第3章

组合逻辑电路的分析与设计

第 2 章介绍了数的编码和布尔代数的相关知识，本章将对组合逻辑的表示、分析和设计方法进行深入的讲解。同时，本章将引入几种常见的组合逻辑电路模块的设计实例展开进一步的分析和讲解，对数字系统的实现进行细致的讨论。本章的主要内容总结概括为图 3-1 的思维导图。第 2 章介绍了布尔逻辑的表示、化简以及具体的电路实现，本章建立在基础逻辑门之上，进一步讨论如何分析和设计具备特定功能的组合逻辑电路。

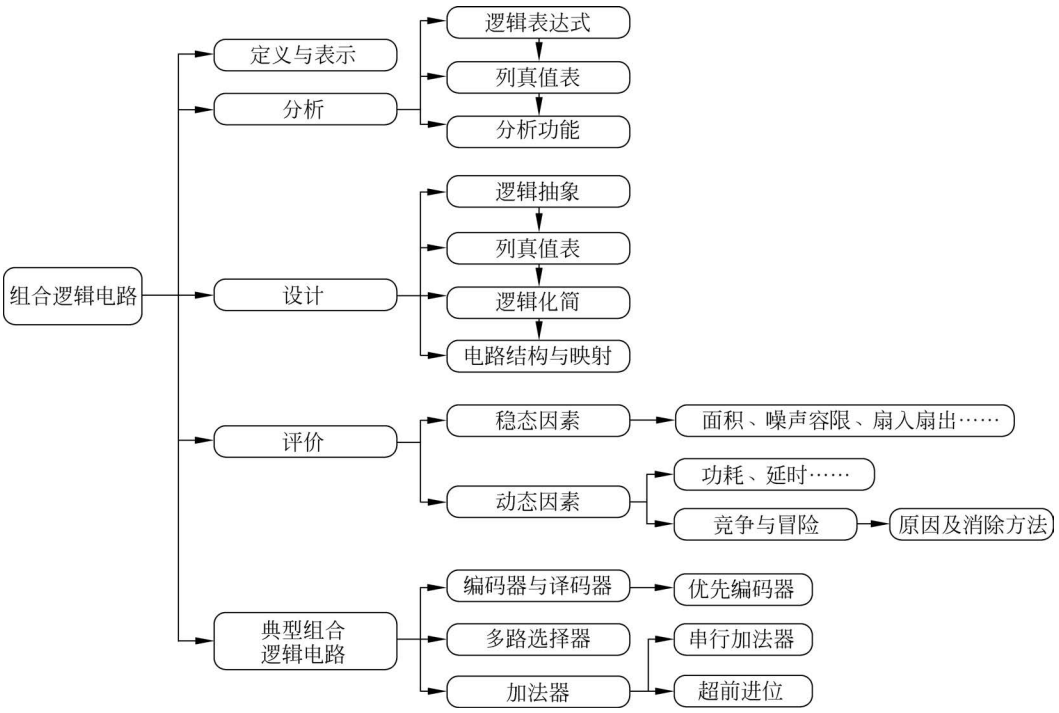


图 3-1　本章思维导图

为了更好地设计组合逻辑电路，首先需要对组合逻辑电路的基本定义和分析方法进行介绍，主要包括功能和性能层面的分析方法，从而可以针对一个电路进行解构和评价。然后，本章将介绍一个标准的组合逻辑电路的设计流程，包括对具体算法或问题进行逻辑抽象，继而通过列真值表等手段对逻辑进行化简，最后完成实际电路结构的映射。

设计完成后，对于一个组合逻辑电路的好坏，需要制定关键的评价指标和评价方法，包括稳态因素和动态因素，指导组合逻辑电路的调整，实现具体场景下的电路优化。另外，在组合逻辑中存在固有的属性——竞争和冒险，由于不同电信号在线路中的传播会有竞争现象，导致电路输出可能出现瞬态的错误结果，从而引发一系列问题，本章也将对这一现象的产生原因和解决方案做相关介绍。最后，基于以上基础知识，进一步介绍一些更为复杂、更贴近实际应用的典型组合逻辑电路设计实例，例如多路选择器、编码器、解码器、加法器等，并介绍这些组合逻辑电路在实际计算机芯片中的应用。

总体来说，本章希望帮助读者掌握基本的组合电路分析和设计方法，熟悉常见的组合电路的设计规范和背后的设计考量，以及在实际数字系统中所发挥的作用，为后续更深入的学习打下良好的基础。

3.1　从布尔表达式到数字逻辑电路的构建

布尔函数能够表述输出和输入的逻辑映射关系，而要完成布尔表达式的运算，需要实现相应的组合元件（例如逻辑门）映射，再使用互联元件将这些组合元件连接成电路。那如何实现一些基本的逻辑门器件呢？20 世纪早期的系统，如电话交换网络，是由电子和机械模块组成的，主要包括开关（switch）和继电器（relay）[①]。到 20 世纪中期，真空管开始用于实现简单的门逻辑，取代了原有的机械开关。真空管[②]是一类电子控制开关，配合适当的电阻和电容可以实现逻辑门。相较于传统的机械开关，真空管速度快、功耗低且造价便宜。

1947 年晶体管被发明，也取代了真空管成为广泛应用的器件。晶体管是类似于真空管的半导体，在正确的电学条件下，可以起到闭合开关的作用。晶体管相比于真空管，体积更小，并且在速度、功耗和可靠性方面都更有优势。晶体管技术如今已经十分成熟，尺寸也在不断微缩，目前可量产的最先进制程已经达到 5nm，使大规模集成电路成为可能，集成电路的性能和功能也得到巨大提升。

晶体管的发明也极大促进了数字电路的发展。在第 1 章也提到过数字电路相比于模拟电路的许多优势。例如，数字电路的稳定性好、可靠性强；数字信号更易传输和存储；数字电路设计层次化明确，能够实现较高程度的自动化设计；能够与布尔逻辑进行对应，利用简单的门组合可以实现完备的逻辑运算；适合尺度缩减，基于 CMOS 器件的数字电路功耗极低、面积极小，能够大规模集成；等等。因此，越来越多的处理器开始对信号进行数字化，并使用逻辑的运算方式完成各种运算。

需要强调的是，虽然目前数字电路占据主流，但模拟电路在许多领域依然发挥着不可替代的作用。一方面，很多信号都源自自然界，例如声音、图像、电磁波等，这些信号都是以模拟的形式存在的，要针对这些信号进行采集和处理，就必然涉及一些模拟域的计算或从模拟域到数字域的转换；另一方面，在一些特定的场景下，模拟计算具备更高的处理能效、更快的处理效率。比如近年来神经网络的大规模应用，也重新让人们思考数模混合矩阵计算的

---

① Morse S E B. Improvement in the Mode of Communicating Information by Signals by the Application of Electromagnetism: U. S. ,1647[P],1840.

② Fleming J A. Instrument for Converting Alternating Electric Currents into Continuous Currents: U. S. ,803684[P],1905.

优势和潜力，甚至人们已经开始跨入光电混合或纯光学的模拟计算领域。由于本书主要介绍数字系统的设计与分析，因此不会过多地涉及模拟计算的相关内容和知识，如读者有兴趣，可自行查阅相关文献。

数字电路是构建计算机系统的基础，而组合逻辑电路则是实现数字电路的重要部分。通过组合逻辑能够实现输出和输入的与时间无关的函数映射，再辅以一定的记忆单元和时钟就可以构建时序逻辑电路，以实现更复杂算法的计算。因此，本章的学习一方面承接布尔逻辑的具体应用，另一方面也为后续更复杂的时序逻辑电路以及计算机系统设计打下基础。

3.2 组合逻辑的定义与表示

3.2.1 组合逻辑的定义

组合逻辑是用于设计和实现当前输出仅取决于当前输入的**逻辑函数**。换句话说，组合逻辑系统是**无记忆的**，它的**输出**不受历史输入的影响，这是与后续章节介绍的时序逻辑最本质的区别之一。回顾第 1 章所提到的，对于一个计算问题，可以通过三个基本部分进行描述：问题的输入 x，问题的输出 y 以及输入数据到输出数据的计算函数 $f(\cdot)$。而一个组合逻辑电路可求解的问题，其抽象出的计算函数 $f(\cdot)$一定是与时间无关的，且不含任何历史输入的信息。

一个典型的组合逻辑计算单元结构如图 3-2 所示，组合逻辑运算单元负责完成输出逻辑和输入逻辑的函数映射关系。可以看到，整个框架中并无任何与时间或以往状态相关的变量，输出仅取决于当前的输入逻辑值以及运算单元实现的逻辑映射关系。并且，这里的输入和输出均为逻辑变量，即由 0、1 组成。

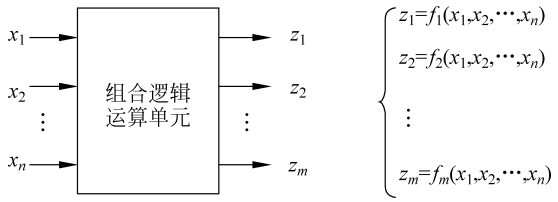


图 3-2 组合逻辑计算单元结构

3.2.2 组合逻辑的表示

1. 两级逻辑

组合逻辑的表示可以有多种形式。在第 2 章所介绍的真值表、卡诺图、逻辑门电路、逻辑表达式等都可以作为组合逻辑的表示方式，只要能够清晰地描述出输出与输入的逻辑关系即可。在第 2 章中引入了一种规范的表示方法：与或两级逻辑(SOP)。这种表示形式利用或函数作为第二级门逻辑，将与函数作为第一级门逻辑进行级联，从而表示逻辑函数。与之相对应的是另一种规范形式：或与两级逻辑(POS)。顾名思义，即利用与函数作为第二级门逻辑，将或函数作为第一级门逻辑进行级联来表示逻辑函数。需要注意的是，在认为输入的信号同时提供正信号和反信号的情况下，输入的取反将不计为一级逻辑门；但若反信号需要通过一个非门来对正信号进行求解，则需要被计算为一级逻辑门。在本章中，若无特

别说明，默认输入变量取反不作为一级逻辑门。如：

$$Y=f(A,B,C)=AB+BC+ABC$$

就是一个典型的两级 SOP 逻辑表达式。理论上，两级 SOP 或 POS 形式能够表示任何组合逻辑，两级逻辑的优势是在门逻辑器件充足且器件的扇入扇出（扇入与扇出的具体含义参见 3.5.1 节）无限制的情况下，能够在两个门逻辑的延时内完成任意逻辑运算。并且，在利用第 2 章提到的常见化简方法（例如卡诺图化简、QM 算法化简等）对逻辑表达式进行化简时，最终得到的化简表达式都是以两级逻辑呈现的。

2. 多级逻辑

在了解了两级逻辑后，自然与之对应的就是多级逻辑，这里的“多级”通常是指三级及以上的级联逻辑。比如上述例子，略加变形就会成为一个三级逻辑：

$$Y=f(A,B,C)=A'B+BC+AB'C'=(A'+C)B+AB'C'$$

仔细观察可以发现，在这种情况下，多级逻辑与两级逻辑相比，使用的与门减少了一个，但是会增加一个逻辑门的延时。希望读者能够形成一个基本概念：组合逻辑的优化往往不是由单方面因素驱动的，许多情况下面积和延时无法同时达到全局最优，需要根据电路设计的实际条件约束（例如面积约束或延时约束），选择相应的优化方式以实现既定功能的逻辑实现。

多级组合逻辑本质上是将一个简单逻辑函数表达为复合函数，其基本形式为

$$f(A,B,C,\cdots)=f(g(\cdots),h(\cdots),\cdots)$$

可以看到，外部函数可以形成内部函数的一个两级逻辑形式，而内部函数又可以继续拆解成复合函数，从而逐层形成复杂的多级逻辑。以下是一个多级逻辑拆解的示例。

【例 3-1】 $F=abc+abd+a'c'd'+b'c'd'$。

解：首先构造 $X=ab$，$Y=c+d$，可以得到

$$\begin{aligned}F&=abc+abd+a'c'd'+b'c'd'\\&=ab(c+d)+(a'+b')c'd'\\&=ab(c+d)+(ab)'(c+d)'\\&=XY+X'Y'=X\odot Y\end{aligned}$$

于是就形成两个复合函数输出结果的同或（⊙）形式。以此类推，更为复杂的多级函数本质上都是多层级的复合函数表达，类似因式分解，通过引入中间层次来构造新的函数。

由于现实应用的复杂性，仅仅使用两级逻辑通常无法满足功耗和面积等因素制约下的实现要求，因此通常会通过多级逻辑的级联来对逻辑进行分层次的解决和处理。一方面，多级逻辑能够共享中间层的计算结果，能够在一定程度上降低电路的代价；另一方面，多级逻辑的优化和化简通常较为复杂，依靠人工仅能处理小规模的问题，这也促使了逻辑综合工具的诞生。

3.3　组合逻辑电路的分析

了解了基础的门电路实现方法后，对于一个由逻辑门构成的复杂电路，如何分析这个电路的功能呢？本节将介绍常见的组合逻辑电路的分析方法，对以上问题进行解答。

组合逻辑电路的分析在于找到输出和输入的逻辑映射关系，也就是找到计算函数

$f(\cdot)$的具体形式，并且能够复原该函数所求解的问题。组合电路的分析是一个从低层次向高层次解析的过程：基础器件→电路结构→抽象算法。在得到一个电路图以后（这个电路图可能具有不同的抽象层次，例如以晶体管级、门级或者以功能模块来呈现），如何分析出其实现的功能是一项需要掌握的、极为重要的基础能力。

一般来说，分析一个较为复杂的组合逻辑电路所实现的功能可以遵循以下三个基本步骤。

- 列出逻辑表达式：观察输入到输出之间逻辑门的类型和连接关系可以推导出电路对应的逻辑表达式，得到 $f(\cdot)$的函数形式。
- 列出真值表：根据布尔表达式，列出真值表或卡诺图。逐个分析输出信号的逻辑组合对应的输出信号，并填充到真值表或卡诺图中相应的位置。
- 分析真值表：对真值表或卡诺图进行分析，确定其功能。对于一些复杂的、不常见的功能，这一步有时候可能很难直接看出其功能，因此还需要结合一些设计经验辅助判断。

接下来用一些实例来展开讲解。

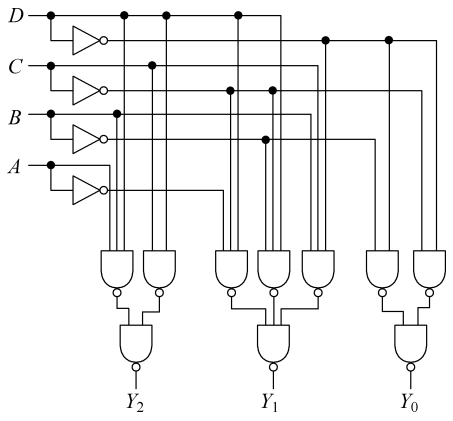


图 3-3　示例电路

【例 3-2】 请分析图 3-3 所示组合逻辑电路的功能。

解：按照三个步骤对该组合逻辑电路进行分析。

第一步，列出逻辑表达式。根据门的逻辑功能和连接关系，可以写出布尔表达式如下：

$$Y_2=((DBA)'(DC)')'=((DBA)')'+((DC)')'=DBA+DC$$

$$Y_1=((DC'A')'(DC'B')'(D'CB)')'=DC'A'+DC'B'+D'CB$$

$$Y_0=((D'B')'(D'C')')'=D'B'+D'C'$$

第二步，列出真值表。根据第一步中的布尔表达式，可以列出真值表（见表 3-1）。

表 3-1　例 3-2 的真值表

D	C	B	A	Y_2	Y_1	Y_0	D	C	B	A	Y_2	Y_1	Y_0
0	0	0	0	**0**	**0**	**1**	1	0	0	0	**0**	**1**	**0**
0	0	0	1	**0**	**0**	**1**	1	0	0	1	**0**	**1**	**0**
0	0	1	0	**0**	**0**	**1**	1	0	1	0	**0**	**1**	**0**
0	0	1	1	**0**	**0**	**1**	1	0	1	1	**1**	**0**	**0**
0	1	0	0	**0**	**0**	**1**	1	1	0	0	**1**	**0**	**0**
0	1	0	1	**0**	**0**	**1**	1	1	0	1	**1**	**0**	**0**
0	1	1	0	**0**	**1**	**0**	1	1	1	0	**1**	**0**	**0**
0	1	1	1	**0**	**1**	**0**	1	1	1	1	**1**	**0**	**0**

第三步，分析真值表。对第二步得到的真值表进行分析，确定电路功能。

从真值表的输出取值可以观察到，随着输入的 4 位二进制数的大小变化，输出的值也随

之发生阶段式的改变，可以看出这是一个判断输入范围的电路，即判断输入的 4 变量所组成的二进制数所对应的十进制数值的范围，具体划分如下：

$$DCBA：0 \sim 5，\quad Y_0 = 1$$

$$DCBA：6 \sim 10，\quad Y_1 = 1$$

$$DCBA：11 \sim 15，\quad Y_2 = 1$$

【例 3-3】 分析图 3-4 所示组合逻辑电路的功能。

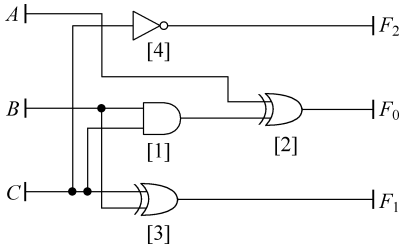


图 3-4　示例电路

解：依然通过分析的三个基本步骤来完成这个组合电路的分析。

第一步，列出逻辑表达式。根据门的逻辑功能和连接关系，可以写出布尔表达式如下：

$$F_0 = A \oplus (BC)$$

$$F_1 = B \oplus C$$

$$F_2 = C'$$

第二步，列出真值表。根据第一步中的布尔表达式，可以列出真值表（见表 3-2）。

表 3-2　例 3-3 的真值表

A	B	C	F_0	F_1	F_2
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

第三步，分析真值表。对第二步得到的真值表进行分析，确定电路功能。可以通过 $F_0F_1F_2$ 和 ABC 的对应关系得知这是一个 3 位的计数器，即

$$(F_0F_1F_2) = (ABC) + 1$$

需要注意的是，根据真值表确定功能往往需要分析输出和输入的对应关系，并找出变化的规律和规则。但是，在一些情况下，真值表无法直观地推理出对应的功能，这时需要结合实际的应用和经验来展开分析。

【例 3-4】 分析图 3-5 所示组合逻辑电路的功能。

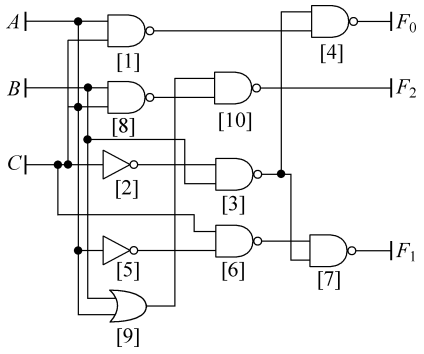


图 3-5 示例电路

解：依然使用三个基本步骤来分析这个组合逻辑电路的功能：

第一步，列出逻辑表达式。根据门的逻辑功能和连接关系，可以写出布尔表达式如下：

$$F_0 = ((AC)'(BC')')' = AC + C'B$$

$$F_1 = ((A'C)'(BC')')' = A'C + BC'$$

$$F_2 = ((AB)'(A + B))' = A \odot B$$

第二步，列出真值表。根据第一步中的布尔表达式，可以列出真值表（见表 3-3）。

表 3-3 例 3-4 的真值表

A	B	C	F_0	F_1	F_2
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

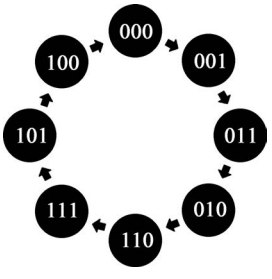


图 3-6 例 3-4 的状态转移图

第三步，分析真值表。对第二步得到的真值表进行分析，确定电路功能。在这个例子中，直接观察真值表很难直观地看出它所实现的功能。但是更细致地观察后可以发现 $F_0F_1F_2$ 和 ABC 之间总是只差一位，由此可以联想到格雷码。将真值表的变化画成状态图后可得图 3-6。

由图 3-6 可以看到，输出总是输入的下一个格雷码的编码形式。所以这是一个格雷码递增的编码器，给定输入的格雷码，就可以输出下一个格雷码的编码形式。

3.4　组合逻辑电路的设计

在了解组合逻辑电路的分析方法后，还需要了解组合逻辑电路的设计方法。面对一个具体的计算问题，如何一步步地给出组合逻辑电路的实现，即给出计算函数 $y=f(x)$ 的电路实现。设计的过程与分析的过程相反，从基本的抽象函数出发，分析出其电路结构，再映射到基础的逻辑器件。但是，设计过程要比分析过程困难很多。对于相同的功能要求，可能存在多种不同的设计，如何让自己的设计最简洁、最高效、功耗最低、最实用，是一个永远值得思考的话题。本节将介绍组合逻辑设计的一般方法和思想。

一般来说，组合逻辑的设计过程包括以下五个步骤。

- **逻辑抽象**：要将一个问题 $y=f(x)$ 映射到电路，首先要对问题做抽象和概括，明确需求电路的输入和输出，对输入和输出进行逻辑变量的定义，并得到输出和输入的逻辑对应关系。
- **列出真值表**：基于抽象出的逻辑和输入输出的对应关系，列出真值表。真值表是为了能够清晰地表示输出与输入的关系，来帮助后续逻辑函数的提取、化简和针对性的优化。
- **逻辑化简**：基于真值表，列出每个输出变量关于输入变量的卡诺图，利用卡诺图进行逻辑函数化简，从而得到每个输出变量的逻辑函数表达式。
- **确定逻辑电路结构**：得到函数表达式以后，就可以确定逻辑电路的结构。
- **映射成基础器件**：再将基础逻辑器件对应到电路结构中的相应组件即可得到最终的逻辑电路。

接下来通过一个比较器的设计实例来更直观地感受以上过程。

【例 3-5】　请设计一个比较器，比较两个 2 位二进制数的大小。

解：遵循组合逻辑电路设计的五个基本步骤进行比较器的设计。

第一步：逻辑抽象。

首先明确比较器的功能：比较两个 2 位二进制数的大小。因此，输入就是两个 2 位二进制数 A 和 B，记作 A_1A_0 和 B_1B_0；输出则是关于大小的判断，一共有三种情况，则可以用三个变量进行表示，LT 表征 $A<B$，EQ 表征 $A=B$，GT 表征 $A>B$，当各自的条件为真时，相应的变量输出 1，其他变量则为 0。当然，这里的输出规定并不是一成不变的，也可以对各种情况进行编码。例如，也可以使用 2 位输出编码格式：00 指代小于，01 指代等于，10 指代大于。此时需要在输出端再接一个解码器对输出结果进行解析（对 2 位输出数据进行解码判断是小于/等于/大于），原理上等同于增加一级逻辑，效率上可能会有所差距。总之，对明确功能的逻辑抽象是多种多样的，但通常情况下会选择较容易理解且效率最高的方案（如图 3-7 所示使用 LT、EQ、GT 进行输出编码表示）。

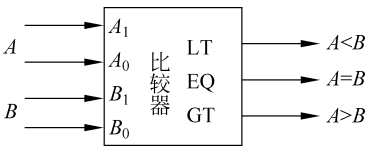


图 3-7　比较器功能示意图

第二步：列出真值表。

根据上述的输入和输出变量定义，可以得到变量之间的逻辑对应关系，这一步较为简单，只需要根据功能规则逐个计算得到输出和输入的对应关系即可。在本例题中，列出真值

表如表 3-4 所示。

表 3-4　比较器的真值表

A_1	A_0	B_1	B_0	LT	EQ	GT	A_1	A_0	B_1	B_0	LT	EQ	GT
0	0	0	0	0	1	0	1	0	0	0	0	0	1
		0	1	1	0	0			0	1	0	0	1
		1	0	1	0	0			1	0	0	1	0
		1	1	1	0	0			1	1	1	0	0
0	1	0	0	0	0	1	1	1	0	0	0	0	1
		0	1	0	1	0			0	1	0	0	1
		1	0	1	0	0			1	0	0	0	1
		1	1	1	0	0			1	1	0	1	0

第三步：逻辑化简。

根据真值表画出每个输出变量对应的卡诺图，如表 3-5 所示。

表 3-5　左：LT 信号的卡诺图；中：EQ 信号的卡诺图；右：GT 信号的卡诺图

A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

化简过程略，化简后得到每一项输出关于输入的布尔逻辑表达式：

$$\mathrm{LT}=(A<B)=A'_1B_1+A'_1A'_0B_0+A'_0B_1B_0$$

$$\mathrm{EQ}=(A=B)=A'_1A'_0B'_1B'_0+A'_1A_0B'_1B_0+A_1A'_0B_1B'_0+A_1A_0B_1B_0$$

$$=(A_1\oplus B_1)'(A_0\oplus B_0)'$$

$$\mathrm{GT}=(A>B)=A_1B'_1+A_0B'_1B'_0+A_1A_0B'_0$$

第四步确定逻辑电路结构，第五步映射成基础器件。

在获得布尔逻辑表达式后，就可以确定逻辑电路结构，映射成基础门器件，如图 3-8 所示。

【例 3-6】 请用尽量少的输入变量和不超过 4 个的“或门”和“与非门”实现

$$Y(A,B,C,D)=\sum m(0,2,4,6,9,13)+\sum d(10,11,14,15)$$

解：由于本题中已经给出了要实现的逻辑函数形式，且以最小项形式给出，因此第一个步骤逻辑抽象和第二个步骤列出真值表可以跳过，直接利用卡诺图来进行化简。

卡诺图如表 3-6 所示。

表 3-6　例 3-6 的卡诺图

AB \ CD	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	0	1	x	x
10	0	1	x	x

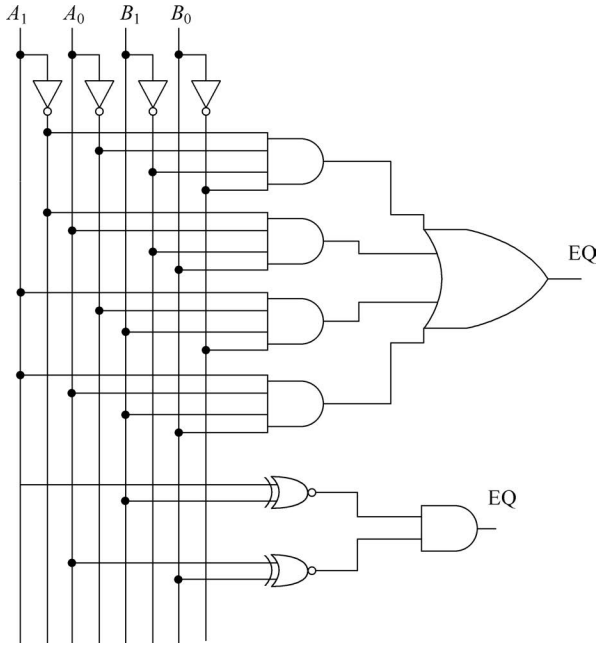


图 3-8　比较器的电路图

从而可以得知，F 的逻辑表达式可以简化为 $Y=AD+A'D'$，但是需要注意，题目中要求仅使用或门和与非门，因此利用德摩根定律对表达式进行转换，可以得到 $Y=((AD)'(A+D))'$。从而得到需映射到门逻辑器件的逻辑表达式，其电路实现如图 3-9 所示。

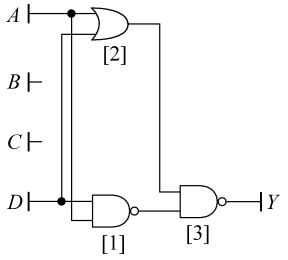


图 3-9　例 3-6 的电路实现图

拓 展 思 考

相信读者们也可以发现，目前给出的示例都是使用两级逻辑的结构来解决组合逻辑的设计问题。这种方法在实际应用中是作为一个简单模块的常用设计思路，因为复杂的布尔函数的表示复杂度是巨大的，且随逻辑变量数量的增长指数级上升。相信读者在做卡诺图化简时应该也有所体会，当输入逻辑变量超过 5 个时，人工完成化简已经很困难了，所以通常需要通过计算机辅助完成逻辑变量的化简。简单做个计算题，若输入有 N 个逻辑变量（即 N 位输入），输出有 1 个逻辑变量（即 1 位输出），则构建一个真值表所包含的项的数量为 2^N，每个输出项为 1 位，于是存储这样一个真值表需要 2^N 位的存储空间。当 $N=50$ 时，存储真值表的输出项占用的存储空间将会达到 2^{50} bit=131,072GB，这样规模的存储需求显然无法接受与满足。

那么最直观地解决规模问题的方法就是“分而治之”，或者说将一个大的逻辑计算模块拆分成更多的小模块，再将小模块级联以有效地降低电路的求解规模。从两级逻辑到多级逻辑是一个复合函数分解的过程，其实本质上也在拆解整个电路，通过引入中间层次的计算来显著降低整体的电路设计问题的规模。举个简单的例子，如果要设计一个两个 16 位数的加法器，输入总共为 32 位，输出为 16 位，若用两级逻辑来表示和设计，则真值表的规模将是 $2^{32}\times16$ 位，显然求解起来十分复杂。所以可以

将这个问题拆解为 16 个 1 位加法器(输入含进位)的级联来完成设计。这样需要设计的电路模块仅仅是一个 1 位的加法器,输入为 1 位的数据位和 1 位的进位输入,输出为该位的加法结果和 1 位的进位输出,再级联 16 个这样的加法器即可实现两个 16 位数的加法运算。可以看到,从两级到多级连接的计算模式肯定会在延时上产生巨大的影响,因此绝大多数情况会做一些速度和开销的权衡,例如引入超前进位的加法器,将一部分进位信号的运算使用直接的求解法来大幅缩小延时。

3.5 组合逻辑电路的评价

在了解了如何分析和设计特定功能的组合逻辑电路后,还需要对电路的性能进行客观合理的评价。本节将介绍常见的组合逻辑的评价指标。关键的评价指标因素通常分为两类:稳态因素和动态因素。其中稳态因素是电路和门的固有属性,包括允许输入的逻辑电平范围、噪声容限、门的扇入和扇出系数、面积等;动态因素是电路在运行时的主要特征,主要考虑速度、功耗和竞争冒险现象。

3.5.1 稳态因素

1. 逻辑电平

数字电路通过对电平进行二值量化实现 0、1 的数字化,但是门电路要正确地进行区分,需要对输入电平的范围进行约束。如图 3-10 所示,当电平处于$[V_{Hmin},V_{Hmax}]$时,对应逻辑 1;当电平处于$[V_{Lmin},V_{Lmax}]$时,对应逻辑 0;中间的范围是禁止电压范围,因为在此范围内,电路无法正确地区分 0、1 逻辑,会影响输出的稳定性。当然也存在少部分情况将高电平定义为逻辑 0,低电平定义为逻辑 1,也称为“负逻辑”。对于门的输入电平需要有一定的范围限制。一般情况下,逻辑 0 和逻辑 1 对应的电压范围越宽,其抗干扰能力越强。但是,为了实现更低功耗,目前使用的电源电压也在不断下降,这导致逻辑电压范围越来越窄,因此对电子元件的参数精度与电源稳定性的要求越来越高。

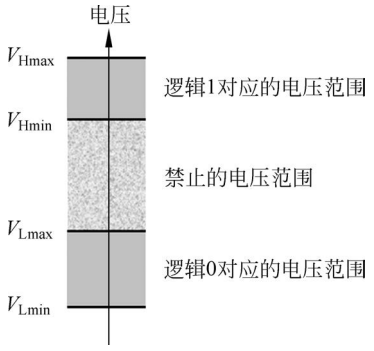


图 3-10 逻辑 0/1 对应的电压范围

对于不同的工艺、不同的工作电压,逻辑 0/1 对应的电压范围不是一成不变的,例如图 3-11 展示的是一些典型的电平范围①。

① Texas Instrument . How to select little logic.

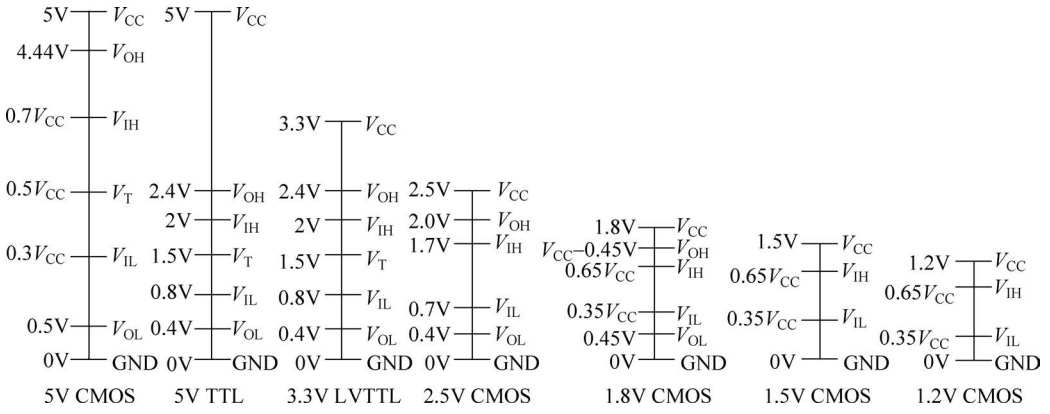


图 3-11　逻辑电平“家族”

2. 噪声容限

上面提到了门电路对于输入的逻辑 0 和逻辑 1 电压有一定的范围限制。自然而然地，在一个级联的门逻辑电路中，上一级的输出电压需要输入到下一级门，因此每一级门的输出都需要满足下一级门的输入电平的允许电压范围约束。但是，一个逻辑门通常无法保证输出精确的逻辑高/低电平值，其输出也会产生一定的波动。因此，需要保证前一级门的输出在最坏的情况下，在受到各种噪声的影响下，依然可以落在下一级门的正确电压输入范围内。这个能够允许的噪声幅度就称为噪声容限。

在数字电路中，噪声容限一般定义为上一级门输出电压和下一级门输入电压之间的可容忍电压差范围。如图 3-12 所示，V_o 表示上一级逻辑门的输出电压，其正常范围落在$[V_{OHmin}, V_{DD}]$(逻辑 1)和$[0, V_{OLmax}]$(逻辑 0)中。对于下一级的允许输入电压范围为$[V_{IHmin}, V_{DD}]$(逻辑 1)和$[0, V_{ILmax}]$(逻辑 0)，因此这里的噪声容限就是 $V_{NH}=V_{OHmin}-V_{IHmin}$(逻辑 1)和 $V_{NL}=V_{ILmax}-V_{OLmax}$(逻辑 0)。噪声容限在一定程度上表明级联电路的抗干扰水平，为保证电路功能正确，需保证信号传输时引入的噪声在容限内。电路中的噪声来源有很多，例如热噪声、电磁噪声、地噪声、电源带来的噪声等，以及前后逻辑门之间互联线电阻等因素也会导致压降，因此在设计时应当考虑噪声带来的影响，并尽可能提高电路的噪声容限。

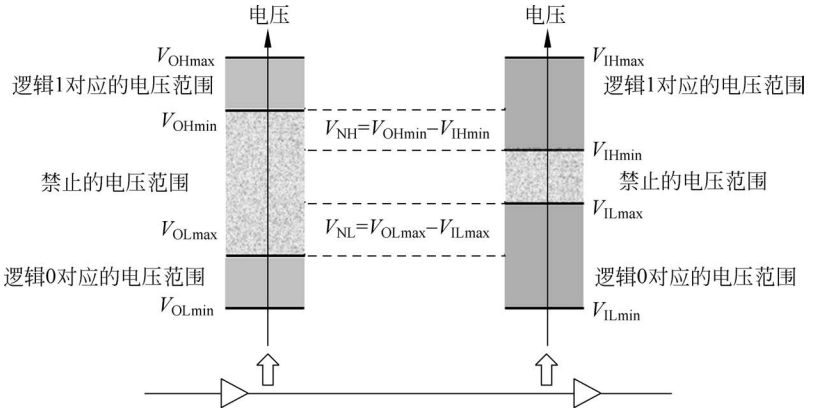


图 3-12　0/1 逻辑的噪声容限示意图

3. 扇入和扇出系数

一个逻辑门能够驱动的后级输入端数目是有上限的，在最坏的负载情况下，其最多能够驱动后级输入端的个数称为**扇出系数**(**fan-out**)①。通常情况下，TTL 电路的扇出系数一般小于 10，MOS 电路的扇出系数则不受负载影响，但是随着扇出的增大，外部负载电容增大，会导致输出端的信号需要更长的时间来完成信号准备(等效电容的充放电时间更长)，从而增加门延时，引起工作速度下降。如图 3-13 所示，若输出端连接的门个数增多，则会使外部负载电容增大，从而 V_{out} 的准备时间也将更长。

一个逻辑门的**扇入系数**(**fan-in**)表示其被前级模块调用的个数，即门电路允许的输入端的数目。一个模块的扇入系数也不宜过大，因为门电路输入端的增加，会使串联的 MOS 管总等效本征负载电容增加，增加输入信号准备的时间，也会增加门延时。如图 3-13 所示，若输入端的个数增多，则会使本征负载电容增大，从而 V_{in} 以及 CMOS 电路的准备时间也将更长。

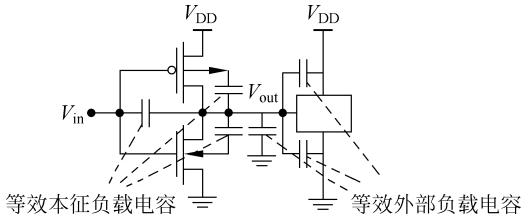


图 3-13　逻辑门的等效负载电容示意图

4. 面积

逻辑电路的面积也是设计时的一个重要考量。一方面，在一块固定面积的晶圆上刻蚀芯片，若单个芯片的面积越小，则最后这一块晶圆上可以切割出的芯片数量也越多，从而能够降低单片芯片的成本。另一方面，面积小的芯片也方便在系统中集成，尤其是在一些空间受限的设备中，如手机、传感器等。所以在设计电路时也需要把面积的代价考虑在内。逻辑电路的面积主要取决于两个方面，一是基础门器件和基础模块的面积占用，二是器件之间互联所产生的面积约束。在得到逻辑表达式后，还需要进行工艺映射(technology mapping)，把逻辑函数映射到实际的电路。当然，对于目前集成数亿至数百亿个晶体管的复杂芯片而言，人工去完成布局布线的优化已经不可能，因此通常借助 EDA 软件来完成，在例 3-7 中也会对 EDA 的用途和相关知识做简单的介绍。

3.5.2　动态因素

1. 时间评价指标

逻辑门无法产生理想的方波波形(如图 3-14(a)的波形)，即信号来临时刻输出就发生瞬变。逻辑门的时间评价指标主要有两个，一是逻辑门的输出从一个状态变化到另一个状态所需要的时间，称为转换时间(transition time)；二是从逻辑门的输入发生变化到相应的输出发生变化所需要的时间，称为延时(delay)。

转换时间包括上升时间 t_r(raising time)和下降时间 t_f(falling time)，分别表示输出从逻辑 0 变换到逻辑 1 的用时和逻辑 1 变换到逻辑 0 的用时。如图 3-15 所示，实际的波形并非理想的方波，信号的变换需要一定的时间。

① Gopalan K G. Introduction to digital electronic circuits，1996.

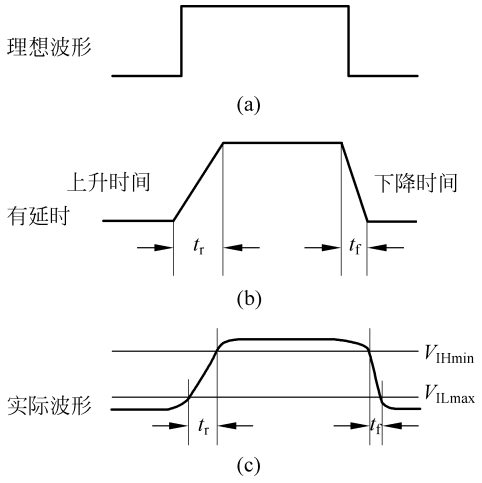


图 3-14　逻辑门的延时特性

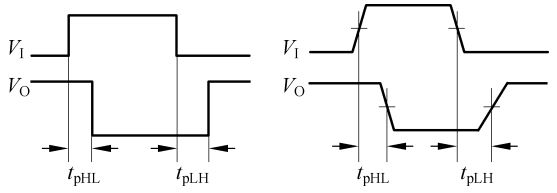


图 3-15　逻辑门的理想方波延时和实际波形的延时

延时是从输入发生变化到相应的输出发生变化所经过的时间。理想情况下，输入信号应当能够瞬时对输出产生影响，从而能使信号更快地传输。但实际情况下，输入的改变并不会立即引起输出的改变，而是经过一段时间后才会引起输出发生变化。图 3-15 是一个非门的延时示意图，输出信号和输入信号存在一个时间差。延时的计算通常用平均值、最大值或最小值表示，即表达式为 $t_p=1/2(t_{pHL}+t_{pLH})$或 $t_p=\max(t_{pHL},t_{pLH})$或 $t_p=\min(t_{pHL},t_{pLH})$。

延时会对电路带来一些影响，这些影响既有消极的一面，也有积极的一面。一方面，延时导致信号传输的到达时刻有差异，因此在一些情况下会导致电路的冒险(3.5.2 节将展开介绍)。并且由于延时的存在，在设计时序电路时需要对时钟信号进行严格的约束；另一方面，延时也可以用于产生环形振荡器。如图 3-16 所示，利用三个及以上的奇数个非门，将它们的输出端和输入端首尾相连构成环状，就可以产生具有一定周期的振荡信号，这类信号在时序电路中可以用作时钟信号。

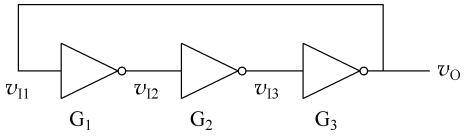


图 3-16　利用延时特性基于奇数个反相器构建周期信号

对于一个逻辑电路，其整体延时通常以**关键路径的延时**来计算。关键路径即从输入到输出所经过的最长延时路径，决定了一个逻辑电路的计算速度。这个概念在时序逻辑章节

中也会涉及，并且决定了时序电路所能使用的时钟频率边界。

2. 功耗和能量

功耗(power)是一项极为重要的评价指标，表示电路在单位时间中所消耗的能量，单位为 W(瓦特)。而能量(energy)则是衡量一个电路消耗的能量总数，单位为 J(焦耳)。首先介绍几个常见的概念：

(1) 瞬时功耗即在某一时刻的功耗，$P(t)=i_{DD}(t)V_{DD}$。

(2) 能量消耗即在一段时间内的总能量消耗，$E=\int_0^T P(t)\mathrm{d}t=\int_0^T i_{DD}(t)V_{DD}\mathrm{d}t$。

(3) 平均功耗即在一段时间内的平均功耗，$P_{avg}=E/T=1/T\int_0^T i_{DD}(t)V_{DD}\mathrm{d}t$。

衡量一个 CMOS 电路的功耗，主要关注动态功耗(dynamic power)和静态功耗(static power)。CMOS 电路在输入稳定时总有一个晶体管截止，所以静态功耗在理想情况下应该为零，但是由于漏电流的存在，实际上静态功耗不为零，因此静态功耗也称为泄漏功耗(leakage power)。CMOS 电路工作时则会产生动态功耗，主要由开关电流和短路电流引起。如图 3-17 所示，在 CMOS 电路中，当电路输入为 0 时，PMOS 导通，电源通过 PMOS 向负载电容充电；当电路输入为 1 时，NMOS 导通，负载电容通过 NMOS 向地放电。可以看到，开关电流就是在不断对负载电容充放电的过程中产生的，而短路电流则是在输入的转换过程引起 PMOS 和 NMOS 同时导通形成的。

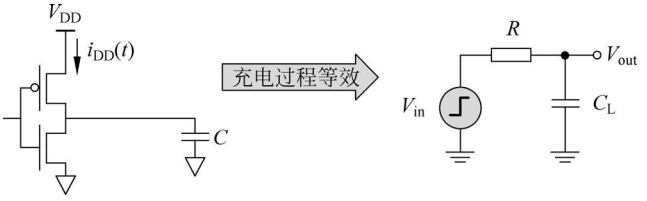


图 3-17　等效电容的充电过程示意

总结一下，总功耗的表达形式可以写成如下形式：

$$P_{total}=P_{dynamic}+P_{dynamic\_short}+P_{leakage}$$

实际芯片中，常见情况下动态功耗要远大于静态功耗。计算动态功耗可以用等效电容的充放电来近似得到①。首先，一次充电的总能耗可以计算如下：

$$E_{0\to1}=\int_0^T P(t)\mathrm{d}t=V_{DD}\int_0^T i_{supply}(t)\mathrm{d}t=V_{DD}\int_0^{V_{DD}} C_L\mathrm{d}V_{out}=C_LV_{DD}^2$$

因此，动态功耗可以计算如下：

$$P=E/\Delta T=E_{0\to1}f_{0\to1}=C_LV_{DD}^2f_{0\to1}=C_LV_{DD}^2\alpha_{0\to1}f$$

式中，C_L 为等效电容，$\alpha_{0\to1}$ 为 0 到 1 的翻转概率，f 为电路的工作频率。

如果想降低电路的功耗，从以上的参数可以得出结论，应当降低电路的工作电压和频率，但是降低频率会导致工作速度下降。所以现实情况下很难做到低功耗延时积，即很难同时达到功耗和速度的最优化。此时，需要结合应用场景的敏感指标来进行定制化设计，例如

① Soudris D, Pirsch P, Barke E. Integrated Circuit Design: Power and Timing Modeling, Optimization and Simulation: 10th International Workshop[C]. PATMOS 2000, Göttingen, Germany, 2000.

对于服务器端的设备，功耗的优先级可能不如速度高，就可以以降低延时为主要目标来设计电路；对于一些边缘端的设备，出于续航时间的考虑，功耗可能是比速度更需要考虑的，则可以针对降低功耗来进行定制设计。

拓 展 思 考

对于一些极为复杂的逻辑电路，在给定的元件库中，如何快速找到逻辑电路的最优实现方案呢？举个例子，对于例 3-7 中的表达式，在给定的单元库中，如何找到面积最小或延时最小的逻辑电路？

【例 3-7】 已知逻辑函数：$f(A,B,C)=\sum m(1,3,4,5)$，并有工艺库如表 3-7 所示，不考虑互联线和空间拓扑引起的面积或延时开销，仅考虑门电路占用的面积和延时，请设计相应的逻辑电路结构，分别使最终的面积/延时最小。

表 3-7　例 3-7 使用工艺库

逻辑门	延时/ps	面积/μm^2	逻辑门	延时/ps	面积/μm^2
NOT	20	10	AND4	90	40
AND2	50	25	NAND4	70	30
NAND2	30	15	OR4	100	42
OR2	55	26	NOR4	80	32
NOR2	35	16			

解：首先针对这个最小项的表达形式写出其逻辑表达式：$f=AB'+A'C$。

通过观察可以看到，NAND2 和 NOR2 的面积和延时均小于 AND2 和 OR2，所以应尽量使用最小的 NAND2 和 NOR2 来实现电路。所以可以对函数进行一个变形，列出尽可能多地使用 NAND2、NOR2 或混合使用的可能实现方案。

(1) $f=AB'+A'C=((AB')'(A'C)')'$

电路图如图 3-18 所示。

(2) $f=(A+C)(A'+B')=(A'C'+AB)'=(AB+(A+C)')'$

电路图如图 3-19 所示。

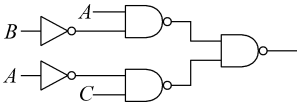


图 3-18　例 3-7 电路图(1)

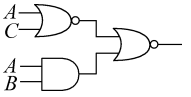


图 3-19　例 3-7 电路图(2)

分别看一下这两种实现方案的面积和延时，其中：

(3) 使用了 3 个 NAND2 和 2 个 NOT，且关键路径上共有三级门逻辑，故面积为 $2\times10+15\times3=65(\mu m^2)$，延时为 $20+30+30=80(ps)$。

(4) 使用了 2 个 NOR2，一个 AND2，且关键路径上共有 2 级，故面积为 $2\times16+25=57(\mu m^2)$，延时为 $50+35=85(ps)$。

所以综合以上分析，方案 1 具有最短的延时，方案 2 具有最小的面积。

虽然经过分析，能够给出该问题的考虑延时和面积的最优电路映射，但是不难看出，解

题方法中使用了启发式的搜索以及大量的前置经验来找出具备最优面积和延时的电路设计。但是，即使该问题中的解空间较小，也无法遍历所有的可能实现，面对更困难的电路优化问题时，依靠人工去搜索最优设计会耗费大量的人力。因此 EDA 技术被引入作为一种自动化的电路设计方法。EDA 通过计算机辅助的方式来帮助电路设计，以下是对 EDA 的简单介绍和相关参考文献。

【拓展知识】 EDA 与综合

EDA 是电子设计自动化，利用计算机软件来自动对电路进行优化和设计，具体过程包括综合、布局布线、仿真等。对于大型电路，很难通过人工去对电路的结构和组成形式进行优化，因此 EDA 技术能够大幅减轻电路设计工程师的劳动强度，提高电路设计的效率和性能。

EDA 技术中间的一个重要环节是逻辑综合，即将给定的硬件描述语言（VHDL）转换为门级表述，并自动化地在给定的优化目标（约束条件）下进行逻辑化简，有些逻辑综合工具还会整合布局布线的功能。由于本课程不涉及逻辑综合的相关内容，对相关内容感兴趣的读者可以参考以下书目：

[1] Damiani, Maurizio. Synthesis and optimization of synchronous logic circuits[D]. Diss. to the Department of Electrical Engineering. Stanford University, 1994.

[2] Iman, Sasan, Pedram M. Logic synthesis for low power VLSI designs[M]. Springer Science & Business Media, 1998.

3. 组合逻辑电路的竞争与冒险

冒险的定义。理想的组合逻辑电路应在任何时刻都符合门逻辑级联定义的输入输出关系，但由于门电路存在延时，导致信号从不同路径到来的时刻不同，因此到达电路中某一汇合点的时间有先后，称为竞争。这种信号的先后到来会引起在输出端的多余脉冲，使本应保持不变的输出值出现瞬时变化，这类瞬时脉冲信号称为毛刺，有可能产生毛刺的组合逻辑电路称为存在冒险（hazard，也称险象）①。冒险是不希望发生的现象，一方面，对于定义好的功能，如果输出发生了瞬时变化，有可能改变整个电路的状态，从而导致计算发生错误；另一方面，毛刺会导致不必要的晶体管反转，带来额外的动态功耗。注意，冒险是电路的一个固有特征，与电路的组成和结构相关，存在冒险的电路有可能产生毛刺，也可能不产生毛刺，取决于输入值以及电路的电特性。

以图 3-20 中的电路为例，该电路所实现的组合逻辑表达式为 $L=((A+B)'+A)'$，当 B 信号为 0 时，此时的表达式会简化为 $L=(A'+A)'$，理想情况下这个表达式的输出应始终为 0，但由于 $(A+B)'$ 或非门的存在，使 A' 信号会晚于 A 信号到达第二级的或非门。因此，当 A 信号从 1 跳变到 0 时，会存在第二级的与非门的输入同时为 0 的瞬时时刻，此时电路会输出一个 1，此即为毛刺，即该电路存在冒险。

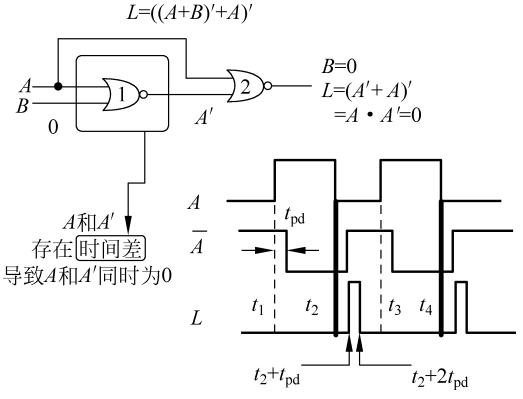


图 3-20 冒险产生的原因

① Unger S H. Hazards, Critical Races, and Metastability[J]. IEEE Transactions on Computers, 1995, 44(6): 754-768.

冒险的分类。组合逻辑电路的冒险分为两种基本类型：静态冒险与动态冒险。如图 3-21 所示，静态冒险是一个本应保持不变的输出经历了瞬时的状态转换，主要包括两种情况。

(1) 静态 1 冒险：本应保持为 1 的输出瞬时经历 0 状态。

(2) 静态 0 冒险：本应保持为 0 的输出瞬时经历 1 状态。

图 3-21　静态冒险的类型

图 3-20 冒险产生的原因中提到的例子就是一种静态 0 冒险，本应保持为 0 的输出，由于输入信号的改变，产生了瞬时的 1 脉冲信号。

动态冒险的定义是本应发生从 1 到 0 或者 0 到 1 单次跳变的输出信号发生不止一次的跳变。其形成的原因是多级电路往往存在多条路径，且这些路径的延时是不对称的。对于一个不存在静态冒险的多级电路网络，仍然有可能存在动态冒险。但由于动态冒险的分析较为复杂，因此本章不讨论这类冒险，感兴趣的读者可自行查阅相关资料。

【例 3-8】 请判断以下组合逻辑电路是否存在静态冒险，若是，请分析冒险类型；若不是，请说明原因。

(1) $L=AB+A'C$

(2) $L=(A'+B)(A+C)$

解：第(1)小题的电路图如图 3-22 所示。

由图 3-22 可以看到，虽然 A 和 A' 信号到达最后一级门的时间相同，但当 A 发生变化时，A' 的变化时刻要推迟一个非门的延时。所以当 $B=C=1$ 时，最后一级门的输入瞬时出现 A 和 A' 同时为 0 的情况，于是产生一个 0 脉冲，即该电路存在静态 1 冒险。

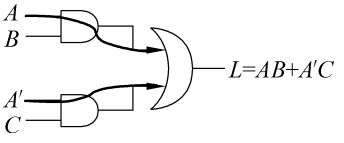


图 3-22　例 3-8 第(1)小题的逻辑电路

第(2)小题与第(1)小题类似，请读者尝试画出电路图进行分析。可以观察到虽然 A 和 A' 信号将同时到达最后一级，但由于 A' 和 A 信号变化不同步，会出现 A' 和 A 同时为 1 的情况。当 $B=C=0$ 时，且 A 从 0 变化到 1 时，就会出现瞬时的 1 脉冲，即该电路存在静态 0 冒险。

冒险的消除。冒险是不希望发生的现象，因为它是一个组合电路的不稳定因素，在某些情况下，产生了瞬时的错误输出，从而导致电路状态的改变。因此，需要设计相应的方法，在不影响正常功能的情况下消除冒险。常见的消除冒险的方式有两种，分别是添加冗余本原蕴含项和利用采样脉冲来消除冒险。

添加冗余蕴含项即在原逻辑表达式上增加一项冗余的本原蕴含项。来看例 3-8 中的第(1)小题例题，它的卡诺图如表 3-8 所示。

表 3-8　例 3-8(1)的卡诺图

A \ BC	00	01	11	10
0	0	1	1	0
1	0	0	1	1

根据前面的分析，当 ABC 由 111 变化至 011 时，由于非门延时有可能出现 A 和 A'同时为 0 的情况，此时本应输出 1 的情况下输出 0，电路存在冒险。注意，此时没有本原蕴含项能够覆盖输入的初始值(111)和最终值(011)。为了消除冒险，只需保证输入的初始值和最终值能被同一个本原蕴含项覆盖。这样做的原因在于，输入在同一个本原蕴含项内变化时，输入变量发生反相的变化与输出无关。例如本例中的 ABC 从 111 变化为 011，如果加入 BC 项，使得初始值和最终值被 BC 覆盖，即 $L=AB+A'C+BC$，此时当 $B=C=1$ 时，A 与 A'具体的值不会引起最终输出的变化，即不会出现毛刺。修改后的电路图如图 3-23 所示。

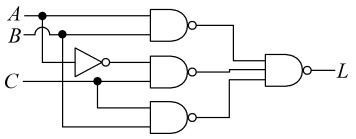


图 3-23　冒险消除电路

再看上面的第(2)小题例题，它的卡诺图与第(1)小题无异。由于它的冒险类型是静态 0 冒险，也就是会出现瞬时的 1 脉冲。要消除 ABC 从 000 变化为 100 时发生的毛刺，就需要对电路形式做一个修改：$L=(A'+B)(A+C)(B+C)$，从而避免在 A 发生变化的时刻出现瞬时的 1 输出。

【思考】 “输入的初始值和最终值被同一个本原蕴含项覆盖”是“电路不存在冒险”的必要条件吗？

再来观察一下上面的几个例子：

(1) $L=AB+A'C$　(静态 1 冒险)

(2) $L=(A'+B)(A+C)$　(静态 0 冒险)

(3) $L=AB+A'C+BC$　(无冒险)

(4) $L=(A'+B)(A+C)(B+C)$　(无冒险)

相信读者已经观察到，这四个表达式的功能/真值表是完全相同的，但是它们却存在完全不一样的冒险类型。这也说明，冒险是电路本身的固有属性，与实现的功能无关，而与实现的方式相关。虽然在逻辑表达式上可以做相互的转换，在功能上是等价的，但是最后的冒险类型可能完全不相同，这一点需要特别注意。

另一种消除冒险的方法是通过采样器对输出进行采样。毛刺的最大特点是产生的脉冲是瞬时的，主要发生在输入变化后信号还未稳定的短时间内。因此一个很直观的想法是即使无法避免毛刺的产生，也可以通过屏蔽毛刺来防止它对后续的电路工作产生影响。那么，要实现毛刺的屏蔽，就可以用一个电路器件来对电路进行隔离，然后利用一个采样脉冲在信号稳定后对输出进行采样，从而屏蔽毛刺带来的影响，原理图如图 3-24 所示。实际上，信号的周期采样在时序电路的设计中是十分必要的，而能够实现采样的电路器件一般为触发器，在时序逻辑章节中会做更详细的介绍。

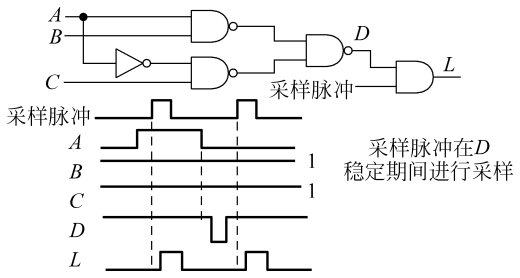


图 3-24　通过采样脉冲消除上述冒险

3.6　典型组合逻辑电路的设计

本节将介绍几种常见的组合逻辑电路，并对其功能、设计原理、设计方法等进行讲解，主要包括编码器、译码器、多路选择器、加法器等。在第 1 章中提到，进行 $y=f(x)$ 的计算时，计算机或其他硬件通常需要完成五项基本功能。

- 数据输入：提供接口接收输入信号 x。
- 数据存储：将计算需要的参数 w、输入信号 x 和数据处理过程中产生的中间数据 x' 等信息存储起来。
- 数据处理：完成计算功能 $f(\cdot)$。
- 数据输出：提供接口输出处理结果 y。
- 管理与控制：控制计算机系统工作，保障上述四项功能的正确运行。

对于数据的输入和输出，必然涉及信息从一种形式到另一种形式的转换，因此需要编解码器来完成数据的处理；对于数据处理，加法器是实现大部分计算的基础单元，因此本章也将对常见的加法器设计进行介绍；在计算过程中，也通常需要选择不同的计算通路来执行不同的运算，因此需要多路选择器来进行多种计算路径的选择。从计算机的构成来说，编码器、译码器、多路选择器、加法器等都是基础且不可缺失的模块，因此本章将逐个展开介绍。

在计算机体系架构中，这些基础模块也扮演着十分重要的角色。例如，译码器在内存寻址中将地址转换为具体的读使能信号，从而得到对应地址的数据；加法器是算术逻辑单元(ALU)中的重要模块，绝大多数计算都需要通过加法器完成；多路选择器能够依据不同的控制信号实现不同数据通路的选择，从而完成多种多样的计算。本节将介绍这些计算机基础模块的设计与优化方法，第 6、7 章将详细介绍这些基础模块在中央处理单元(CPU)中的具体应用。

3.6.1　编码器

编码，从广义上讲是指信息从一种形式或格式转换到另一种形式或格式的过程，从狭义上可以将编码理解为将图像、语音、信号转换为具有特定格式字符序列的过程。相对应地，译码(或解码)是编码的逆过程，即将转换后的格式解析回原格式。举个简单的例子，学校对每位同学都赋予了一个学号，这其实也是一个编码的过程，把每位同学用一个数字序列来“标记”，这样可以将一个“物理形式”上的人转换为一个“数字形式”上的序列来指代。当看

到一个学号时，想得知这个学号对应的是哪位同学，就需要一个译码过程。这里所讨论的编码器/译码器，具体来说，是针对一串数字序列转换为另外一种形式的数字序列的组合逻辑，一个典型的编码器框图如图 3-25 所示。

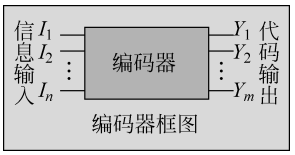


图 3-25　编码器框图

假设输出的编码是一个 m 位二进制数，那么最多可以对 $n \leqslant 2^m$ 种信号进行编码，通常命名为 2^n-n 线编码器。还是以学号为例，若学号位数为 10 位，则最多可以为 1024 位同学进行编号。下面以一个抢答器的设计为例进行分析。

【例 3-9】 设计一个抢答器，将 4 个抢答器的输出信号编码为二进制代码，抢答器按下，用 1 表示；抢答器未按下，用 0 表示。

解：图 3-26 是输入和输出关系的示意图。

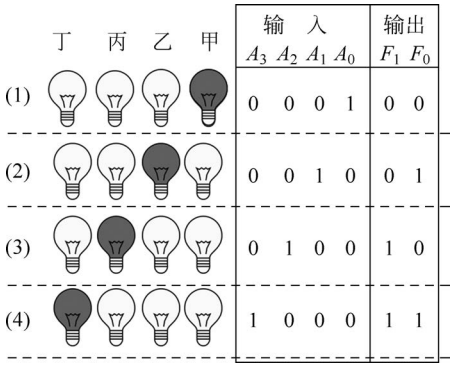


图 3-26　抢答器的输入和输出对应关系

根据输入输出的对应关系，其真值表如表 3-9 所示。

表 3-9　抢答器的真值表

A_3	A_2	A_1	A_0	F_1	F_0
×	×	×	1	0	0
×	×	1	0	0	1
×	1	0	0	1	0
1	0	0	0	1	1

将其他输入组合都作为无关项，则可以很容易得到输出的逻辑表达式为

$$F_0 = A_1 A'_0 + A'_2 A'_0, \quad F_1 = A'_1 A'_0$$

所以其组合逻辑电路为图 3-27 所示的电路。

但是相信读者也发现了一个问题，真值表中出现了很多无关项（‘×’）。在设计时认为“其他输入都作为无关项”，是认为现实情况下这些情况应当不会出现，然而这些无关项的情况实际上是有可能出现的，此时就要依据需求的定义进行设计的调整。例如，如果 4 个抢答器中有不止一个同时按下该怎么办呢？

若在 4 个抢答器中有不止一个同时按下，由于必然要决断出一个且仅有一个信号灯点亮，则会涉及优先级的问题，即通过一个优先级的定义，来保证多个抢答器同时按下时，仍有

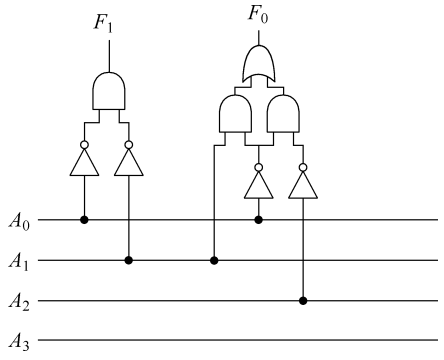


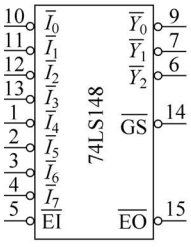
图 3-27　抢答器的组合逻辑电路

序地输出一个响应信号。例如，再看上面的例子，前述真值表实际上就是规定了 $A_0>A_1>A_2>A_3$ 的优先级顺序。当有多个抢答器同时按下时，就会根据 $A_0>A_1>A_2>A_3$ 的优先级来进行判定。若需要规定不一样的优先级顺序，则需要对真值表进行一些改变，例如规定 $A_1>A_0>A_2>A_3$，则真值表可以相应地变化为表 3-10。

表 3-10　优先抢答器的真值表

A_3	A_2	A_1	A_0	F_1	F_0
×	×	0	1	0	0
×	×	1	×	0	1
×	1	0	0	1	0
1	0	0	0	1	1

以下介绍一个常用的 8-3 线优先编码器芯片 74LS148。8-3 线的意思就是输入有 8 位信号，将其编码为 3 位的输出。以此类推，例 3-9 中的编码器就是一个 4-2 线编码器。74LS148 是带有扩展功能的 8-3 线优先编码器芯片，它具备 8 个输入信号端，3 个输出信号端，同时还有输入使能信号、输出使能信号和一个用于扩展的信号。74LS148 的逻辑符号和功能表如图 3-28 所示。



74LS148的逻辑符号

74LS148的功能表

输入									输出				
$\overline{EI}$	$\overline{I}_7$	$\overline{I}_6$	$\overline{I}_5$	$\overline{I}_4$	$\overline{I}_3$	$\overline{I}_2$	$\overline{I}_1$	$\overline{I}_0$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$	$\overline{GS}$	$\overline{EO}$
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	×	×	×	×	×	×	×	0	0	0	0	1
0	1	0	×	×	×	×	×	×	0	0	1	0	1
0	1	1	0	×	×	×	×	×	0	1	0	0	1
0	1	1	1	0	×	×	×	×	0	1	1	0	1
0	1	1	1	1	0	×	×	×	1	0	0	0	1
0	1	1	1	1	1	0	×	×	1	0	1	0	1
0	1	1	1	1	1	1	0	×	1	1	0	0	1
0	1	1	1	1	1	1	1	0	1	1	1	0	1

图 3-28　74LS148 的逻辑符号及功能表

利用 74LS148 芯片可以搭建一些简单的电路系统，例如搭建一个监视 8 个化学罐液面的报警编码电路，如图 3-29 所示。

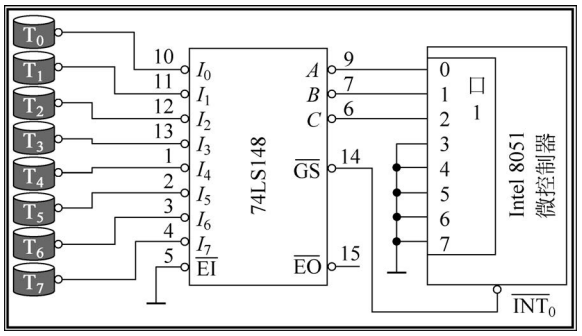


图 3-29　报警编码电路

可以看到，若 8 个化学罐中任何一个的液面超过预定高度，其液面检测传感器便输出一个 0 电平到编码器的输入端。通过编码器后，8 个化学罐的监测信号就可以转换为 3 位二进制码的输出到微控制器，从而微控制器仅需要 3 根输入线就可以监视 8 个独立的化学罐，本质上能够压缩信息的传输量。

3.6.2　译码器

译码（或解码）就是编码的逆过程，将一种用二进制码表示的信息转换成另一种形式。与编码器的命名类似，将 n 位输入解码为 2^n 种输出的译码器，通常命名为 n-2^n 线译码器。译码器可以近似理解为一个最小项发生器，比如输入是 001，输出为 $Y_7Y_6Y_5Y_4Y_3Y_2Y_1Y_0=00000010$，所以输出 Y_1 为 1，其实也就是输出了 001 对应的最小项。通常为了提高可扩展性，译码器会增加一个使能输入，以控制电路的工作状态，通过多个译码器的级联来处理更多比特的输入序列。图 3-30 是一个常见的 n-2^n 线译码器的逻辑框图。

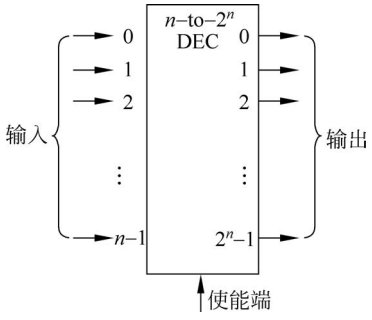
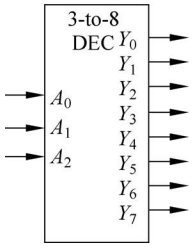


图 3-30　译码器框图

以一个 3-8 线译码器的设计为例，即将 3 位二进制代码的所有组合状态变换成 8 个输出的电路。若将输入的 3 位信号标记为 $A_2A_1A_0$，输出的 8 位信号标记为 $Y_7Y_6Y_5Y_4Y_3Y_2Y_1Y_0$。其模块示意图及功能真值表如图 3-31 所示。



A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
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

图 3-31　3-8 线译码器模块示意图与真值表

根据真值表，可以写出布尔表达式并画出相应的电路设计图如图 3-32 所示。

74HC138 是一个常用的 3-8 线译码器芯片，除了规定的三个编码输入信号外，还有三个使能信号，可以实现更多功能的扩展。

通过图 3-33 中的电路可以观察到，对于三个使能信号，当 $E_3=1$，$E_2'+E_1'=0$ 时，该芯片实现正常的 3-8 线译码器功能；当 $E_3=0$ 或 $E_2'+E_1'=1$ 时，不译码，输出都为高电平。分析输出与输入的关系，例如 $Y_0'=(A_2'A_1'A_0')'=m_0'$，$Y_1'=(A_2'A_1'A_0)'=m_1'$，…，等同于一个最小项的反的发生器，即最大项发生器。详细功能表如表 3-11 所示。其中，当使能信号不为 100 时，输出将始终为高电平。

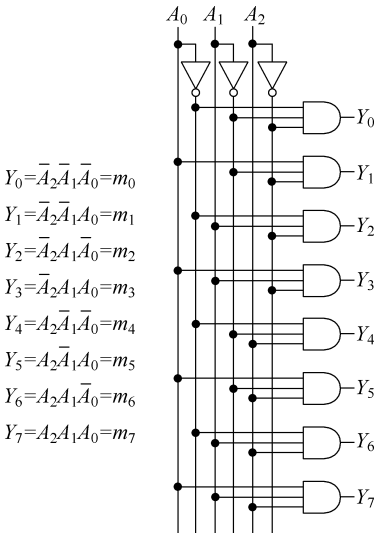


图 3-32　3-8 线译码器的电路图

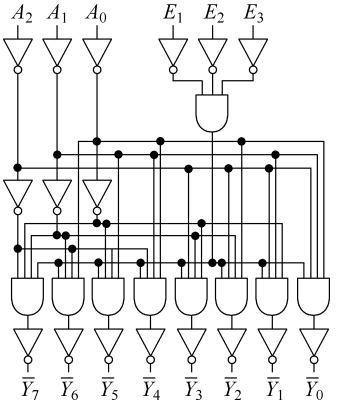


图 3-33　74HC138 的电路图

表 3-11　74HC138 的功能表

$E_3E_2'E_1'$	$A_2A_1A_0$	Y_7'	Y_6'	Y_5'	Y_4'	Y_3'	Y_2'	Y_1'	Y_0'
100	000	1	1	1	1	1	1	1	0
	001	1	1	1	1	1	1	0	1
	010	1	1	1	1	1	0	1	1
	011	1	1	1	1	0	1	1	1
	100	1	1	1	0	1	1	1	1
	101	1	1	0	1	1	1	1	1
	110	1	0	1	1	1	1	1	1
	111	0	1	1	1	1	1	1	1
0××，×1×××1	×××	1	1	1	1	1	1	1	1

有趣的是，译码器不仅可以用来实现译码功能，还可以用来实现多输出的组合逻辑函数。比如要实现一个组合逻辑，可以将其转换为最大项的与非形式，从而利用 74HC138 的“最大项发生器”的特性来实现这样的组合函数。例如，下列逻辑表达式可以转换为一系列最大项的与非形式，再在输出端口利用与非门连接即可。

$$Z=A\overline{C}+\overline{A}BC+A\overline{B}C=AB\overline{C}+A\overline{B}\overline{C}+\overline{A}BC+A\overline{B}C$$
$$=m_3+m_4+m_5+m_6=\overline{\overline{m}_3\cdot\overline{m}_4\cdot\overline{m}_5\cdot\overline{m}_6}$$

图 3-34 展示了其电路示意图。

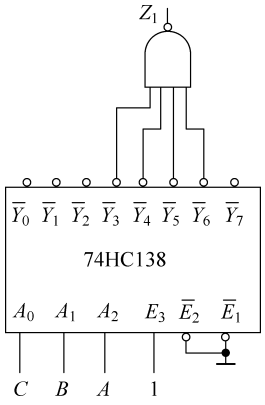


图 3-34 使用 74HC138 实现组合函数

另一方面，如果想实现更多比特输入的译码，也可以利用已有的 3-8 线 74HC138 译码器芯片来构造这样的一个扩展芯片。例如，构造一个 4-16 线的译码功能。

首先，将真值表列出如表 3-12 所示。

表 3-12 译码功能真值表

$D_3D_2D_1D_0$	Y'_{17}	Y'_{16}	Y'_{15}	Y'_{14}	Y'_{13}	Y'_{12}	Y'_{11}	Y'_{10}	Y'_{27}	Y'_{26}	Y'_{25}	Y'_{24}	Y'_{23}	Y'_{22}	Y'_{21}	Y'_{20}
0000								0								
0001							0									
0010						0										
0011					0											
0100				0												
0101			0													
0110		0														
0111	0															
1000																0
1001															0	
1010														0		
1011													0			
1100												0				
1101											0					
1110										0						
1111									0							

其次，可以通过 D_3 这个输入信号来控制 74HC138 的使能端，这样可以通过两个 3-8 线译码器来分别输出其中的 8 个最大项，具体电路框图如图 3-35 所示。

当且仅当使能端输入为 $E_3E_2E_1=100$ 时该编码器的输出有效，所以当 D_3 为 0 时，左边译码器输出有效；当 D_3 为 1 时，右边译码器输出有效。

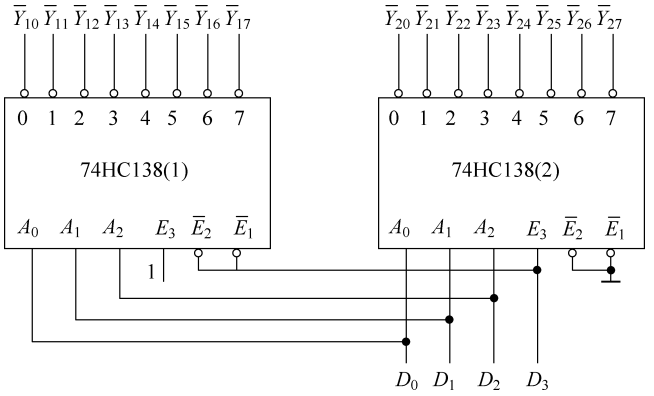


图 3-35　4-16 译码电路图

译码器的一个典型且重要的应用是处理器中的寻址模块，通常需要译码器来完成存储单元的寻址。将指令中的数据地址输入到译码器后拉起对应的存储单元，并将对应的数据读取或写入。一个典型的结构图如图 3-36 所示，在处理器章节会对图 3-36 进行更详细的介绍。

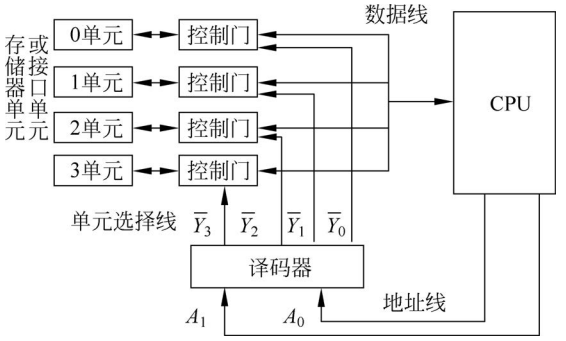


图 3-36　使用译码器进行存储单元读写控制

3.6.3　多路选择器

多路选择器，顾名思义，就是从多个输入数据中选择一个送往唯一通道输出，也称为数据选择器或多路开关，通常简写为 MUX。若一个多路选择器从 m 个输入数据中选择 1 个，通常记为 m∶1 多路选择器。若需要从 2^n 个数据输入中选择一个输出，则需要 n 位的选择信号。多路选择器的一个常见框图如图 3-37 所示。

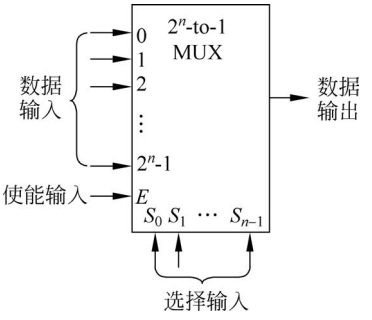


图 3-37　多路选择器框图

【例 3-10】 请设计一个 8 选 1 多路选择器，其中 8 个数据输入记为 $D_7D_6D_5D_4D_3D_2D_1D_0$，3 个选择信号记为 $A_2A_1A_0$，输出数据记为 Q；要求实现的功能是根据输入的取值，将 8 个数据输入中的一个送到 Q。

解：根据功能描述，可以列出真值表如表 3-13 所示。

表 3-13　8 选 1 多路选择器的真值表

A_2	A_1	A_0	Q	A_2	A_1	A_0	Q
0	0	0	D_0	1	0	0	D_4
0	0	1	D_1	1	0	1	D_5
0	1	0	D_2	1	1	0	D_6
0	1	1	D_3	1	1	1	D_7

根据真值表，可以得到输出与输入、选择信号的逻辑表达式：

$$Q=\overline{A}_2\overline{A}_1\overline{A}_0D_0+\overline{A}_2\overline{A}_1A_0D_1+\overline{A}_2A_1\overline{A}_0D_2+\overline{A}_2A_1A_0D_3+$$
$$A_2\overline{A}_1\overline{A}_0D_4+A_2\overline{A}_1A_0D_5+A_2A_1\overline{A}_0D_6+A_2A_1A_0D_7$$

映射到对应的逻辑门后，可以设计电路如图 3-38 所示。

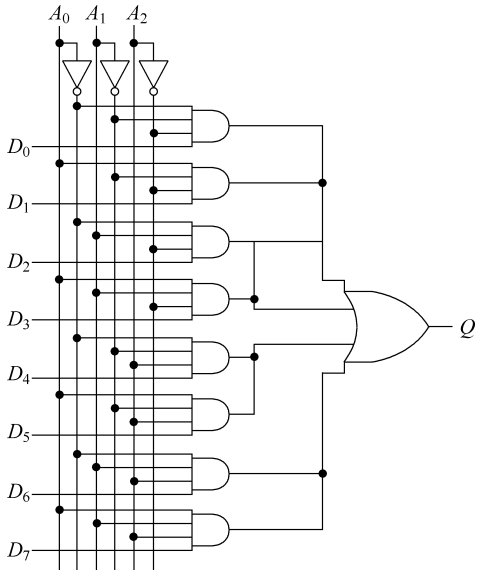


图 3-38　例 3-10 的功能电路

以上便完成了 8 选 1 多路选择器的设计。

多路选择器在实际应用中也很常见。例如利用 2∶1 多路选择器来构造移位器。图 3-39 就是一个构造成的 8 位右移位器。通过选择输入信号来控制移动的位数。比如，要右移 4 位，则将选择信号设置为 $S_2S_1S_0=100$，两位则为 010。注意，最高位需要补充输入，若有符号位，则需补符号位；若无，则补零。同样可以观察到，对于一个 2^n 位的移位器，需要构建 n 级，比如实现一个 32 位的移位器，则需要 5 级。

多路选择器也在目前的 CPU 中应用广泛，图 3-40 是一个 CPU 的示意图（第 6、7 章将对此部分内容详细介绍），可以看到，CPU 中数据通路的控制就是通过多路选择器实现的（图 3-40 的 MUX 模块），指令译码的过程实际上就是产生各个多路选择器的控制信号的过程。

多路选择器还可以用于实现任意的逻辑函数，本质上多路选择就是对应通过地址来查

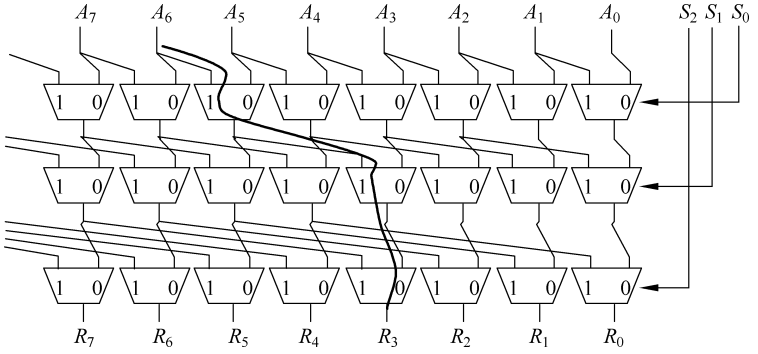


图 3-39　使用多路选择器构建移位器

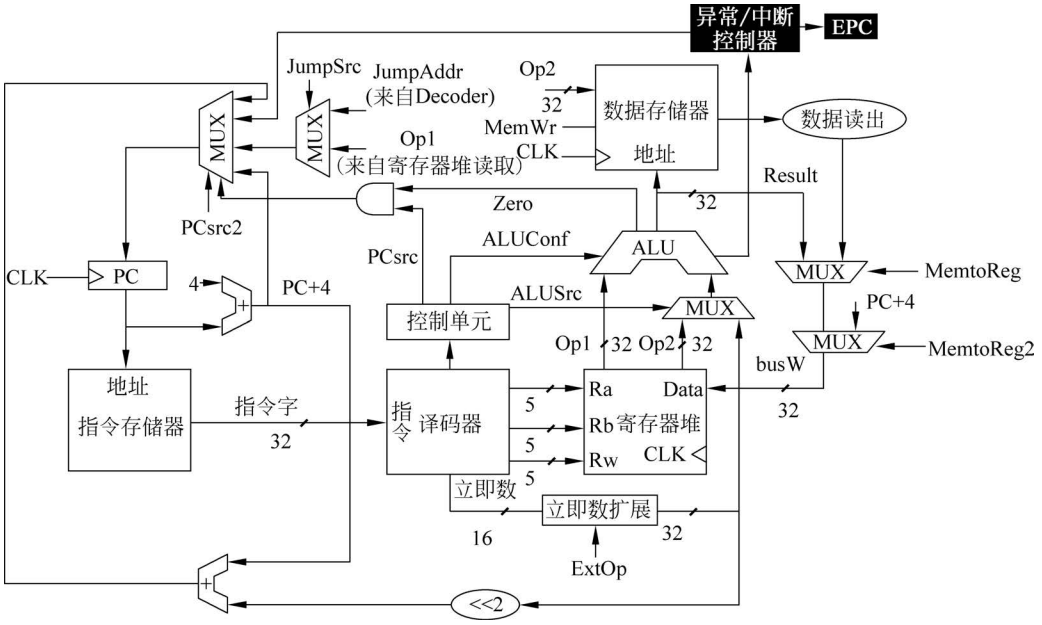


图 3-40　MIPS CPU 示意图

表的过程。若用具有 n 个地址端的多路选择器来实现 n 变量的逻辑函数，则直接将函数的输入变量加到多路选择器的选择信号上，并将多路选择器的数据输入端按照次序以函数 F 的输出值来赋值，可以看到这其实就是一个查表的过程，且实现逻辑函数不需要任何附加的逻辑门。比如，图 3-41 中使用 8 选 1 MUX 实现 3 变量逻辑函数。

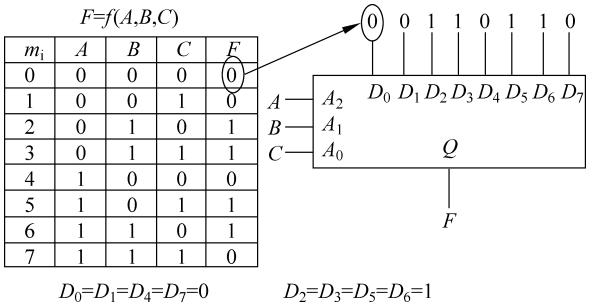


图 3-41　8 选 1 MUX 实现 3 变量逻辑函数

将输入送到选择信号端，将 F 的函数值依次赋值给输入端，即可实现如图 3-41 真值表所对应的逻辑函数。若用具有 n 个地址端的多路选择来实现 $m(m>n)$ 变量的逻辑函数，则应将函数的其中 n 个输入变量加到地址端，并将多路选择器的数据输入端按次序以函数 F 与另外 $m-n$ 个输入变量的逻辑关系来赋值，当 $m=n+1$ 时，不需要任何除非门外的附加逻辑门。例如图 3-42 中使用 8 选 1 MUX 实现 4 变量逻辑函数。

A	B	C	D	L
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

$L=f(A,B,C,D)=ABC+BC'D'+AB'C'+AB'D+B'C'D$

AB \ CD	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	1	0	1	1
10	1	1	1	0

图 3-42　8 选 1 MUX 实现 4 变量逻辑函数

通过将 A、B、C 信号作为地址端，然后将 L 和 D 的逻辑关系对输入端进行赋值，对真值表做一个转换并设计成图 3-43 中的电路。

ABC	D	L
000	0	D
	1	
001	0	0
	1	
010	0	D'
	1	
011	0	0
	1	
100	0	1
	1	
101	0	D
	1	
110	0	D'
	1	
111	0	1
	1	

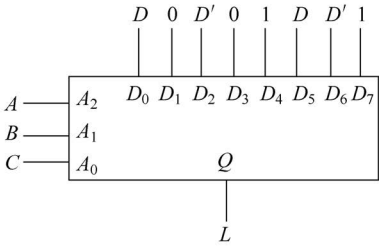


图 3-43　4 变量逻辑函数实现电路

使用 8 选 1 MUX 实现 4 变量逻辑函数的方式并不唯一。例如，可以将 B、C、D 作为地址端，将 L 和 A 的逻辑关系对输入进行赋值。不同的实现方式会涉及代价权衡的问题，因使用的逻辑门数量不同，从而使整体的代价不同。该问题在变量数目较多的情况下更为显著。

3.6.4　加法器

加法运算是计算机中的一种基础性运算，加、减、乘、除四则运算均可以通过某些方式转换为加法运算。加法器 Adder 主要分为两类，一种是半加器，不考虑来自低位的进位，也就

是只有 2 个本位数相加；另一种是全加器，考虑来自低位的进位，也就是对 2 个本位数和 1 位来自低位的进位进行相加。以下分别介绍这两种类型的加法器。

1. 半加器

根据功能定义，半加器(half adder)的设计依然遵循组合逻辑电路设计的基本步骤。首先进行逻辑抽象，输入为待相加的两个本位数，即加数 A 和被加数 B，输出为两个数的和 S。从而可以根据加法运算的规则列出真值表如表 3-14 所示。

表 3-14　半加器的真值表

A	*B*	*S*	*A*	*B*	*S*
0	0	0	1	0	1
0	1	1	1	1	0

根据真值表，就可以得到 S 和 A、B 的逻辑表达式：

$$S = A'B + AB' = A \oplus B$$

于是可以设计出电路图如图 3-44 所示。

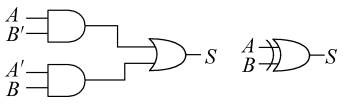


图 3-44　半加器的逻辑电路

2. 全加器

与半加器不同的是，全加器(full adder)需要考虑来自低位的进位，完成 2 个本位数＋1 位低位进位＝3 位数的相加，并输出一个和以及向高位的进位。首先进行逻辑抽象，输入为本位加数 A_i、本位被加数 B_i、来自低位的进位(Carry)C_i，输出为和(Sum)S_i 和向高位的进位 C_{i+1}。从而根据全加器的运算规则，列出真值表如表 3-15 所示。

表 3-15　全加器的真值表

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

根据真值表，可以写出和与进位的逻辑表达式：

$$S_i = A'_iB'_iC_i + A'_iB_iC'_i + A_iB'_iC'_i + A_iB_iC_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = B_iC_i + A_iB_i + A_iC_i$$

从而设计电路图如图 3-45 所示。

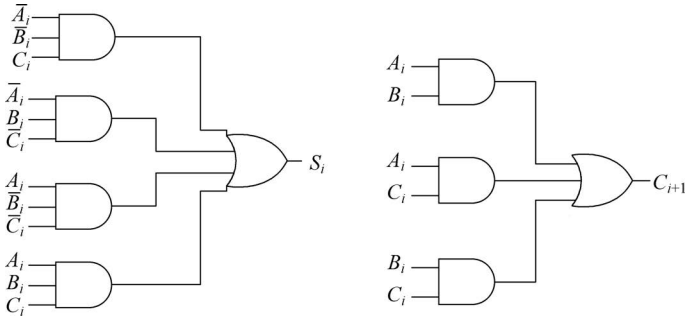


图 3-45　全加器的逻辑电路

对上述电路的一个化简电路形式，其信号延时分析如图 3-46 所示。

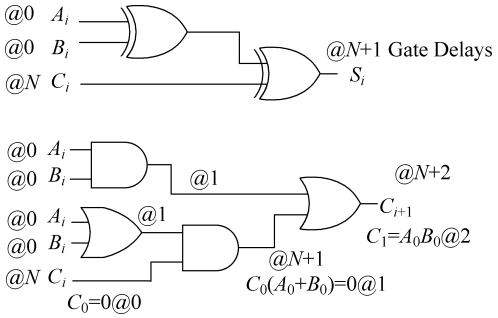


图 3-46　全加器信号延时分析

假设所有门的延时都为 1，对于一个全加器而言，若加数信号为零时刻，进位输入 C_i 到达的时刻为 N 时刻（假设 $N>1$），则根据逻辑电路图可以看到，得到该级进位输出的时刻为 $N+2$ 时刻，和位输出为 $N+1$ 时刻（因为所有 A 与 B 的运算都将在 1 时刻完成）。

1 位全加器又称为（3，2）adder，分别指代三个输入和两个输出，其符号表示一般画成图 3-47 所展示的形式。

3. 多位加法器

实际应用中，通常遇到的都是多位数相加的问题，所以需要构建多位加法器。然而，如果通过直接逻辑映射的方法去设计多位加法器，其真值表空间随位数指数级增长，所以需要对问题进行分解，通过多个 1 位全加器来构成多位的加法器。多位加法器实现的功能如图 3-48 所示。

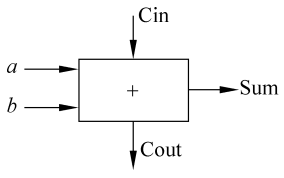


图 3-47　全加器符号

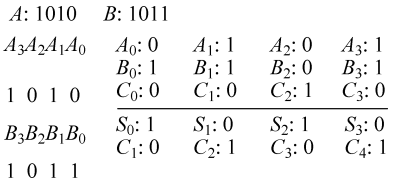


图 3-48　多位加法器功能

构建多位加法器最直观的方法就是通过多个一位全加器的级联来实现，这种构建模式是串行加法器。串行加法器的优点是构建方式简单、电路简单、使用器件少，缺点是位间进位是串行进行的，必须等低位进位计算完成后才能进行当前比特位的加法计算，也就是 S_i 和 C_{i+1} 的运算必须等 C_i 来到后才能进行，所以速度与位的数量正相关。一般认为延时是线性增长的。图 3-49 展示了一个典型的串行加法器结构（也称行波进位加法器）。

对于单个全加器的延时已经分析过，这里以 4 位加法器作为示例对串行加法器的延时进行分析。如图 3-50 所示，已知从输入 C_i 到输出 C_{i+1} 的延时为 2 个门延时（依

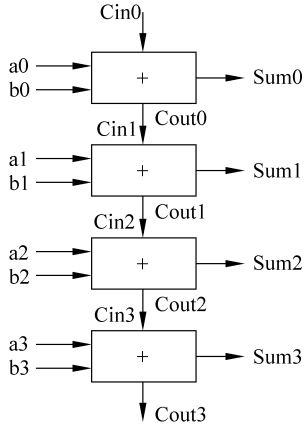


图 3-49　串行加法器结构

据电路设计)、到输出 S_i 为 1 个门延时；且假设从输入 A_i 到输出 C_{i+1} 的延时为 2 个门延时、到输出 S_i 的延时为 2 个门延时，以 A_0、B_0 的输入时刻为零时刻，则各个信号产生的时间如图 3-50 所示。可以看到最终实现完整的运算需要 8 个门延时。注意，各级的加数和被加数都是同时输入，且其中一部分门的运算都可以提前运算。

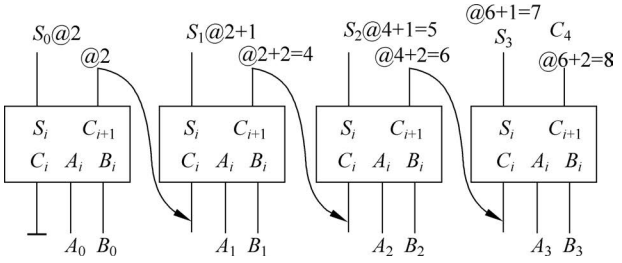


图 3-50　串行加法器延时分析

行波进位加法器虽然简单，但是每一位的计算都要在上一位进位输出计算完毕后才可以开始有效计算，所以延时会随加法器的级数线性增长。可以看到，不同级之间的依赖关系主要是级间传递的进位信号，所以若能把进位的依赖关系剥离，则不同级就可以同时进行运算，而不需要依赖于来自其他级的输出作为输入来源，显然能够大幅降低整体计算需要的延时。

这样的设计其实是一种并行加法器的设计思想，也称超前进位加法器[①]。它的基本计算流程是：①输入各比特位的加数 A_i 和被加数 B_i；②计算各比特位的进位输出 C_{i+1}；③计算各比特位的和输出 S_i。超前进位加法器的设计思想是递推的思路。首先，列出进位输出的表达式如下所示：

$$C_1 = A_0C_0 + B_0C_0 + A_0B_0$$
$$C_2 = A_1C_1 + B_1C_1 + A_1B_1$$
$$\cdots$$
$$C_{i+1} = A_iC_i + B_iC_i + A_iB_i$$

将 C_i 的表达式逐级代入展开，将就可以得到

$$C_2 = A_1(A_0C_0 + B_0C_0 + A_0B_0) + B_1(A_0C_0 + B_0C_0 + A_0B_0) + A_1B_1$$

从而可以摆脱不同级之间的依赖关系，达到超前进位计算的目的。但是也可以观察到，这样逐级展开到更多级的情况，逻辑表达式会变得异常复杂，那么如何让这个递推关系式写得更清晰明了一些呢？

重新观察进位信号的表达式：

$$C_{i+1} = A_iC_i + B_iC_i + A_iB_i$$
$$C_{i+1} = A_iB'_iC_i + A'_iB_iC_i + A_iB_iC'_i + A_iB_iC_i$$
$$= (A'_iB_i + A_iB'_i)C_i + A_iB_i(C'_i + C_i)$$
$$= P_iC_i + G_i$$

其中，$P_i = A_i \oplus B_i$，$G_i = A_iB_i$，根据等式的特性，称 P_i 为进位传播(propagation)信号，G_i 为进位产生(generation)信号。回顾求和 S_i 的逻辑表达式，也可以用 P_i 表示：

① Rosenberger G B. Simultaneous Carry Adder: U. S., 2966305[P], 1960.

$$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i$$

图 3-51 分析了其中一个比特位的超前进位加法器的延时。可以看到，在输入信号 A_i、B_i 到达时刻后的 1 个门延时即可得到进位传播信号 P_i 和进位产生信号 G_i，而 S_i 也可以在获得 A_i、B_i、C_i 信号后的 2 个门延时得到。

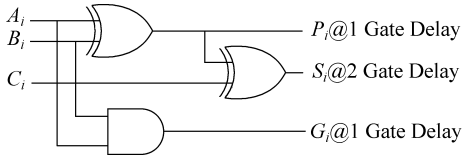


图 3-51　超前进位加法器延时分析

为了更快地得到 S_i 的计算结果，可以基于这个等式利用进位传播信号 P_i 和进位产生信号 G_i 两个信号来完成 C_i 的递推：

$$C_{i+1} = P_i C_i + G_i = P_i(P_{i-1}C_{i-1} + G_{i-1}) + G_i = P_i P_{i-1} C_{i-1} + P_i G_{i-1} + G_i$$
$$= \cdots = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \cdots + P_i P_{i-1} P_{i-2} \cdots P_1 G_0 + P_i P_{i-1} P_{i-2} \cdots P_0 C_0$$

以下是一个 4 位进位的逻辑表达示例：

$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$
$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

实现超前进位的目的是减少延时，因此对这样的逻辑实现方式进行延时的分析。以一个 4 位的加法逻辑为例，分析从输入到最后一级进位输出所经过的延时。如图 3-52 所示，其中红色标注为信号延时，未标注信号认为在零时刻到达。首先，进位信号 C_1、C_2、C_3、C_4 由 P_i、G_i 两级逻辑产生，相比 P_i、G_i 有两级门延时；其次，由上图可知，P_i、G_i 由 A_i、B_i 经过一级门延时后产生；因此，进位信号在三级门延时后产生。进一步，对于和位输出，S_1、S_2、S_3 在 C_1、C_2、C_3 产生一个门延时之后产生，而 S_0 在 P_0 产生一个门延时后产生。综上所述，S_0 在输入经过两个门延时后产生，S_1、S_2、S_3 在输入经过四个门延时产生。

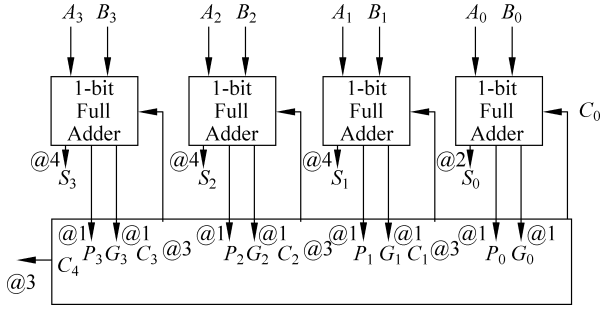


图 3-52　4 位超前进位加法器

但是依然可以发现，当加数与被加数的位数增多时，需要考虑一些实际电路的约束。随着级数增加，与门和或门的输入端数目也在线性增加，若依然使用两级逻辑，此时单个逻辑

门的输入端数目迅速增多，一方面电路的输入端数目受限制，另一方面扇入增加时速度也会变慢；由于 P、G 都被多次调用，随着比特数增多，P、G 逻辑门的扇出也会增大，增加延时。所以为了缓解这样的现象，使加法器设计的可扩展性更好，可以引入层次化的设计思想，即组内并行、组间串行的进位加法器。如图 3-53 所示，对于一个 16 位的加法器，可以分为四组，其中各组都是一个 4 位的超前进位加法器，不同组之间采用串行进位的方式进行连接。

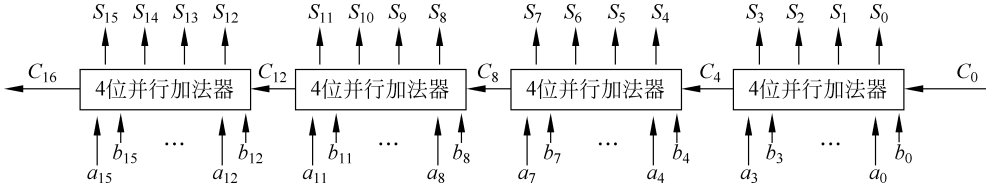


图 3-53　组内并行组间串行加法器

【思考】 图 3-54 是一个组内并行、组间串行的 8 位超前进位加法器，请问 S_7 和 C_8 会在输入完成后几级门延时产生？（假设 A_i、B_i、C_0 同时在 0 时刻输入）

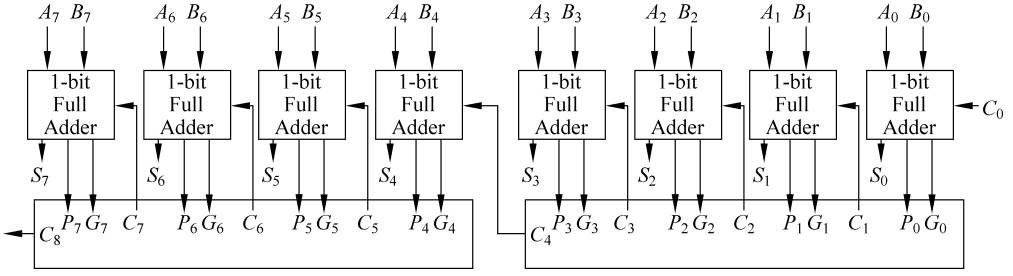


图 3-54　组内并行组间串行加法器

3.7　总结

阅读完本章，相信读者对组合逻辑的基础知识和设计方法都有了比较清晰的认识，下面总结一下本章的重要内容：

本章首先介绍组合逻辑电路的定义与表示方法，请读者牢记组合逻辑当前时刻的输出仅取决于当前时刻的输入，与历史输入或状态无关。然后介绍组合逻辑的分析与设计方法。其中，分析是指从电路推导出逻辑表达式，并建立真值表或卡诺图，再分析其功能；设计方面介绍一个通用的组合逻辑设计流程，包括逻辑抽象、列真值表、逻辑化简和电路结构映射等。然后介绍组合逻辑电路的评价方法，一般分为稳态因素和动态因素。稳态因素包括面积、扇入扇出系数、噪声容限等，动态因素包括延时、功耗等。这些因素共同构成评价组合电路优劣的参考性指标，能够帮助设计者更好地根据应用的约束和偏好调整电路设计。此外介绍组合逻辑电路的竞争与冒险现象，包括其产生原因以及如何消除组合电路中的冒险现象。最后以编码器、译码器、多路选择器和加法器为例，介绍典型组合逻辑电路的设计与实现方法，第 6、7、9 章将进一步介绍在 CPU 与存储器中这些基础模块的功能应用。

通过本章，希望读者能够掌握基本的数字逻辑设计思想，并能够延伸扩展到实际的应用算法问题中，思考从真实世界到数字世界的建模和实现过程。

3.8 拓展阅读

[逻辑化简] Quine W V O. The Problem of Simplifying Truth Functions[J]. The American Mathematical Monthly,1952,59(8): 521-531.

[逻辑综合] Damiani M. Synthesis and optimization of synchronous logic circuits[D]. Stanford University,1994.

[逻辑综合]Iman S, Pedram M. Logic synthesis for low power VLSI designs[M]. Springer Science & Business Media,1998.

3.9 习题

1. 根据下列布尔函数的定义,写出化简过程及最简两级与或表达式;利用或非门(输入端口数不限)和非门实现电路;判断电路是否存在冒险,若存在,实现其无冒险电路。

(1) $f(A,B,C,D)=\sum m(0,1,3,4,9,11)+\sum d(7,15)$

(2) $f(A,B,C,D)=\sum m(0,1,4,5,7,13,15)+\sum d(2,3)$

(3) $f(A,B,C,D)=\sum m(1,3,5,7,9)+\sum d(4,11)$

2. 考虑逻辑门的延时,将奇数个反相器首尾相连时,可以产生一定周期的振荡信号。假设反相器的延时为 t_d,如何实现周期为 $6t_d$ 的振荡时钟信号?可以实现周期为 $8t_d$ 的振荡时钟信号吗?

3. 用一个 4∶1 多路选择器和其他任意逻辑单元(但附加的逻辑要尽可能最少)实现函数:$f(A,B,C,D,E)=A+C'D+BD'+B'D+B'CE$,应当使用哪两个信号来控制多路选择器?利用由下列输入信号组合控制的多路选择器实现上述函数。哪一种结果需要更少的逻辑?为什么?

(1) 将 A 和 B 作为 4∶1 多路选择器的控制输入。

(2) 将 B 和 C 作为 4∶1 多路选择器的控制输入。

(3) 将 B 和 D 作为 4∶1 多路选择器的控制输入。

(4) 将 C 和 D 作为 4∶1 多路选择器的控制输入。

4. 设计一种优先编码器如图 3-55 所示,$I_3I_2I_1I_0$ 为 4 位有权输入,O_1O_0 为编码输出,该编码器将同时输出结果有效位 V,$V=1$ 表示输出 O_1O_0 有效,输出 O_1O_0 为输入 $I_3I_2I_1I_0$ 中权重最大的 1 出现的位置(例如,输入 1111 时输出 11,输入 0001 时输出 00);$V=0$ 表示输出 O_1O_0 无效,此时输入 $I_3I_2I_1I_0$ 均为 0。请画出该编码器的卡诺图,并使用与非门和非门来实现该电路功能。

图 3-55 优先编码器

5. 考虑两位十进制数加法器设计:对两位十进制数进行 8421BCD 编码(仅需要考虑非负数),请使用如下电路模块设计一个基于 8421BCD 码的加法器,输入输出形式为

$$(A_{13}A_{12}A_{11}A_{10}A_{03}A_{02}A_{01}A_{00})_{BCD}+(B_{13}B_{12}B_{11}B_{10}B_{03}B_{02}B_{01}B_{00})_{BCD}$$
$$=(C_{20}C_{13}C_{12}C_{11}C_{10}C_{03}C_{02}C_{01}C_{00})_{BCD}$$

例如：$50_{10}+50_{10}=(01010000)_{BCD}+(01010000)_{BCD}=(10000,0000)_{BCD}$

可供选择的电路模块为：4 位加法器(A、B 为 4 位输入，S 为 4 位结果输出，C_{in}、C_{out} 分别为进位输入、输出)，二选一多路选择器(A、B 为两输入，Sel 为选择信号，OUT 为输出)，具体如图 3-56 可供选择电路所示。(提示：注意考虑进位问题，如有需要，请思考如何利用多路选择器实现与、或运算。)

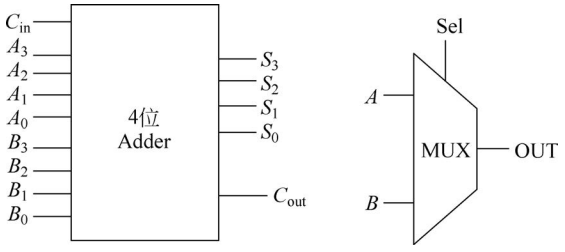


图 3-56　可供选择电路

6. 请设计一个特殊的编码器，将一个 4 位的 one-hot 向量按照格雷码顺序编码，具体为：将 0001 编码为 00，0010 编码为 01，0100 编码为 11，1000 编码为 10，并仅使用与非门和非门来实现电路设计。

7. 请设计一个特殊的解码器，将一个 2 位的二进制码解码为 4 位的 one-hot 向量，具体为：将 00 解码为 0001，将 01 解码为 0010，将 10 解码为 0100，将 11 解码为 1000。

8. 深度学习在近年来大幅推动了人工智能领域的发展，有一类神经网络称为二值神经网络，其中有一种方法称为 XNOR-Net。在 XNOR-Net 中，卷积层或全连接层的权重、输入和输出都是二值的形式，即仅能取 1 或−1。因此，权重和输入的乘法遵循以下规则：

输入	权重	输出	输入	权重	输出
1	1	1	−1	1	−1
1	−1	−1	−1	−1	1

如果把 1 编码为 1，−1 编码为 0，可以看到等价于一个 XNOR 运算。现在考虑一个具有 5 维输入、5 维权重、1 维输出的全连接计算，全连接计算需要将所有的输出加和后再进行一次判断，取最终的符号位作为计算结果，即 $y=\mathrm{sign}\left(\sum_{i=0}^{4} w_i x_i\right)$，即当内部累加和大于 0 时，输出为 1；当内部累加和小于 0 时，输出为−1。请设计相应的组合逻辑电路来实现这一运算，并尽可能使电路简单高效。

9. 使用多个带有使能端的 2∶4 译码器和逻辑门电路实现一个 4∶16 译码器。

10. 请类比加法器，完成下述题目：

(1) 一位全减器与一位全加器类似，其包含两个数据输入(1 位被减数 A 与减数 B)，一个低位的借位输入(BL_{in})，一个高位的借位请求输出(BL_{out})和一个 1 位差结果输出(D)。请列出一位全减器的真值表，并给出具体的电路实现。

(2) 类比超前进位加法器，考虑 3 位超前借位减法器，输入信号为 $A_2A_1A_0$ 与 $B_2B_1B_0$，完成 $A-B$ 的运算后输出差 $D_2D_1D_0$，及借位输出信号 BL_{out}，请写出各级借位传

播信号、借位产生信号的布尔表达式，并写出各级计算差输出与借位输出的布尔表达式。

(3) 根据上面的分析结果，设计一个**模式可调、超前进/借位的3位加减法器**。要求：输入两个3位有符号 A 与 B(1位符号位+2位数据位，负数为2补码形式表示)，当模式控制信号 $M=0$ 时，该超前进/借位加减法器完成 $A+B$ 的计算，当模式控制信号 $M=1$ 时，该超前进/借位加减法器完成 $A-B$ 的计算，并在设计中添加一个1位溢出指示位 Overflow，Overflow=1 时表示运算有溢出，Overflow=0 时计算无溢出。可供使用的电路模块包括与门、或门、非门、异或门、2选1多路选择器，请使用**尽可能少的电路模块**完成加减法器设计。

11. 设计一个2位有符号数(2-补码)乘法器，并与2位加法器进行比较，定性分析二者之间的能耗和面积差距。假设：仅使用与门、或门、非门，且认为所有的门具有相同的代价(无论扇入或扇出是多少)。

12. 图3-57是一位加法器的实现电路图，其中与门的延时为 a，或门的延时为 b，异或门的延时为 c，现在用这样的一位加法器构成 N 位行波进位(串行)加法器，如果最低位加法器的进位输入与各位加数同时到达，求完成一次 N 位加法运算时，产生最终所有“和”(S_i，$i=0,1,2,\cdots,N-1$)位所需的总时间以及产生最终进位输出(Cout_N)所需要的时间。

13. ALU(算术逻辑单元)是指能实现多组算术运算和逻辑运算的组合逻辑电路，是CPU中的核心执行单元。考虑一个简单的ALU，包含2个运算选择输入 S_1 和 S_0，对于两个输入数据 A 和 B(均为2位有符号数)，可以实现以下功能：

$S_1S_0=00$，实现 $F=A+B$

$S_1S_0=01$，实现 $F=A-B$

$S_1S_0=10$，实现 $F=A$ OR B

$S_1S_0=11$，实现 $F=A$ AND B

14. 实现一个“周末”判别器，其功能为：输入一个初始的时间设定，例如，该月的第1天为周三，并输入一个日期数，判定这一天是否是周末(周六或周日)，例如，若输入5，即该月份的第五天是一个周日，即是一个周末，此时输出1；若输入7，则该天为周二，不是周末，此时输出0。设计组合逻辑电路实现上述功能，注意一个月的最大天数为31天。

15. 查找表(LUT)是组合逻辑的一种经典实现方式，通过将各种情况下的计算结果存到一个随机访问存储(RAM)阵列中，根据输入直接取出存储阵列中对应的结果输出，这样在计算一些复杂运算时可以大幅提高运算速度。比如，要实现一个 A 和 B 的与运算，可以将 AB 作为地址索引信号，并存储到 4×1 RAM，可以利用图3-58中的查找表完成。

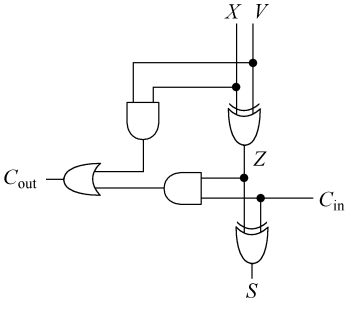


图3-57 加法器实现电路

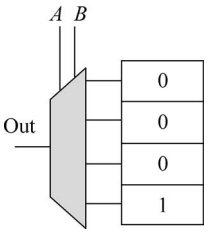


图3-58 查找表示意图

现考虑逻辑函数 $F(A,B,C,D)=AB'+BCD$，输入为 A、B、C、D 四根地址线，按照二进制码顺序可以将依次取出对应行的存储数值，请问图 3-59 中的 16×1 RAM 中的存储数值该如何排列？

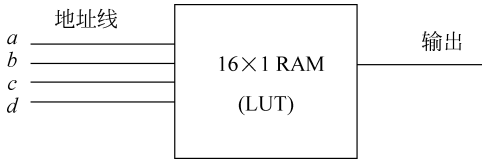


图 3-59　待实现功能电路框图

16. 除法器的设计：

设计一个 2 位除法器，输入为两个 2 位二进制数，输出为 A 除 B 的结果和余数。