数字逻辑电路设计与实现

李宜达　编著

科学出版社
www.sciencep.com

数字逻辑电路设计与实现

李宜达　编著

科学出版社

北京

图字：01-2003-7270 号

内 容 简 介

本书是讲述用 AHDL 和 VHDL 进行逻辑电路设计的书籍。全书共分为 9 章，第 1 章介绍了一些基本概念；第 2 章介绍了 MAX+plus II 软件所提供的设计输入法；第 3 章介绍了 HDL 语法；第 4 章~第 8 章介绍了组合逻辑电路、触发器、计数器和寄存器的内容；第 9 章介绍了时序逻辑电路分析与设计的过程。

本书适合电子类专业的学生使用，也可作为相关从业人员的参考书。

图书在版编目(CIP)数据

数字逻辑电路设计与实现/李宜达编著. —北京：科学出版社，2004

ISBN 7-03-013223-8

Ⅰ. 数… Ⅱ. 李 … Ⅲ. 数字电路：逻辑电路－电路设计

Ⅳ. TN 790.2

中国版本图书馆 CIP 数据核字(2004)第 026813 号

策划编辑：吕建忠/责任编辑：丁 波

责任印制：吕春珉/封面设计：北新华文

科学出版社 出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2004 年 5 月第 一 版 开本：787×1092 1/16

2004 年 5 月第一次印刷 印张：19 3/4

印数：1—4000 字数：452 000

定价：28.00 元

（如有印装质量问题，我社负责调换〈环伟〉）

前　言

近几十年来，数字电路的设计发展非常迅速，在设计方法上也有一些变革。早期设计数字电路时大多使用 TTL 逻辑器件，后来由于数字电路的功能越来越复杂，设计所需的时间越来越多，TTL 逻辑器件已经不能满足需要。近年来，数字电路的设计方式已发展到硬件描述语言（HDL）。

本书共分为 9 章，第 1 章是逻辑概论，主要介绍了数字系统、二进制数的四则运算、BCD 码、格雷码、基本逻辑组件、布尔代数、布尔函数的化简等内容；第 2 章为绘图输入法，主要介绍了数字电路设计、安装 MAX+plus II、绘图输入法、功能模拟、产生符号文件等内容；第 3 章是 HDL 语法，主要介绍了 AHDL，VHDL 等方面的内容；第 4 章是组合逻辑设计（一）——计算电路，主要介绍了半加器与全加器、半减器与全减器、加/减法器、BCD 加法器、BCD 减法器、BCD 加/减法器等内容；第 5 章是组合逻辑设计（二）——处理电路，主要介绍了多任务器、解多任务器、译码器、编码器、比较器、同位检查器与产生器等内容；第 6 章是触发器，主要介绍了门控 RS 触发器、JK 触发器、D 触发器、T 触发器、触发器的绘图描述法、触发器的 HDL 语言描述语法等内容；第 7 章是计数器，主要介绍了异步计数器、任意模数的异步计数器、同步计数器、含同步清除功能的计数器等内容；第 8 章是寄存器，主要介绍了移位寄存器、移位寄存器的种类、环形计数器、詹森计数器等内容；第 9 章是时序逻辑电路分析与设计，主要介绍了触发器的特征方程式、触发器的激发表、触发器的状态图、时序逻辑电路分类、时序逻辑电路分析、时序逻辑电路的设计等内容。

当然，要培养逻辑电路的设计能力，光看是不够的，最重要的是通过学习优秀的实例来增长经验，所以，笔者根据自己的实际经验，精心撰写了本书，希望能帮助更多的人提高设计逻辑电路的能力。

本书的完成，除了靠笔者本人的恒心与耐力之外，还得到了家人和很多朋友的帮助与鼓励，在此一并向他们表示感谢！

本书的重点在于数字逻辑电路的设计与实现，适合电子类专业的学生使用，也可供相关从业人员参考阅读。

作　者

目　录

第1章 逻辑概论

1.1 数字系统

相信每一个人对于十进制的数字系统都很了解，它很直观而且易于接受，例如，一支笔 25 元、现在温度是 28℃等，都是十进制数的表示方式。十进制的数字系统由 10 个阿拉伯数字 0，1，2，…，9 组成，一个十进制数的值即为每个位置的数字乘以相对应基底为 10 的“权衡值”，再相加而得，例如：

$$2637 = 2\times10^3 + 6\times10^2 + 3\times10^1 + 7\times10^0$$

千位数 2 的权衡值是 1000，百位数 6 的权衡值是 100，十位数 3 的权衡值是 10，个位数 7 的权衡值是 1。同理，十进数中如含有小数点位数的话，所表示的数值的结算方式仍相同，不同之处在于每个位置的权衡值不同而已，例如：

$$1.375 = 1\times10^0 + 3\times10^{-1} + 7\times10^{-2} + 5\times10^{-3}$$

小数点后一位数字的权衡值是 10^{-1}，后两位数字的权衡值是 10^{-2}，以此类推。

人们虽然习惯于使用十进数，但在数字系统硬件线路中，却不易制作。像晶体管之类的半导体组件，我们很容易控制晶体管作用于截止区（Cut Off，相当于关闭）和饱和区（Saturation，相当于导通）间的切换动作，这也是为什么计算机中采用二进制的原因。二进制的表示法与十进制的表示法相同，不同点在于是由 0 和 1 两种数值所组成，而且每个数值的权衡值是以 2 为底来表示，例如：

$$\begin{aligned}1101.01 &= 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2}\\ &= 13.25\end{aligned}$$

由上式中也可看出二进制数的 1101.01 等于十进制数的 13.25，这也说明了二进制数如何转换为十进制数。那么十进制数如何转换为二进制数呢？我们将十进位数分成整数部分和小数部分来讨论，因为十进制数转换为二进制数时，它的整数部分和小数部分是分开计算的。

（1）整数部分

利用“长除法”，将十进数连续除以 2，依序取其所有的余数（0 或 1）所组成的数列便是二进制数，第一次除以 2 所得的余数是二进制数的最低位（LSB），最后一次除以 2 所得的余数是二进制数的最高位（MSB），现举例说明如下。

2 | 123	余数		
2 | 61	……	1 ←	（LSB）
2 | 30	……	1	
2 | 15	……	0	
2 | 7	……	1	
2 | 3	……	1	
（MSB）→ 1	……	1	

所以，$123=(1111011)_2$。

（2）小数部分

利用“长乘法”，将十进数的小数部分连续乘以 2，然后依序取其进位至整数的值（1 或 0，0 表示没进位），第一次乘 2 所得的进位数是二进制小数部分的最高位（MSB），最后一次乘 2 所得的进位数是二进制小数部分的最低位（LSB），现举例说明如下。

$$\begin{aligned} 0.6875\times2 &= 1.375 \\ 0.375\times2 &= 0.75 \\ 0.75\times2 &= 1.5 \\ 0.5\times2 &= 1.0 \end{aligned}$$

所以上例中，十进制数的 0.6875 可以转换为二进制数的 0.1011。此例中我们连续乘 2 可以得到小数部分为零的情形，但若连续乘 2 仍无法得到小数部分为零的话，则计算至所需小数字数即可。

综合前面两个范例可知，十进位数 123.6875 可转换为二进制数的 1111011.1011。

虽然实际计算机中所处理的是二进制数，但是二进制数值若太大，则表示的位数（bit）将会很长，容易造成书写错误，此时可以采用八进制或十六进制的表示法。如前所述，八进制的数字系统是由 8 个阿拉伯数字 0，1，2，…，7 组成，一个八进制数的值即为每个位置的数字乘以相对应底数为 8 的权衡值再相加而得，例如：

$$(746)_8=7\times8^2+4\times8^1+6\times8^0=(486)_{10}$$

同理，十六进制的数字系统是由 0，1，2，…，9 这 10 个阿拉伯数字加上 A，B，C，D，E 这 5 个英文字母所组成，其中，A 代表数字 10，B 代表数字 11，以此类推，F 即代表数字 15，例如：

$$(1A2B)_{16}=1\times16^3+A\times16^2+2\times16^1+B\times16^0=(6699)_{10}$$

那么如何从二进制数转换为八进制数或十六进制数呢？方法非常简单，只要将二进制数由右向左每 3 个位为一组转换成八进制数，或每 4 个位为一组转换成十六进制数即可，例如，将二进制数 11010100111 转换成八进制数为：

$$(11010100111)_2=(3247)_8$$

又如，将 11110001001101 转换成十六进制数为：

$$(11110001001101)_2=(3C4D)_{16}$$

若二进制数中含有小数部分，则需将整数部分与小数部分分别转换。至于小数部分的转换方法，与前面的整数部分的转换方法一样，但是以小数点为基准，从左向右每 3 个位为一组转换成八进制数字即可，或每 4 个位为一组则转换成十六进制数字，例如，将二进制数 11010.10011 转换成八进制数为：

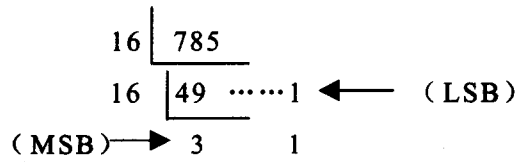
$$(11010.100110)_2=(3246)_8$$

注意：小数部分最后一位需补 0，以凑成 3 个位为一组。但若转换成十六进制数，则为 $11010.10011000=(1A.98)_{16}$。

至于如何将十进制数转换为八进制数或是十六进制数，方法如同十进制数转换成二进制数一样。整数部分以“长除 8”或是“长除 16”法，小数部分则以“长乘 8”或是“长乘 16”法，下面举例说明。

将十进制数 785.23 转换成十六进制数（至小数点后第三位）。

（1）整数部分



（2）小数部分

0.23×16=3.68

0.68×16=10.88

0.88×16=14.08

所以，$(785.23)_{10}=(311.3AE)_{16}$。

在本节中介绍了二进制数、八进制数、十进制数、十六进制数的表示方法，以及彼此间的转换关系。接下来介绍二进制数的四则运算，这是计算机中执行运算动作的基础。

1.2　二进制数的四则运算

由于计算机中是执行二进制数的数值运算，所以有必要对二进制数的四则运算加以说明。平常惯用的十进制加减运算法，逢 10 进到下一位数，如要借位时，向上一位数借 1，但对这一位数而言等于借到 10。同理，二进制数的运算是逢 2 进到下一位数，如要借位时，向上一位数借 1，但对这一位数而言等于是借到 2。二进制数单一位加法运算法则如下：

0+0=0
0+1=1
1+0=1
1+1=10　　（有进位至下一位）

单一位减法运算法则如下：

0−0=0
0−1=1　　（有向上一位借位）
1−0=1
1−1=0

单一位乘法运算法则如下：

0×0=0
0×1=0
1×0=0
1×1=1

现举例对二进制数的四则运算加以说明。

加法运算：

```
  101011
+ 110001
--------
 1011100
```

减法运算：

```
  101001
- 010011
--------
  010110
```

如果二进制数中含有小数部分，则将小数点对齐后，再执行加减法运算。

乘法运算：

```
    1001
  × 1010
--------
    0000
   1001
  0000
 1001
--------
 1011010
```

除法运算：

```
     101   ──→ 商
11 ) 10001
     11
    ----
     10
     00
    ----
     101
      11
    ----
      10   ──→ 余数
```

1.2.1　2 的补码

试想计算机内执行加法运算的硬件线路，如果也能执行减法运算，是不是可以简化硬件线路的复杂度呢？所以才有了补码概念的产生，补码的主要应用是将“减法运算”转换成“加法运算”。

那何谓“1 的补码”（Complement）呢？简而言之，就是将一个二进制数的 0 变成 1，1 变成 0 后的二进制数称为“1 的补码”，例如，101010 的“1 的补码”为 010101，

若以文字定义，则如下所示。

假设存在一个二进制数 M，其整数部分有 n 位，小数部分有 m 位，则其“1 的补码”定义为：2^n-M-2^{-m}。

例如，10.0101，其“1 的补码”为 $2^2-10.0101-2^{-4}$。

$$\begin{array}{rll} 100 & \cdots\cdots\blacktriangleright & 2^2 \\ -)\ 010.0101 & \cdots\cdots\blacktriangleright & M \\ \hline 001.1011 & & \\ -)\ 000.0001 & \cdots\cdots\blacktriangleright & 2^{-m} \\ \hline 001.1010 & & \end{array}$$

接着说明“2 的补码”的定义，假设存在有一个二进制数 M（$M\neq 0$，若 M=0 则其“2 的补码”为 0），其整数部分含有 n 位，不管是否存在有小数部分，其“2 的补码”定义为：2^n-M。

例如，10.0101，其“2 的补码”为 $2^2-10.0101$。

$$\begin{array}{rll} 100 & \cdots\cdots\blacktriangleright & 2^2 \\ -)\ 010.0101 & \cdots\cdots\blacktriangleright & M \\ \hline 001.1011 & & \end{array}$$

由以上的讨论也可看出，“2 的补码”可以由“1 的补码”来求出。其方法是将“1 的补码”值的最低位处加上 2^{-m} 求得（m 为小数部分所含有的位数，若无小数部分，则加 1），如上例中：

$$10.0101 \xrightarrow{0\to 1\text{和}1\to 0} 01.1010(1\text{的补码}) \xrightarrow{\text{加}0.0001} 1.1011(2\text{的补码})$$

另外一种求“2 的补码”的方法更为快捷，即将二进制数（不论是否含有小数部分）的最右边位处开始往左检查，遇到第一个“1”位时，其右边的个位值不变（含此位 1），而左边的个位值，“0”变成“1”，“1”变成“0”，例如：

$$\begin{array}{ll} & 10.0101 \\ 0\to 1\text{和}1\to 0 & \quad\downarrow\ \downarrow\ \text{不变} \\ & 01.1011 \end{array}$$

下面介绍将补码概念用于减法运算上的方法，先说明“2 的补码”的减法，例如，要求 A－B 的值。

① 先将 B 取“2 的补码”值－B。

② 执行 A+（－B）的加法运算。

③ 所得结果若有进位，则去掉进位后所得值即为结果（为正值）。

④ 若无进位，表示结果为负值，但真正的结果值需再执行一次“2 的补码”运算。

现举例说明“2 的补码”用于减法运算上的方法，例如，计算 1101－111（13－7）。

① 111 的“2 的补码”值为 1001。

② 执行 1101+1001 的加法运算。

③ 上式加法所得为 10110，所得结果产生了进位，则去掉进位后的值 0110（6）即为所得结果值。

另一例为：计算 111－1101（7－13）。

① 1101 的“2 的补码”值为 0011。

② 执行 0111+0011 的加法运算。

③ 上式加法所得为 1010，所得结果并没有产生进位，表示结果为负值，但真正的结果值需再将 1010 取“2 的补码”值得 0110，故所得结果为－110。

接下来说明“1 的补码”的减法，下面是求 A－B 的值的步骤。

① 先将 B 取 1 的补码值－B。

② 执行 A+（－B）的加法运算。

③ 所得结果若有进位，则将进位加至最小位处后面，所得值即为结果（为正值）。

④ 若无进位，表示结果为负值，但真正的结果值需再执行一次“1 的补码”运算。

现仍以上例说明“1 的补码”用于减法运算上的方法，如计算 1101－111（13－7）。

① 111 的 1 的补码值为 1000。

② 执行 1101+1000 的加法运算。

③ 上式加法所得为 10101，所得结果产生了进位，则将进位加至最小位处后，所得值即为结果值 0110（6）。

另一例，计算 111－1101（7－13）。

① 1101 的 1 的补码值为 0010。

② 执行 0111+0010 的加法运算。

③ 上式加法所得为 1001，所得结果并没有产生进位，表示结果为负值。但真正的结果值需再将 1001 取“1 的补码”值，得 0110，故所得结果为－110。

由上面的例子可知，不管是采用“1 的补码”还是“2 的补码”，所得的减法运算结果值是一样的。

本节中曾举例提及二进制数的加减法运算，如果被减数小于减数时，将会产生负数值。当时文字上只用“－110”来表示，但实际计算机中也必须用 0 或 1 来表示正（+）、负（–）符号，这样才能表现出有意义的符号数目来，所以，在二进制数系中是以最高位 （MSB）的 0 代表符号“+”，而用 1 代表符号“－”，例如，以 4 位为例，它所能表示的大小为 0（0000）～15（1111），但是若以第四位表示+/–号的话，则所能表示的大小范围为–7～+7，如表 1-1 所示，此种表示法称为“符号－大小”（Sign-Magnitude）表示法，一个 n 位的二进制数所能表示的数值范围为：$-(2^{n-1}-1)\sim(2^{n-1}-1)$。

表 1-1 “符号－大小”表示表

带号二进制数	相等十进制数	带号二进制数	相等十进制数
0000	+0	1000	–0
0001	+1	1001	–1
0010	+2	1010	–2
0011	+3	1011	–3
0100	+4	1100	–4

续表

带号二进制数	相等十进制数	带号二进制制数	相等十进制数
0101	+5	1101	−5
0110	+6	1110	−6
0111	+7	1111	−7

再举一例说明有符号数的表示法。–65 如何表示成二进制有符号数呢？首先将 65 表示成二进制数为 1000001，–65 则在最高位处加入符号位“1”，所以–65 的二进制有符号数表示为 11000001。

但是在数字系统中，表示负数最常使用的方法是“符号－2 的补码”表示法，如表 1-2 所示。前面曾说过“2 的补码”的优点是可以用加法运算来执行减法运算。

表 1-2　“符号－2 的补码”表示表

符号–2 的补码	相等十进制数	符号–2 的补码	相等十进制数
0000	+0	0000	–0
0001	+1	1111	–1
0010	+2	1110	–2
0011	+3	1101	–3
0100	+4	1100	–4
0101	+5	1011	–5
0110	+6	1010	–6
0111	+7	1001	–7
		1000	–8

以“符号–2 的补码”表示法来表示一个 n 位的二进制数，所能表示的数值范围为：$-2^{n-1}\sim(2^{n-1}-1)$。

1.3　BCD 码

何谓 BCD 码？我们以 4 个位为一组的二进制数来表示一个十进制数，就称为 BCD 码（Binary Code Decimal），如表 1-3 所示。

表 1-3　十进制数和 BCD 码

十进制数	BCD 码
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

4 位的二进制数可以表示 0～15 的十进制数值，因此，在 BCD 码中，从 1010（A）～

1111（F）止，但有 6 个二进制数是没有意义的，下面是一个二进制数的 BCD 码表达式：

5284＝0101 0010 1000 0100

BCD 码执行加减法运算需要注意，因为 BCD 码中没有定义从 1010～1111 的字码，以 BCD 码的加法运算而言，假如 BCD 码相加后的结果在 1010～1111 的字码中，则必须加以修正，修正的方式为：将相加后的结果再加 0110（6）予以修正。为什么要加 0110 修正呢？试想，如果两个 BCD 码相加后的结果是 1100（C，即 12），加 0110（6）后即变成 10010（18），最高位的 1 会进位至下一位的 BCD 码中，所以实际上此 BCD 码是 0010，也就是十进制数的 2。

例如，249+578＝？

```
   249    →  0010 0100 1001
+) 578    →  0101 0111 1000
-------------------------
           0111 10110 0001
```

个位（10001）与十位（1011）所得都超过 9（1001），所以需要加 0110 修正，先从个位开始修正。注意，个位（10001）的最高位 1 需要加至十位处（如下面○中的 1）。

```
   0111 1011 0001
        (1)0110  ..... 个位修正
+)      0110  ..... 十位修正
-------------------
   0111 0010 0111
+)   1  ..... 进位 1
-------------------
   1000 0010 0111 ....▶ 结果为 827
```

同理，BCD 码的减法运算，如果有发生向上一位借位的情形时，也必须予以修正，修正的方式为 BCD 码相减后再减 0110（6），为什么呢？因为向上借一位，等于是借了 10000（16），事实上 BCD 码向上借一位只借了 1010（10），所以多借了 6，要减回去。

例如，751－362＝？

```
   751    →  0111 0101 0001
-) 362    →  0011 0110 0010
-------------------------
           0011 1110 1111
```

上面减法运算所得结果中，有下划线的都有向上一位借位的情形发生，所以必须减 0110（6）来修正，如下所示。

```
   0011 1110 1111
        0110  ..... 个位修正
-)      0110  ..... 十位修正
-------------------
   0011 1000 1001 ....▶ 结果为 389
```

1.4　格雷码

什么是格雷码（Gray Code）？它是一种最少变化码，也就是说，格雷码从某一个数码变化到下一个数码时，一次只变化一个位，表 1-4 为 4 位的格雷码与其对应的十进制数和二进制代码，例如，十进制数由 11 变到 12 时，格雷码由 1110 变为 1010，只改变了第 3 个位的值（1→0），而二进制代码则由 1011 变为 1100，共改变了 3 个位值。

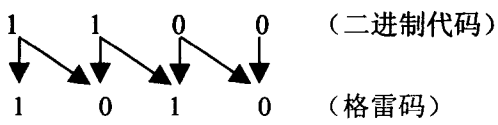
为什么要使用格雷码呢？是为了避免多位同时变化时所产生的误操作。虽名为“同时变化”，但由于各组件门传递延迟时间的不同，所以各位不会同时变化，例如，二进制代码由 1011 变为 1100，中间变化过程却可能产生 1010，1111，1001 等瞬时码，这些可能产生的瞬时码会对组件产生错误的动作，而使用格雷码一次只变化一个位，从而可以避免这种现象的产生。

格雷码有什么用呢？格雷码不像二进制代码是属于权重码，它是非权重码，它的每一个位不代表任何数值上的意义，所以它不适合于算数运算。但它适合在输入输出装置上使用，格雷码常用于决定转轴的位置，当转轴转动时，相邻两位置间一次只改变一个位的变化，对于后级的数字检测电路，可避免因为多位改变而产生错误。

表 1-4　4 位的格雷码与其对应的十进制数和二进制代码

十进制数	格雷码	二进制代码
0	0000	0000
1	0001	0001
2	0011	0010
3	0010	0011
4	0110	0100
5	0111	0101
6	0101	0110
7	0100	0111
8	1100	1000
9	1101	1001
10	1111	1010
11	1110	1011
12	1010	1100
13	1011	1101
14	1001	1110
15	1000	1111

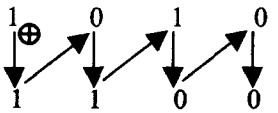
格雷码与二进制代码之间是如何转换的呢？以 4 位为例说明如下，例如，二进制代码 1100 转换为格雷码是 1010，转换方法为二进制代码的最高位即是格雷码的最高位，然后“二进制代码从最高位开始，每两个位依次执行互斥或门运算”，即可得到相对应的格雷码。



所以，n 位二进制代码转换为格雷码的数学关系式如下所示。

$$G_n=B_n$$
$$G_i=B_i \oplus_{i+1} \quad (i=0,1,\cdots,n-1)$$

格雷码如何转换为二进制代码呢？格雷码 1010 转换为二进制代码是 1100。转换方法是：格雷码的最高位即是二进制代码的最高位，然后从格雷码的最高位开始与次高位执行互斥或门运算，所得结果（即为二进制代码）再与格雷码第三高位执行互斥或门运算，以此类推，即可求出二进制代码。



（格雷码）

（二进制代码）

所以，n 位格雷码转换为二进制代码的数学关系式如下所示。

$$B_n=G_n$$
$$B_i=G_i \oplus B_{i+1} \quad (i=n-1,n-2,\cdots,0)$$

1.5 基本逻辑组件

人是一个构造非常精密的生命体，但对于如此复杂的构造，组成其各器官的基本单元却非常简单——那就是细胞。相同的道理，对于数字逻辑电路，无论简单的半加器还是复杂的计算机，其组成的基本单元也很简单，除了触发器（Flip-Flop）之外，其余的即是本节所要介绍的如 AND，OR，NOT，INV，NAND，NOR，XOR，以及 XNOR 等逻辑门，要了解复杂的数字逻辑电路，基本逻辑组件的功能不可不知。

1. AND

AND 门（与门）有两个或两个以上的输入端口以及一个输出端口，输入与输出的关系如下所示。

- 当任何一个或一个以上的输入端口为 0（Low）时，则输出为 0。
- 当所有的输入端口均为 1（High）时，则输出为 1。

若以真值表表示两输入端口的 AND 门，则如表 1-5 所示。

表 1-5 AND 门的真值表

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

将 AND 门写成数学表达式，定义“·”为 AND 门运算符号，所以上式真值表表 1-5 可写成：