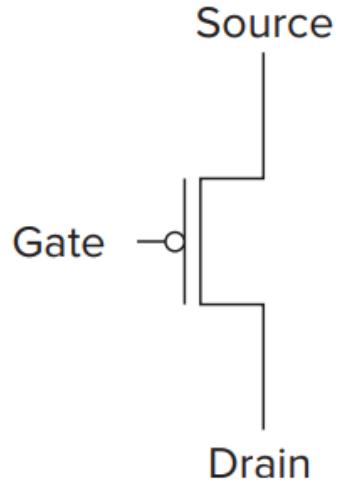
Transistor

MOS Transistors are *below the lowest level of abstraction*, so we don't need to care about how they are constructed and how they work, we just need to use it to build logic gates.

NMOS (N-type MOS)
高电压接通，低电压断开



PMOS (P-type MOS)
高电压断开，低电压接通



其中 Gate 叫栅极，Drain 叫漏极，Source 叫源级。

Logic Gates

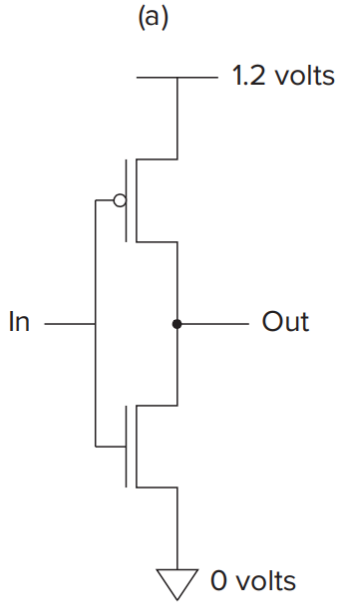
我们尝试用 NMOS 和 PMOS 晶体管搭建逻辑电路。
（我们用 1.2V 代表 "1", 0.0V 代表 "0"）

- NOT 非门 The **NOT** gate is also called **inverter** and its use to *negate the input*. ***e.g.*** If the input is 1.2V then the output is 0V and vice versa.

Logic Gates

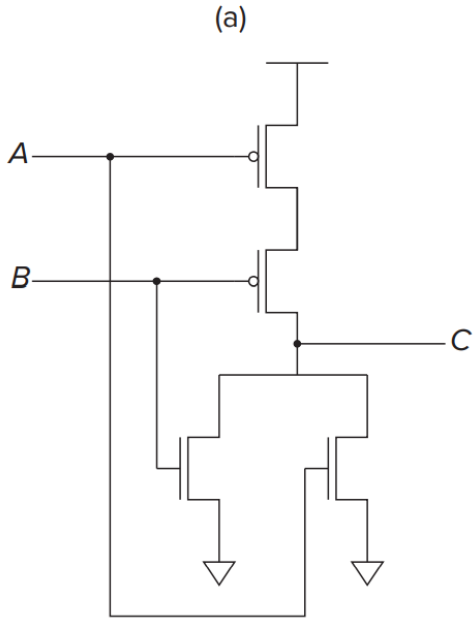
我们尝试用 NMOS 和 PMOS 晶体管搭建逻辑电路。
（我们用 1.2V 代表 "1", 0.0V 代表 "0"）

- NOT 非门 The **NOT** gate is also called **inverter** and its use to *negate the input*. ***e.g.*** If the input is 1.2V then the output is 0V and vice versa.



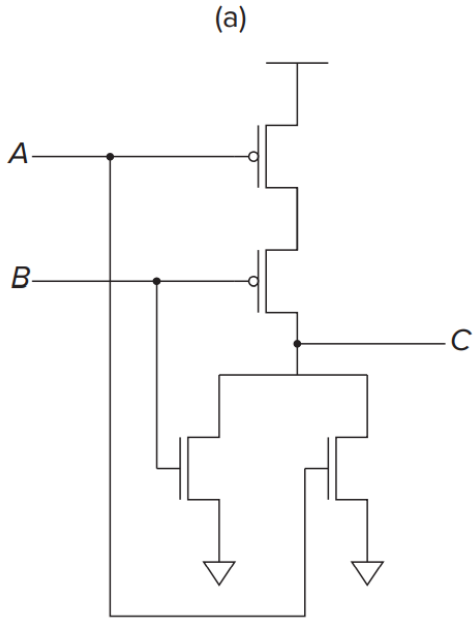
Logic Gates (Cont.)

- NOR
  NOR 门的功能和逻辑函数 NOR
  相同

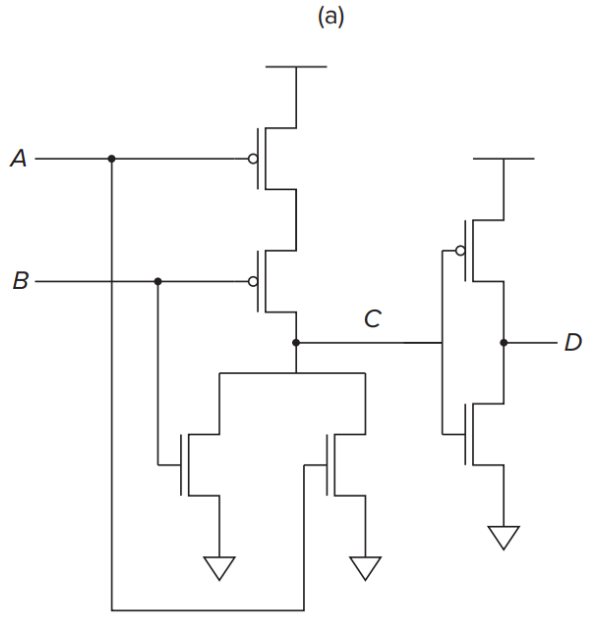


Logic Gates (Cont.)

- NOR
  NOR 门的功能和逻辑函数 NOR 相同



- OR
  我们可以在 NOR 门的输出后面加上一个 NOT 门即可得到 OR.



Logic Gates (Cont.)

- C 下面的 MOS 管不能省略，需要接地。(接地不等于空载)
- MOS 管不能倒过来使用（即 Drain 和 Source 的位置不能变，对于 NMOS 就是从下流到上，对于 PMOS 就是从上流到下），否则会有传输电压的损失。

Logic Gates (Cont.)

- NAND & AND NAND 门和 AND 门的构造方法和 NOR & OR 部分类似

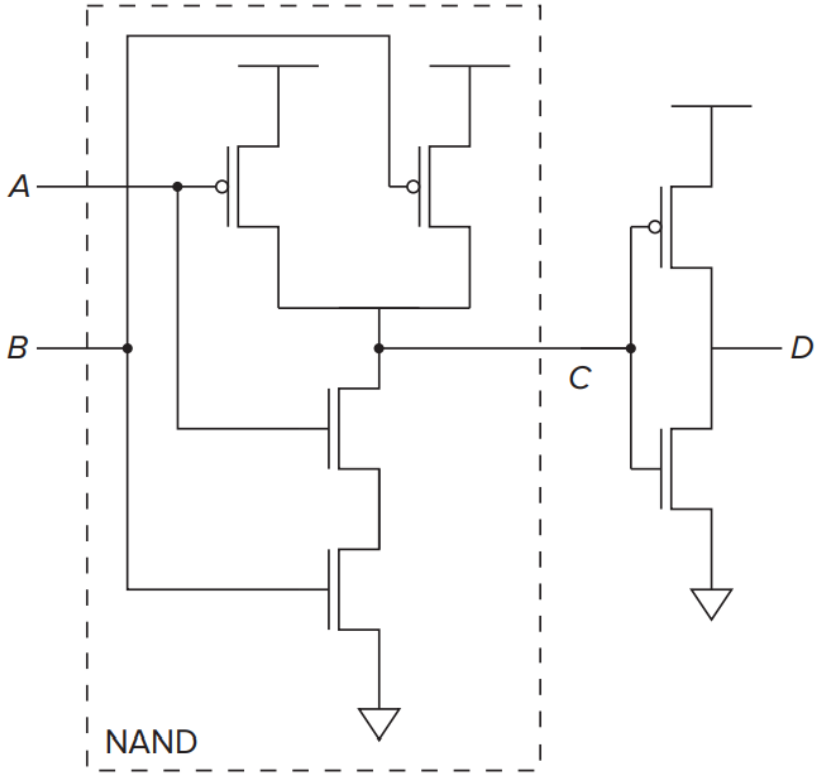


Figure 3.8 The AND gate.

Summary of Logic Gates

到目前为止，我们了解了基本的逻辑门的搭建，这也意味着我们提升了我们的抽象层次（从晶体管到逻辑门），后面的内容中我们将使用以下符号表示逻辑门：

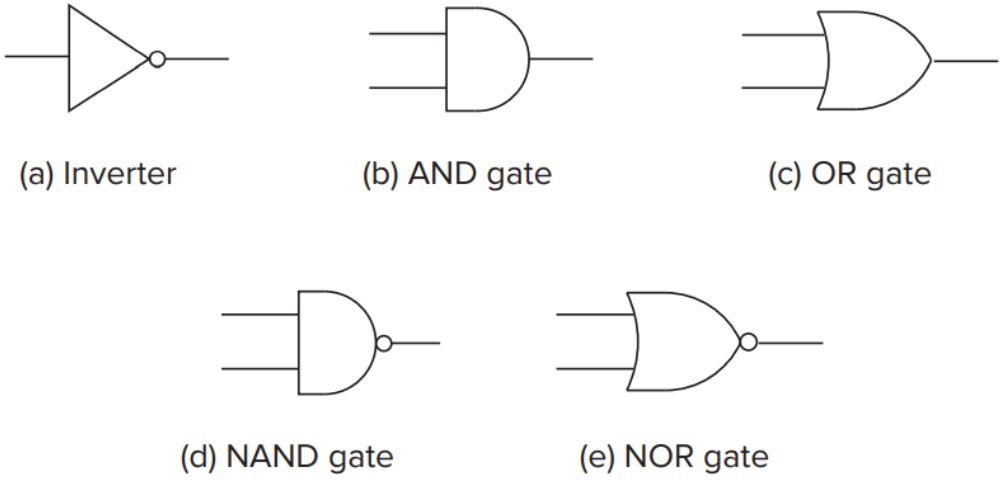


Figure 3.9 Basic logic gates.

这里空心圆圈表示“取反”，我们也可以在逻辑门输入端口加上空心圆圈表示对输入求反。

Combinational Logic Circuits

我们把电路分成两种，**combinational** 组合逻辑电路 and **sequential** 时序逻辑电路。区别在于电路本身是否能存储信息。

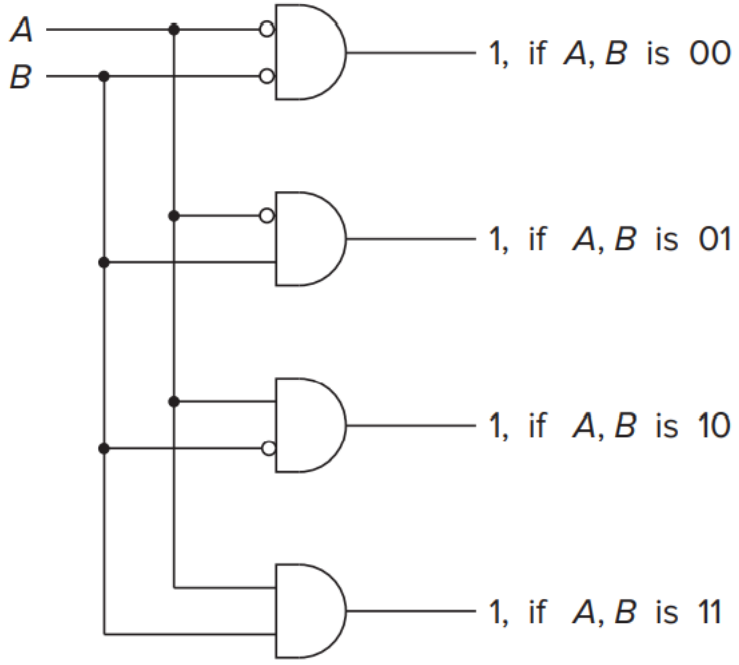
组合逻辑电路不能存储信息，电路输入完全依赖于当前的输入。

3 common combinational logic circuits

- Decoder 译码器
- Mux 多路选择器
- Adder 加法器

Decoder

A **decoder** has n inputs and 2^n outputs and only one of its outputs is 1 and all the rest are 0s.



Multiplexer

Multiplexer（或 Mux）多路选择器，从多个输入中根据选择信号选择一个输入并将其输出。

e.g. 2-to-1 Mux

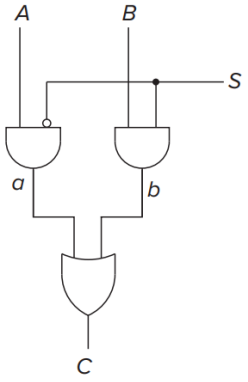
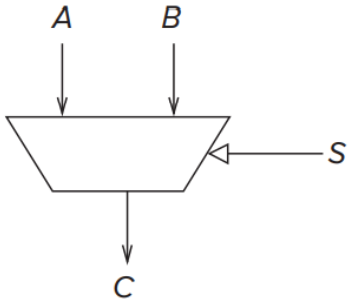


Figure 3.12 A 2-to-1 mux.

2-to-1 Mux 可以记为



- S=0 时，输出 C 和输入 A 相同
- S=1 时，输出 C 和输入 B 相同

Multiplexer (Cont.)

一般来说，一个 Mux 会有至多 2^n 个输入，同时有 n 位的选择信号。

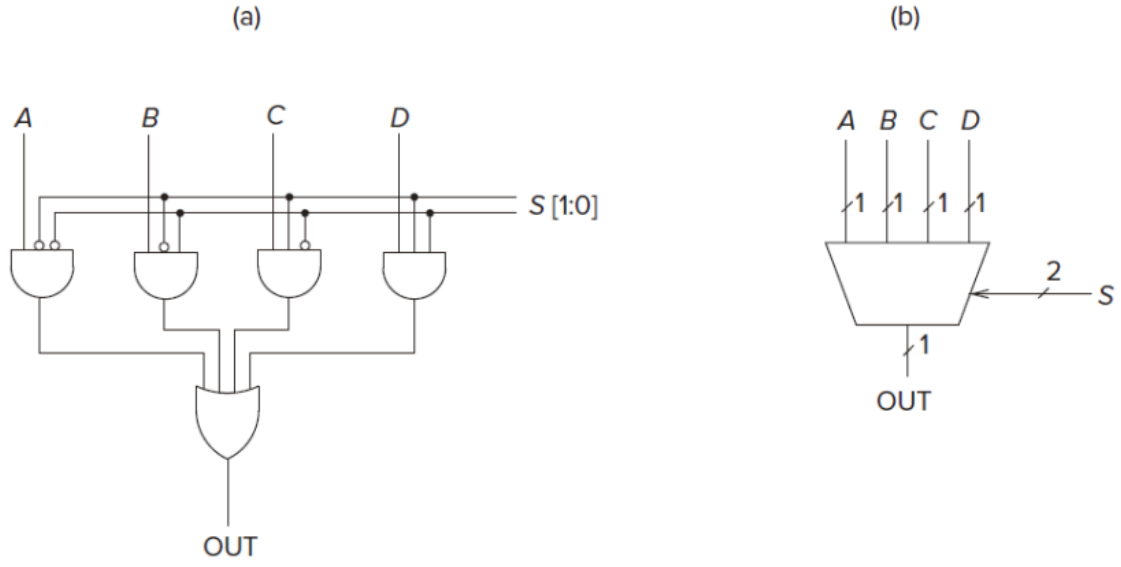


Figure 3.13 A four-input mux.

S[1:0]=00 时选择 A，S[1:0]=01 时选择 B，S[1:0]=10 时选择 C，S[1:0]=11 时选择 D。

Adder

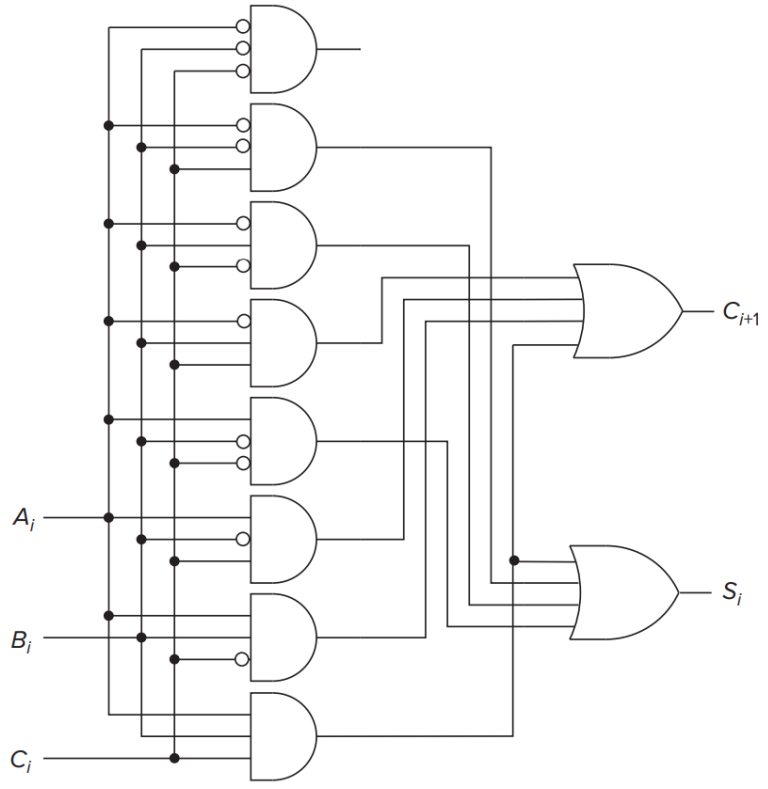
假设有两个 n 位的二进制数 A, B，要计算 $A + B = S$。
我们用 C_i 表示第 i 位的进位，S_i 表示计算加法之后 C_i 的值。

我们可以得到一位加法器的真值表（右图）
A_i, B_i, C_i 为输入
C_{i+1}, S_i 为输出

A_i	B_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

1-bit Adder

根据刚刚的真值表，我们可以得到这样的一位加法器：



这样的 1-bit adder 也称为 **full adder** 全加器

Adder (Cont.)

4 位的加法器可以由下面的结构表示：

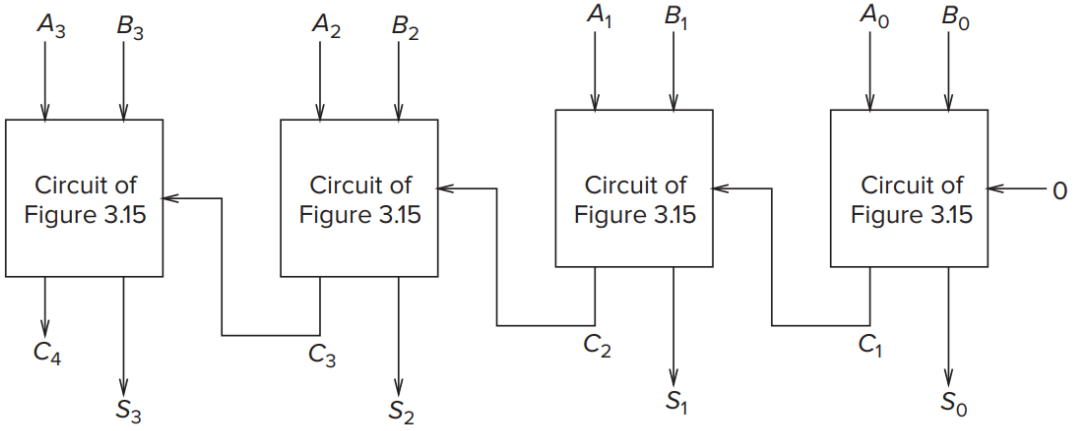


Figure 3.16 A circuit for adding two 4-bit binary numbers.

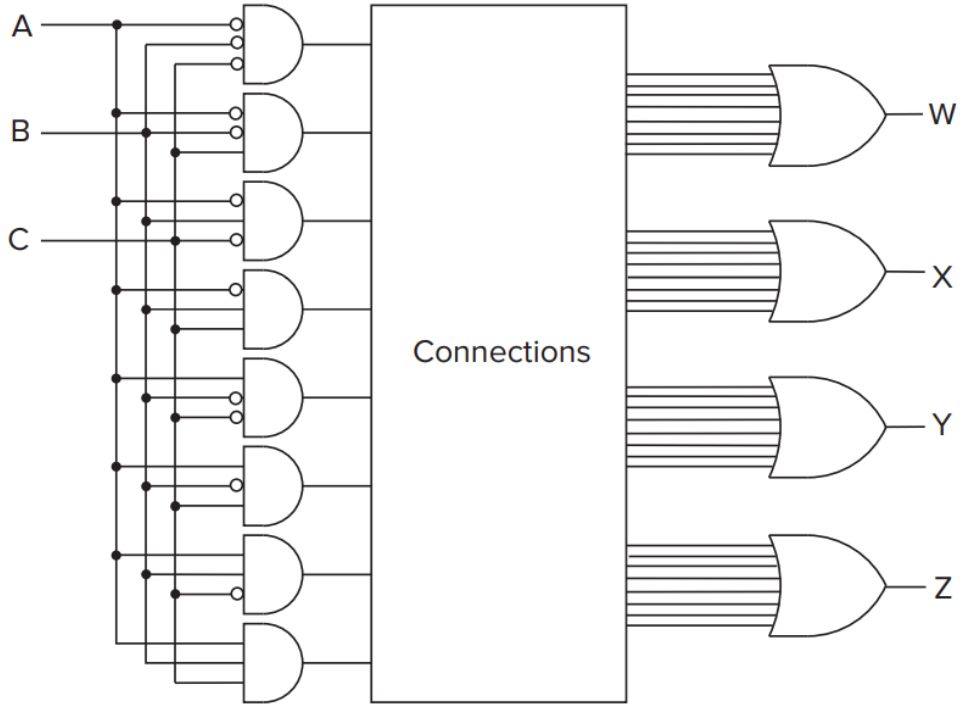
这里最右边的 1-bit adder 称为 **half adder** 半加器，其他的 1-bit adder 都是全加器。

区别在于 carry bit，半加器的 C_i 默认为 0（即不考虑低位进位到第 i 位的情况），全加器的 C_i 需要根据前一位的运算结果决定。

PLA

PLA (Programmable logic array) consists of an array of AND gates followed by an array of OR gates.

PLA 的形式如下:



A programmable logic array.

PLA (Cont.)

假设我们有 n 个输入，有 m 个输出。那么对于每个输出，我们可以找到对应的能使输出为 1 的输入组合（真值表）。
PLA 实现的方式就是先将 n 个输入通过 Decoder 得到输入组合，随后根据真值表连接输出和对应的输入组合。

e.g. 在刚刚的图中，假设 $W = A \cdot B$。只有 $A = B = C = 1$ 或 $A = B = 1, C = 0$ 时 $W = 1$，因此我们把 "111" 和 "110" 这个输入组合连向输出 W 的或门。

PLA (Cont.)

假设我们有 n 个输入，有 m 个输出。那么对于每个输出，我们可以找到对应的能使输出为 1 的输入组合（真值表）。
PLA 实现的方式就是先将 n 个输入通过 Decoder 得到输入组合，随后根据真值表连接输出和对应的输入组合。

e.g. 在刚刚的图中，假设 $W = A \cdot B$。只有 $A = B = C = 1$ 或 $A = B = 1, C = 0$ 时 $W = 1$，因此我们把 "111" 和 "110" 这个输入组合连向输出 W 的或门。

PLA 可以实现任何逻辑函数。$\Rightarrow$ PLA 是 **logical complete**（逻辑完备的）
PLA 只由 AND OR NOT 组成。$\Rightarrow$ {AND, OR, NOT} 是逻辑完备的

PLA (Cont.)

假设我们有 n 个输入，有 m 个输出。那么对于每个输出，我们可以找到对应的能使输出为 1 的输入组合（真值表）。
PLA 实现的方式就是先将 n 个输入通过 Decoder 得到输入组合，随后根据真值表连接输出和对应的输入组合。

e.g. 在刚刚的图中，假设 $W = A \cdot B$。只有 $A = B = C = 1$ 或 $A = B = 1, C = 0$ 时 $W = 1$，因此我们把 "111" 和 "110" 这个输入组合连向输出 W 的或门。

PLA 可以实现任何逻辑函数。$\Rightarrow$ PLA 是 **logical complete**（逻辑完备的）
PLA 只由 AND OR NOT 组成。$\Rightarrow$ {AND, OR, NOT} 是逻辑完备的

Question: NAND 是逻辑完备的吗？NOR？为什么？

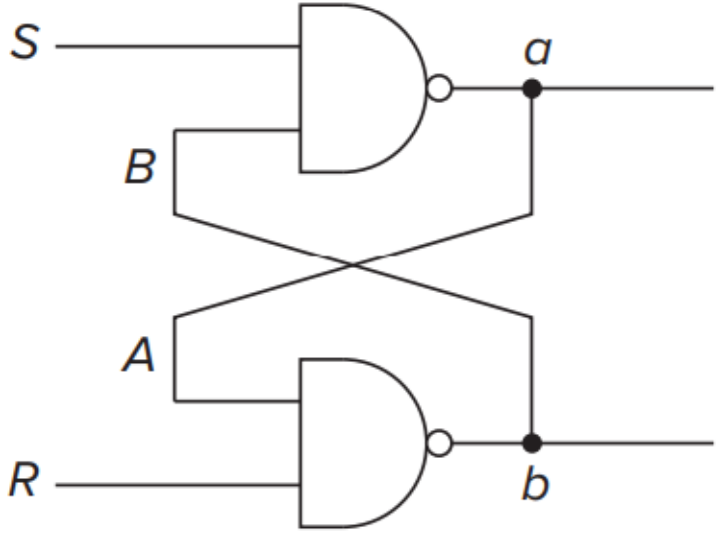
Sequential Logic Circuits

时序电路的特点是可以存储信息，因此我们需要一些特定的存储单元。
常用的存储单元如下：

- R-S Latch 锁存器
- Gated D Latch D 锁存器
- Master-Slave Flip-Flop 主从触发器（边缘触发器）

R-S Latch

R-S 锁存器用来存储 1 bit 信息。
它有两个输入 R 和 S，两个输出 a 和 b。
我们规定 a 就是我们存储在锁存器里的值，同时规定 $a = Q, b = \overline{Q}$。



R-S Latch (Cont.)

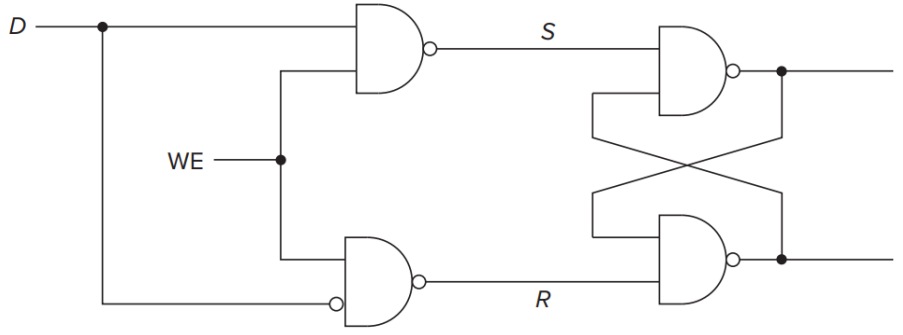
- 起初 $R = S = 1$，Unknown
- 如果我们想要 *set* the latch（存入 1）我们只需要把 S 从 1 拨到 0，这时 a 变为 1，b 变为 0，再把 S 从 0 拨回 1 即可保持刚刚存进去的 1。
- 如果我们想要 *reset* the latch（存入 0）我们只需要把 R 从 1 拨到 0，这时 a 变为 0，b 变为 1，再把 R 从 0 拨回 1 即可保持刚刚存进去的 0。

R-S Latch (Cont.)

S	R	$a(Q)$	$b(\overline{Q})$	
1	1	?	?	Unknown
0	1	1	0	Set
1	1	1	0	The latch "remenber" the value
1	0	0	1	Reset
1	1	0	1	The latch "remenber" the value
0	0	1	1	Both go to high
1	1	?	?	Unknown

Gated D Latch

在 R-S 锁存器的基础上，我们增加了一个 **WE(write enable)** 信号，控制我们能否写锁存器。



- $WE = 1$ 时，表示我们可以修改锁存器内存储的值
  - $D = 1$ 时 $S = 0, R = 1$ 则我们存入 1
  - $D = 0$ 时 $S = 1, R = 0$ 则我们存入 0
- $WE = 0$ 时，表示我们不能修改锁存器内存储的值，无论 D 是多少，均有 $S = R = 1$ 即锁存器保持之前存的值。

Memory

Memory 内存是由很多地址组成的，地址是唯一的，同时每个地址都可以存数据。

- **Address Space** 地址空间
  内存里不同地址的个数，称为地址空间。

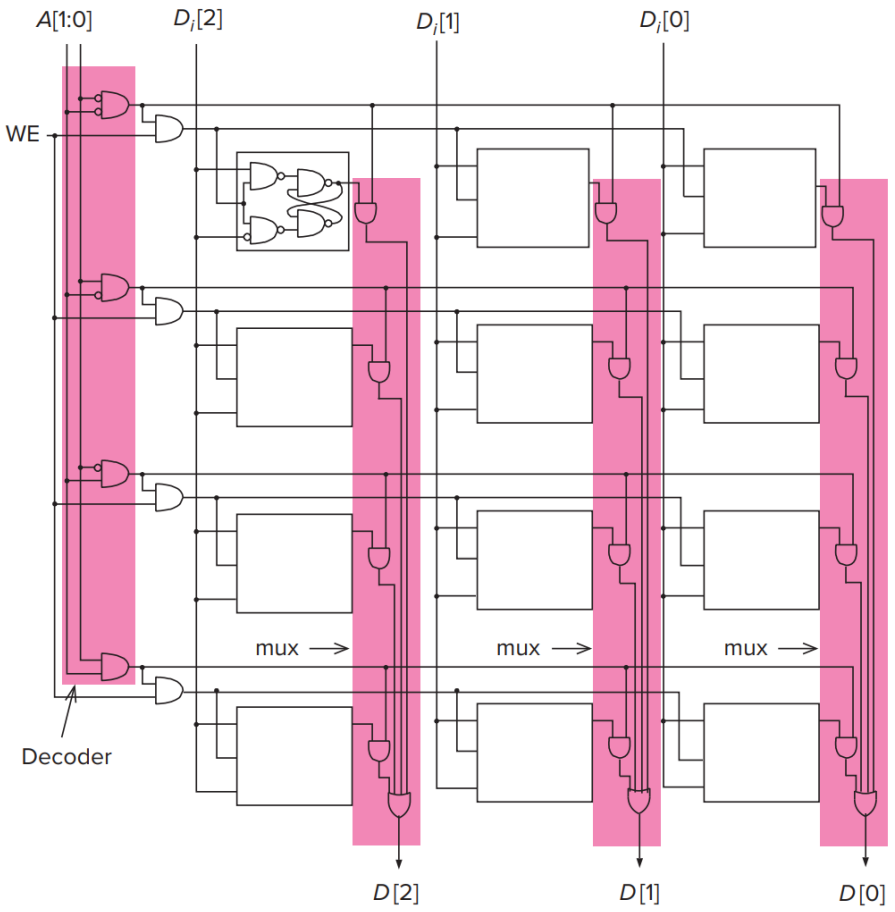
e.g. 我们用 32 位表示地址，则地址空间为 2^{32}

- **Addressability** 寻址能力
  每个地址可以存放的数据位数。

e.g. 一个地址可以放 1 个 bytes，则我们称寻址能力是 8 bits。

Memory (Cont.)

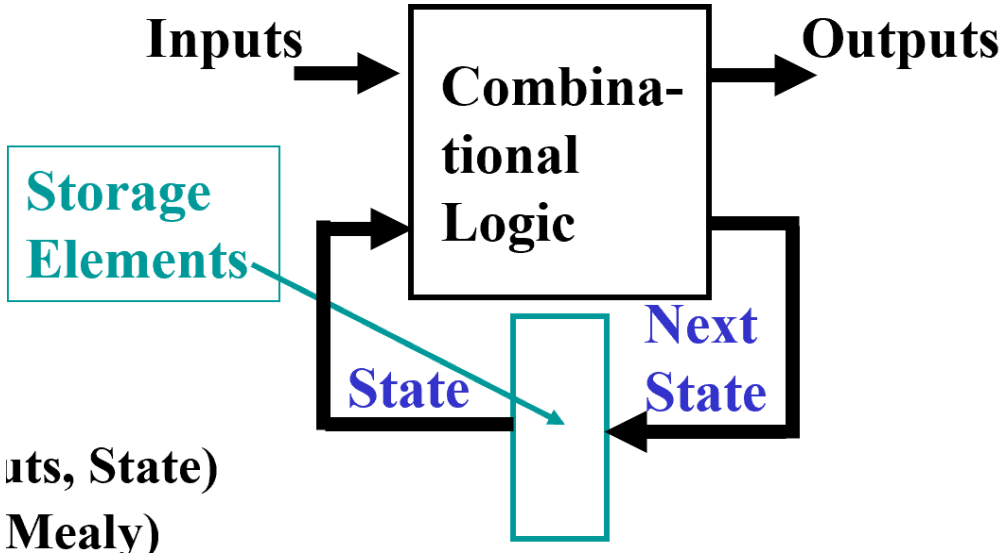
e.g. 2^2-by-3-bit Memory



- 地址是 $A[1:0]$，地址空间为 $2^2 = 4$
- 每个位置有三个存储单元，即可以存储 3 bits 的数据，因此寻址能力是 3 bits。
- $WE = 1$ 时将 $D_i[2:0]$ 写入内存（三个比特同时写）；$WE = 0$ 时从内存中读出数据到 $D[2:0]$

Sequential Logic Circuits Design

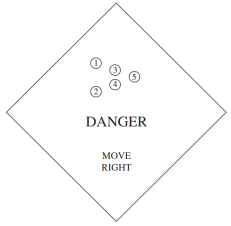
时序电路里，既有存储单元用来存储信息，又有组合逻辑进行运算。其中组合逻辑的作用是根据外部输入和存储单元的内容，得到输出，并更新存储单元的信息，这部分也被称为 Finite State Machine（有限状态机）。



Finite State Machine

- state 状态
  The **state** of a system is a snapshot of all the relevant elements of the system at the moment the snapshot is taken.
  其中存储单元的内容是状态的重要组成部分。
- 有限状态机由有限个状态、输入、输出、状态转化组成。为了表示有限状态机，我们会使用 state diagram 状态图。状态图包括：
  - 圆圈表示状态
  - 有向弧表示状态转化（从现态到次态）
  - 有向弧上标明外部输入和输出（如果有），圆圈内也可以标明当前状态的输出（如果有）。

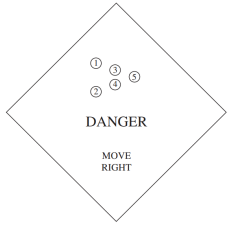
Example of FSM



我们可以画出状态图：

（来自书 P87）这里有 5 个路灯，最开始都是关闭的。如果开关是 ON，那么灯会做如下变化：第一个单位时间内灯都是熄灭的，下一个单位时间 1、2 号灯会亮起，再下一个时钟 3、4 号灯亮起，再下一个时钟 5 号灯亮起（现在所有灯都亮起），再下一个时钟所有灯都熄灭，如此循环。如果开关是 OFF，则所有灯熄灭。

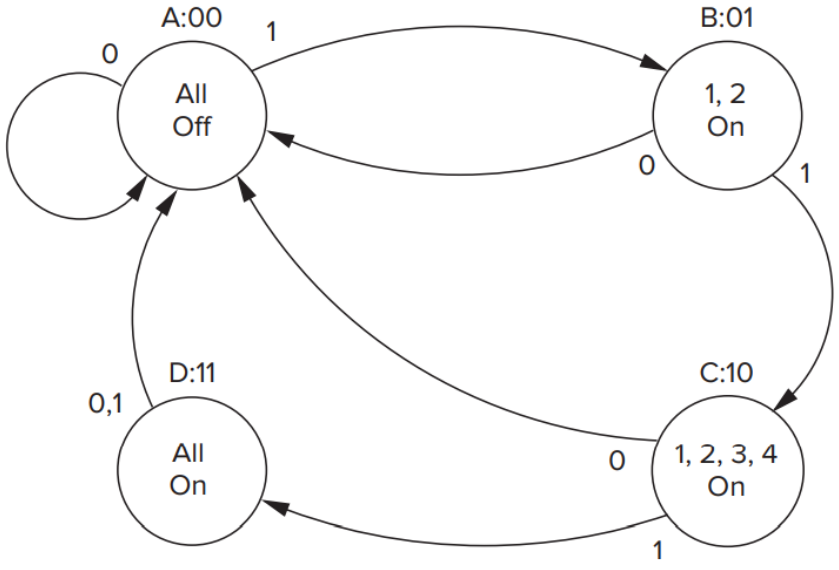
Example of FSM



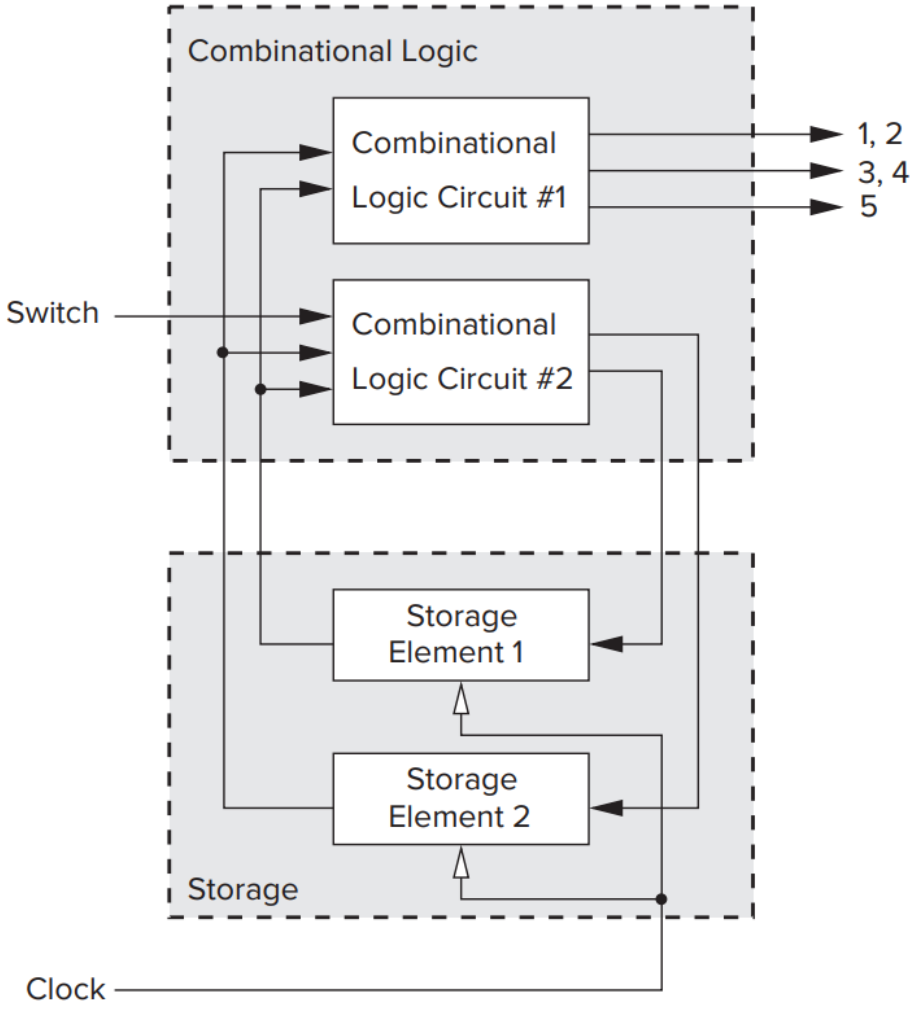
（来自书 P87）这里有 5 个路灯，最开始都是关闭的。如果开关是 ON，那么灯会做如下变化：第一个单位时间内灯都是熄灭的，下一个单位时间 1、2 号灯会亮起，再下一个时钟 3、4 号灯亮起，再下一个时钟 5 号灯亮起（现在所有灯都亮起），再下一个时钟所有灯都熄灭，如此循环。如果开关是 OFF，则所有灯熄灭。

我们可以画出状态图：

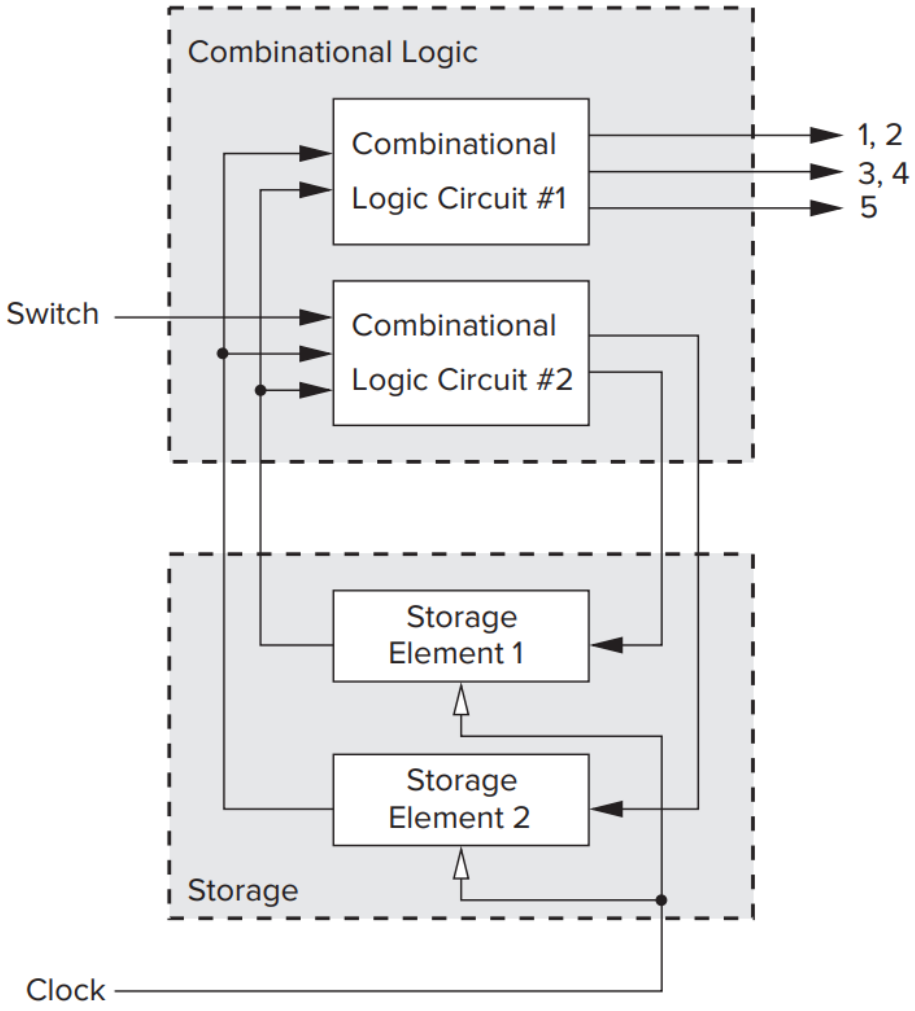
这里我们用两位的二进制数代表 A、B、C、D 四个状态，有向弧上的数字表示开关（1 表示开关为 ON）。



Example of FSM (Cont.)



Example of FSM (Cont.)



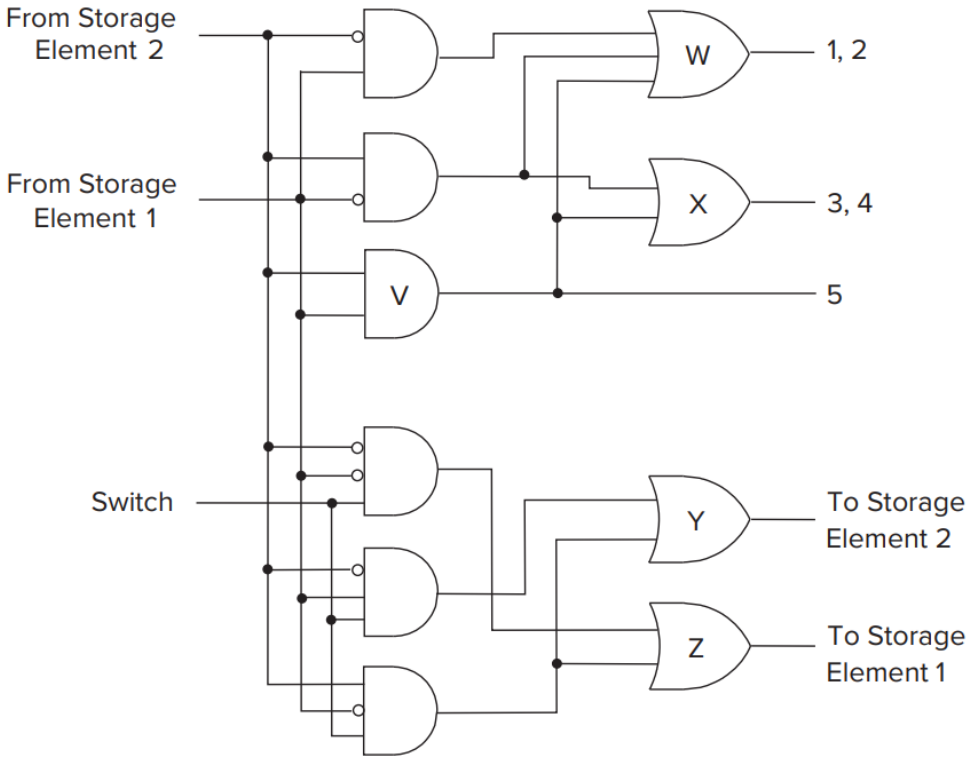
我们用 $U(t)$，$V(t)$ 分别表示时刻 t 时，存储单元 2，1 中的值。 通过状态变迁我们可以得到关于 U, V 真值表：

（前提：开关为 ON）

$U(t)$	$V(t)$	$U(t+1)$	$V(t+1)$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Example of FSM (Cont.)

把真值表利用 PLA 的方式转化成组合电路，得到如图：



Advanced Storage Elements

锁存器不足以满足我们的存储需求，需要更高级的存储单元 —— **Flipflop** 触发器。

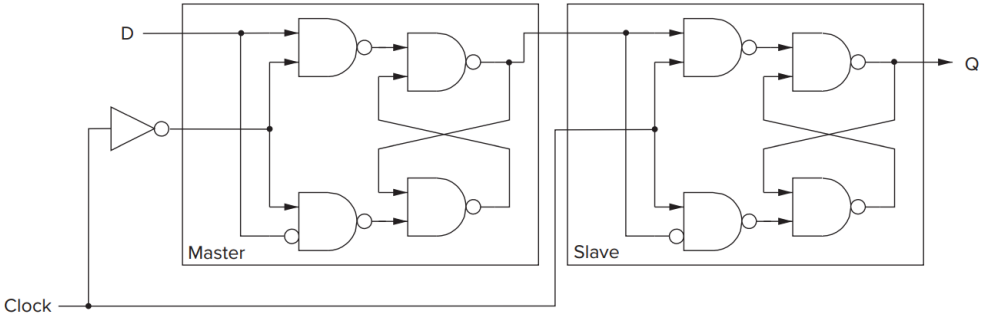
在刚刚的电路图中，存在问题：如果开关为 ON，那么我的状态（存储单元内的值）改变后，组合逻辑电路的输入也会改变，从而组合逻辑的输出也会随之改变。这样的变化是非常快的，没有一个稳定的状态。

首先，我们的电路可以分为两种类型

- **asynchronous** 异步电路，一旦外部输入改变，内部的状态和输出就立即改变。
- **synchronous** 同步电路，我们有一个时钟 clock，时钟不断地从高电平经过固定的时间到低电平，再经过固定的时间到高电平。从高电平到下一次高电平的时间称为 clock cycle 时钟周期。
  在同步电路中，FSM 每个时钟周期只能有一次状态变迁。

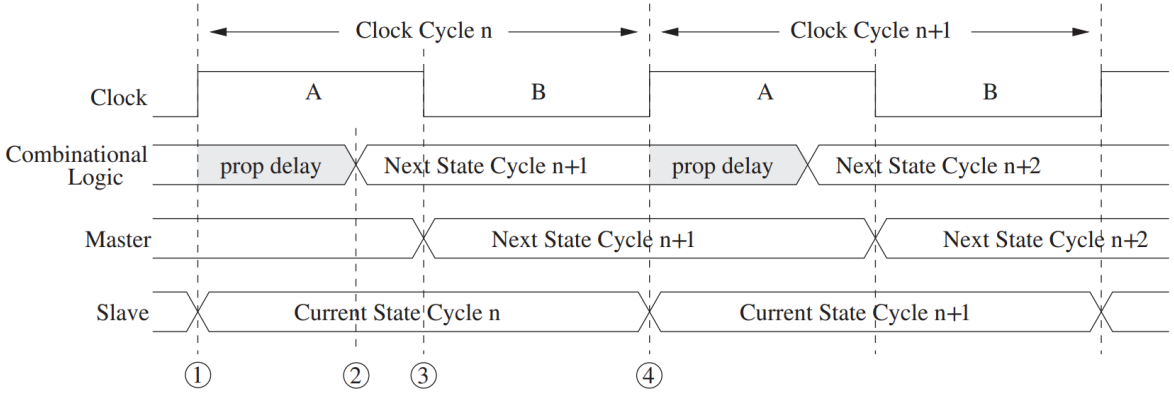
Master-Slave Flip-Flop

Master-Slave Flip-Flop 主从触发器是由两个 D 锁存器实现的，clock 时钟信号相当于 D 锁存器中的 WE 信号。右边的锁存器称为 *slave*，左边的锁存器称为 *master*。



- 在时钟上升沿 master 被 disable，这时 slave 读到 master 的数据，存入，并通过 Q 输出，Q 输出后会经过组合电路运算并来到 D（即 master 的输入）
- 在时钟下降沿 slave 被 disable，这时 master 读取 D 输入并存入，同时输出来到 slave 端的入口等待。

Time Diagram of Master-Slave Flip-Flop



谢谢大家

Question?