

6.111演讲12

二进制算术：多数运算都是相似的

二进制数的每一位的值都是 2^n

$$\begin{array}{rcl}
 5 & = & 00000101 = 1 + 4 \\
 19 & = & 00010011 = 1+2+16
 \end{array}$$

$$\begin{array}{rcl}
 5 & + & 00000101 \\
 19 & & 00010011 \\
 \hline
 & = & 00011000 \quad 24 = 16+8
 \end{array}$$

加法有时候需要“进位”

$$\begin{array}{rcl}
 19 & - & 00010011 \\
 5 & = & 00000101 \\
 \hline
 & & 00001110 \quad 14 = 8+4+2
 \end{array}$$

减法有时候需要“借位”

1

负数的表示：有很多种方法：

1.“符号位”的使用（这就像数带有符号）

$$-5 = 10000101$$

注意加法和减法稍微有点复杂（以及乘法和除法）。

一般先去掉符号位，做运算，然后处理结果的符号位。

2.“反码”：每一位取反。在这方面我们不作更多的讨论。

3.“二进制补码”：逐位取反，然后加1。

2

如果我们做这个运算会怎样：

$$\begin{array}{rcl}
 5 & & 00000101 \\
 -19 & & 00010011 \\
 \hline
 & = & 11110010
 \end{array}$$

注意两点：

1.我们必须从左边“借位”

2.得到的是-14的二进制补码的表达式：

$$\begin{array}{rcl}
 14 & = & 00001110 \\
 -14 & = & 11110001 \\
 & & \quad \quad + 1 \\
 \hline
 & = & 11110010
 \end{array}$$

3

二进制补码是相容的和可逆的：

$$\begin{array}{rcl}
 5 & = & 00000101 \\
 -5 & = & 11111010 + 1 = 11111011 \\
 5 & = & 00000100 + 1 = 00000101
 \end{array}$$

二进制补码之间加法和减法的过程：

$$\begin{array}{rcl}
 -5 & = & 11111011 \\
 +(-19) & & 11101101 \\
 \hline
 & = & 11101000 \quad (\text{这是}24) \\
 & & 00010111+1=00011000=16+8
 \end{array}$$

4

5.5 = 00000101.1
 5.0 = 00000101.0
 -5.0 = 11111010.1
 + 1
 = 11111101.0

许多情况下我们希望扩展数：使用更多的“二进制位”来表达一个数。我们怎样做这种扩展？

为了扩展一个数（用更多的位表示）而不改变值：

如果一个数是：向左扩展 向右扩展
 正 0 0
 负 1 0

5

19 = 00010011
 x -5 = 11111011
 = 00010011
 + 00010011
 = 00111001
 + 00010011
 = 00011010001
 + 00010011
 = 00100000001
 + 00010011
 = 001000110001
 + 00010011
 = 0010010010001
 + 00010011
 = 00100101010001

1010001是
 0101110+1的负：
 = 01011111
 = 64+16+8+4+2+1=95

6

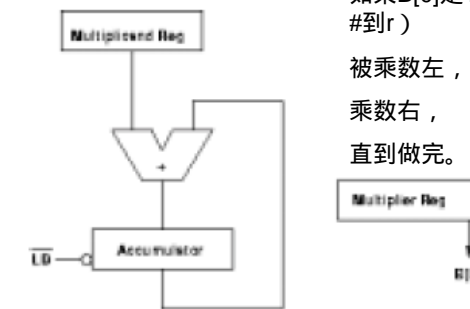
现在考虑怎样做一个简单的乘法

9 = 00001001
 x 13 = 00001101
 = 00001001
 + 00001001
 = 0000101101
 + 00001001
 = 00001110101 (117)

这包括最上面的数不停的向左移位，再把它加到部分和。这种方法运算良好，需要一个和乘积一样宽的移位寄存器，还要存储部分以及最后的乘积。

7

乘法器的硬件描述



如果B[0]是1，装载（加#到r）
 被乘数左，
 乘数右，
 直到做完。

8

```

9      =    00001001
x 13   =    00001101
      =    0000001001
      +    00001001
      =    00000101101
      +    00001001
      =    00001110101

```

一个选择是把部分积向右移位
(相同的数移位)

9

这里是怎么样对负数处理，我们必须扩展符号（对数增加位，补1）

```

-9     =    11110111
x 13   =    00001101
      =    111110111
      +    11110111
      =    11111010011
      +    11110111
      =    111110001011
(-)    000001110101

```

(记住符号扩展)
(-117)

10

“符号扩展”包括移1到最高有效位（MSB），如果它是负数

```

-9     =    11110111
x 13   =    00001101
      =    111110111
      +    11110111
      =    11111010011
      +    11110111
      =    111110001011
(-)    000001110101

```

(记住符号扩展)
(-117)
= 1+4+16+32+64=117

11

基于符号/数值表示的二进制补码乘法

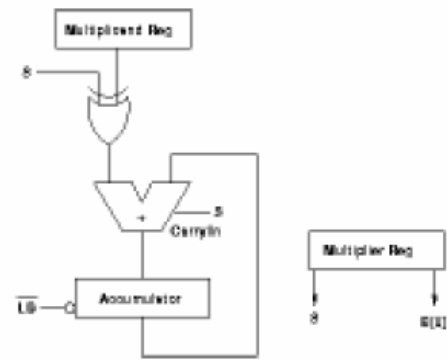
这是一种情况

1.使用符号/数值表示的数作乘数

2.如果最高有效位是1（负数），对被乘数作二进制补码。

12

如果S = 1，XOR对被乘数的每一位取补，而且进位加S（如果S被置位，就是1）。如果乘数是正的，被乘数不取补，进位为0。



13

更多的乘法选择

加X输入y次

- 装载Y到下降计数器
- 当计数下降的时候，每次增加X
- 当计数器到0的时候停止
- 硬件很便宜
- 时间不确定（可能很长）
- 而负数该怎么办？

移位和加法

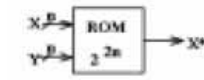
- 这是我们用过的技术
- 在初中就学过的技术
- 有多少位就需要多少循环
- 用一个寄存器来做乘数和累加器
- 如果注意符号扩展，能够一致的处理负数

15

许多问题需要乘法。正如我们说过的，有很多选择：
选择：

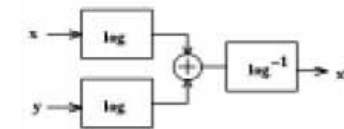
查找表

- 每一个输入是n位宽
- ROM存储 2^{2n} 个答案
- 快速但是使用更多的硅



对数转换

- 转换为对数
- 加
- 反对数转换
- 快速，可能使用较少的硅
- 精确度不确定
- 对负数怎么做



14

例：这是要控制的物理系统：

电车的控制（轻型车）

应用于轮子的控制力

速度由驱动器(控制杆)控制

适应汽车的重量：命令速度和实际汽车速度之间的差异

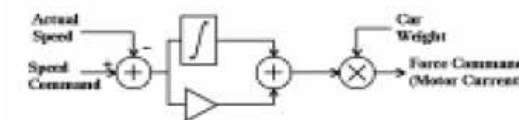
就是加速度。

$$F = MA \quad (\text{需要的力就是加速度乘汽车质量})$$

我们将用PI控制：

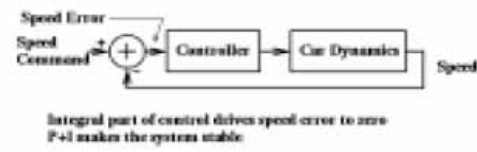
积分部分驱动误差到0

比例部分给出稳定性



16

使用反馈控制



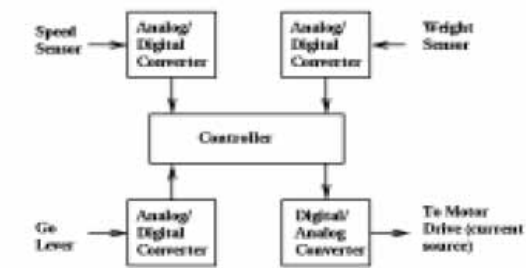
通过测量实际的速度，以及回到PI控制器的反馈，我们能驱动速度误差（指数的）到0。

这样在这个例子中，我们将考虑如何建立控制器。我们假定一个很理想的驱动器，其驱动力与命令马达电流成比例。

17

和其他设计一样，从高层的框图开始设计：这是输入和输出。

速度传感器和“控制杆”分别代表真实的，和要求的速度：通过A/D重量由装载单元或者空气支持系统压力测量
输出是对马达/驱动器（假定工作正常）的电流命令



18

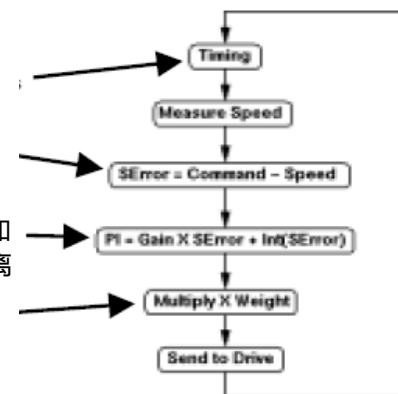
系统的流程图

定时产生一个固定的间隔，控制系统在这段时间内完成功能。

速度误差是测量值和命令值之间的差

PI是比例和积分信号的加法。积分这里是附加的离散信号。

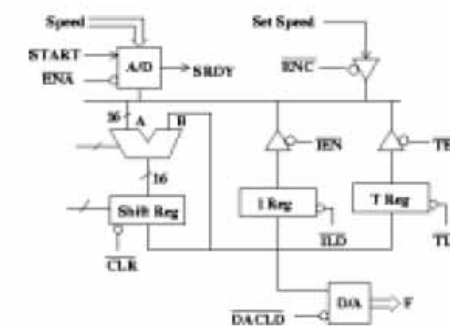
哈：乘将是有意思的。



19

PI控制器的一种可能的数据通路

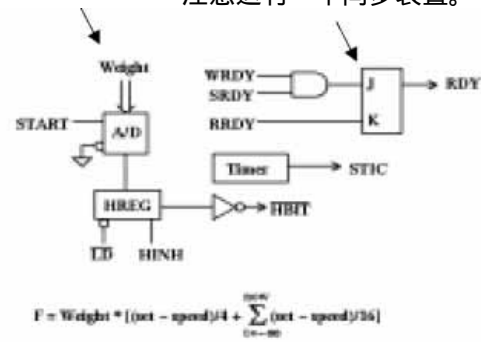
积分由和近似可求



20

PI控制器数据通路 (B)

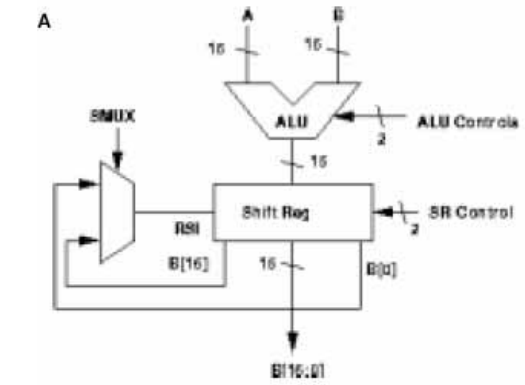
当用PI的时候，我们必须乘上权值
 数据通路的部分要把乘处理为移位和加法。
 注意这有一个同步装置。



21

这是算术部分

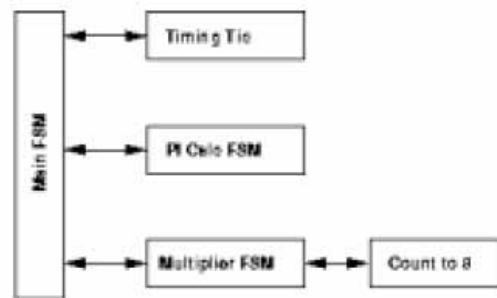
- ALU 控制：
- 00 F = A 的反
 - 01 F = A + 1
 - 10 F = A + B
 - 11 F = A - B
- SR控制：
- 00 保持
 - 01 移位R
 - 10 清零
 - 11 装载
- SMUX：
- 0 旋转
 - 1 符号扩展



注意：不是所有算法都用到！

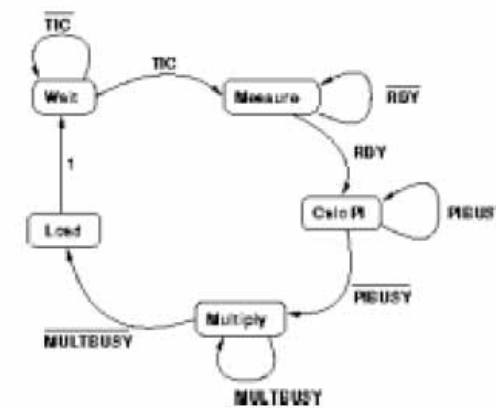
22

可以用一个大的有限状态机来控制，但是似乎把控制器拆为几个小的（更容易开发和测试）有限状态机更为合理，而这些有限状态机必须是并列的。



23

这是我关于“主”有限状态机的最初设想，它协调所有进程。它启动其他进程并等待他们完成，然后继续。注意TIC由一个有限状态机（一个计数器）产生，它不受主有限状态机的控制。



24