使用MAXQ乘法-累加单元 (MAC) 进行信号处理

在模块化的MAXQ架构中，单周期乘法-累加(MAC)单元的加入使典型的信号处理技术所需的运算变得容易。

传统微控制器与数字信号处理器 (DSP)有时被视作微型计算机领域中两个对立的极端。微控制器最适合于需要对非同步过程实现低延迟响应的控制应用，而DSP在需要高强度数学计算的应用中表现卓越。微控制器可以被用在繁重算术应用中，但是由于绝大多数微控制器的ALU具有每次一项运算的特性，使这样的应用不甚理想。同理，DSP也可以勉强用于控制应用中，但是绝大多数DSP的内部架构在这种操作中会反映出代码与时间上的低效率。

当越来越多与控制有关的应用都需要少量信号处理时，在DSP与传统微控制器之间进行选择变得越来越困难。在这样的应用中，将DSP代码挤进微控制器的解决方法是很吸引人的。不过，通常设计者会发现应用程序的大部分时间都花在执行DSP函数上了，而控制应用不得不有所牺牲。

这样两难的问题可以通过现代的处理器架构来解决，例如MAXQ架构。在模块化的MAXQ架构中，可以很容易地将乘法-累加单元 (MAC)加入到设计中，并集成到架构中。使用了硬件MAC后，16 x 16的乘法-累加运算出现在一个周期中，而不会影响在控制处理器上运行的应用。本文提供了一些实例，用来说明如何使用典型MAXQ微控制器中的MAC模块解决实际问题。

MAC模块与MAXQ结合使用

对DSP来说，最普通的应用是对模拟信号的滤波。在滤波应用中，将经过适当调理的模拟信号提供给ADC，在数字域中对采样得到的数据流进行滤波。常规滤波器的执行过程可以用如下公式实现：

$$y[n] = \sum b_i x[n-i] + \sum a_i y[n-i]$$

式中b_i和a_i分别是系统前馈与反馈响应的特征值。

根据a_i和b_i的值，可以将数字滤波器分为两大类：有限冲击响应 (FIR)与无限冲击响应 (IIR)。系统不包含任何反馈元素时 (所有$a_i = 0$)，滤波器被称作FIR型：

$$y[n] = \sum b_i x[n-i]$$

而当a_i和b_i的元素都不为0时，系统是IIR滤波器。

从上文FIR滤波器的公式中可以看到，主要的数学运算是将每个采样输入与常数相乘，然后将每个乘积累加，共进行n次。这可以用如下C语言程序段来说明：

```
y[n]=0;
for(i=0; i<n; i++)
        y[n] += x[i] * b[i];
```

对带乘法器单元的微处理器，可以根据以下伪码来实现：

```
move ptr0, #x       ;Primary data pointer -> samples
move ptr1, #b       ;Secondary DP -> coefficients
move ctr, #n        ;Loop counter gets number of samples
move result, #0     ;Clear result register
```

```
ACC_LOOP:
        move acc, @ptr0    ;Get a sample
        mul   @ptr1        ;Multiply by coefficient
        add   result       ;Add to previous result
        move  result, acc  ;...and save the result back
        inc   ptr0         ;Point to next sample
        inc   ptr1         ;Point to next coefficient
        dec   ctr          ;Decrement loop counter
        jump  nz, ACC_LOOP ;Jump if there are more samples
        end
```

电话网络中使用的双音多频(DTMF)信令技术用来从网络终端(电话或其他设备)向交换机传送地址信息。

这样，尽管有一个乘法器，乘法与累加的循环也需要12条指令以及(假定是单周期运行的单元与乘法器) 4 + 8n个周期。

MAXQ的乘法器是真正的乘法-累加单元。在MAXQ架构中完成相同的操作，代码空间将从12个字缩减到9个字，运行时间将减少到4 + 5n个周期。

```
        move DP[0], #x        ; DP[0] -> x[0]
        move DP[1], #b        ; DP[1] -> b[0]
        move LC[0], #loop_cnt ; LC[0] -> number of samples
        move  MCNT, #INIT_MAC ; Initialize MAC unit
MAC_LOOP:
        move DP[0], DP[0]     ; Activate DP[0]
        move MA, @DP[0]++     ; Get sample into MAC
        move DP[1], DP[1]     ; Activate DP[1]
        move MB, @DP[1]++     ; Get coeff into MAC and multiply
        djnz  LC[0], MAC_LOOP
```

需要注意的是，在MAXQ乘法-累加单元中，当第二个操作数装入该单元时，请求的运算自动执行。其结果存储在MC寄存器中。还应当注意的是，MC寄存器的长度是40位，在溢出前可以累加大量32位的乘法计算结果。这是对传统方法的改善，在传统方法中每个基本运算后都必须进行溢出检测。为了说明如何在信号处理流程中高效的使用MAC，这里给出一个用于双音多频(DTMF)收发器的简单应用。

DTMF概述

DTMF是用于电话网络中的信令技术，从网络终端(电话或其他设备)向交换机传送地址信息。其机理是使用两组各四路独立音调，互相之间没有谐波相关关系，例如“低频组”(低于1kHz)与“高频组”(高于1kHz)。电话键盘上的每个数字都刚好用一路低频组音调和一路高频组音调来表示。**图1**给出了这些音调的分配。

...在MAXQ乘法-累加单元中，当第二个操作数装入该单元时，请求的运算自动执行。

DTMF音调编码器

DTMF收发器的编码器部分相当简单明了。需要两个数字正弦振荡器，每个振荡器都可以调谐到四个低频组频率或高频组频率之一。

有很多方法可以用来解决数字合成正弦波的问题。产生正弦波的方法之一是完全取消数字合成。而只是对端口引脚产生的方波进行较强的滤波。尽管这种方法在很多应用中都有效，但是Bellcore的必要条件对正弦波频谱纯度的要求高于这种技术所能实现的水平。

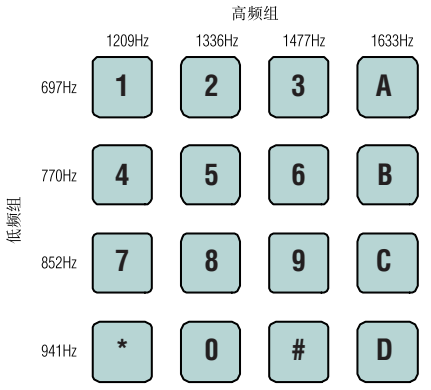

图 1. 用高频组中的一个频率与低频组中的一个频率组合产生 DTMF 信号。

第二种方法是利用速查表来产生正弦波。在这种方法中，将一路正弦波的1/4存储在一张ROM表中，这张表对预先计算的间隔采样来生成所需的波形。不过，要生成一个分辨率足够高以能够满足频谱要求的1/4正弦表，需要很大的存储量。幸而还有更好的方法。

递归数字谐振器[1]可以用来产生正弦波 (图 2)。该谐振器用双极点滤波器来实现，该滤波器可以用以下差分方程描述：

$$X_n = k * X_{n-1} - X_{n-2}$$

其中 *k* 是一个常数，其定义为：

$$k = 2 \cos(2\pi * \text{音调频率} / \text{采样速率})$$

由于 DTMF拨号盘只需要少量音调，k的 8个数值可以预先计算，并存储在ROM中。例如，在 8kHz采样速率下产生第一列音调 (770Hz)所需的常数为：

$$k = 2 \cos(2\pi * 770 / 8000) = 2 \cos(0.60) = 1.65$$

还有一个数值必须计算：振荡器开始运行所需的初始脉冲。很明显，若 X_{n-1}与 X_{n-2}都为0，随后的每个 X_n都将为0。为了启动振荡器，将 X_{n-1}设置为0，并将 X_{n-2}设置为

$$X_{n-2} = -A * \sin(2\pi * \text{音调频率} / \text{采样速率})$$

在我们的实例中，假定需要单位正弦波，则上式可简化为：

$$X_{n-2} = -1 * \sin(2\pi * 770 / 8000) = -\sin(0.60) = -0.57$$

将上式简化为代码很简单：第一，初始化两个中间变量 (X1，X2)。X1初始化为0，而 X2装入初始激励数值 (以上已计算)启动振荡。执行以下操作产生正弦波的一次采样：

$$X0 = k * X1 - X2$$
$$X2 = X1$$
$$X1 = X0$$

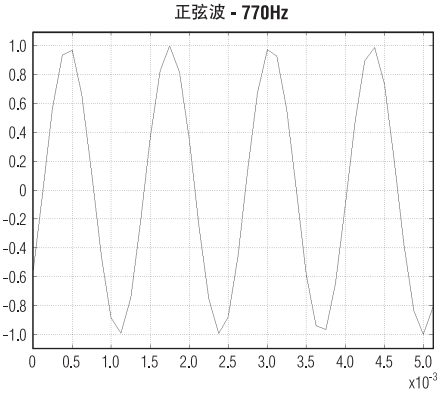

图 2. 递归谐振器产生的正弦波。

每个新的正弦值都用一次乘法与一次减法来计算。使用MAXQ微控制器上的单周期硬件MAC，正弦波可以用如下代码产生：

MAXQ微控制器，与其MAC单元相结合，在传统微控制器与数字信号处理器之间架起了桥梁。

```
move DP[0], #X1             ; DP[0] -> X1
move MCNT, #INIT_MAC        ; Initialize MAC unit
move MA, #k                 ; MA = k
move MB, @DP[0]++           ; MB = X1, MC=k*X1, point to X2
move MA, #-1                ; MA = -1
move MB, @DP[0]--           ; MB = X2, MC=k*X1-X2, point to X1
nop                         ; wait for result
move @--DP[0], MC           ; Store result at X0
```

DTMF音调的检测

由于只需要检测少量频率，因此这里使用了改进的Goertzel算法[2]。该算法比一般的DFT机制更有效，并提供对频段内信号的可靠检测。它可以用简单二阶滤波器来实现，如**图3**所示。

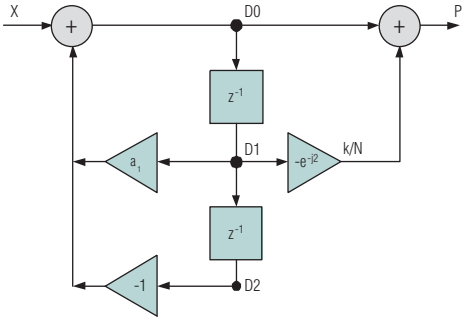

图3. 用二阶滤波器实现Goertzel算法。

为了用Goertzel算法来检测特定频率的音调，必须先计算一个常数。对DTMF检测器，该常数可以在编译时计算。所有的音调频率都已明确地指定了。按以下公式计算该常数：

$$k = \text{音调频率} / \text{采样速率}$$
$$a_1 = 2\cos(2\pi k)$$

首先，将三个中间变量(D0、D1与D2)初始化为0。现在，对接收到的每个采样值X，按以下公式计算：

$$D0 = X + a_1 * D1 - D2$$
$$D2 = D1$$
$$D1 = D0$$

在获得了足够数量的采样值后(若采样速率为8kHz，通常为205)，使用最新计算出的D1与D2计算下式：

$$P = D1^2 + D2^2 - a_1 * D1 * D2$$

这样P中就包含了输入信号中测试频率的平方幂。为了解码全部四列DTMF，每个采样将通过8个滤波器完成。每个滤波器将有自己的k值，以及自己的一组中间变量。由于每个变量都是16位的，整个算法将需要48个字节的中间存储。

一旦计算了不同音调频率的P值，高频与低频组中的一路音调将比其他音调的数值高得多，这意味着高出两倍以上，而通常其幅度高出一个数量级。**图4**所示是提供给解码器的采样输入信号，**图5**给出了Goertzel算法的结果。若信号频谱达不到这个标准，就意味着两种情况，要么是信号中没有DTMF能量，要么是噪声太大阻断了信号。

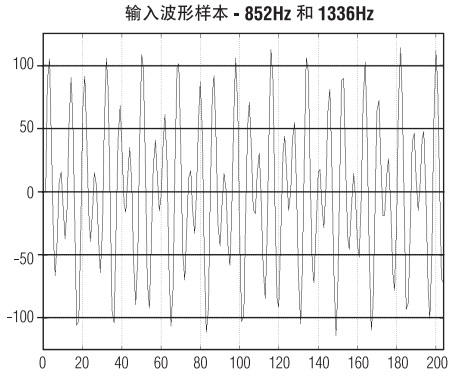

图4. DTMF解码器的输入波形样本。

我们的网站上提供用来说明该算法的电子表格，以及配备了MAC的MAXQ处理器的实例代码。请访问www.maxim-ic.com.cn/MAXQ_DTMF。

结论

MAXQ微控制器，与其MAC单元相结合，在传统微控制器与数字信号处理器之间架起了桥梁。添加了硬件MAC之后，MAXQ微控制器具备了过去16位微控制器不具备的高级信号处理能力。单周期的MAC提供了实际应用最经常需要的函数，使实时信号处理成为可能。

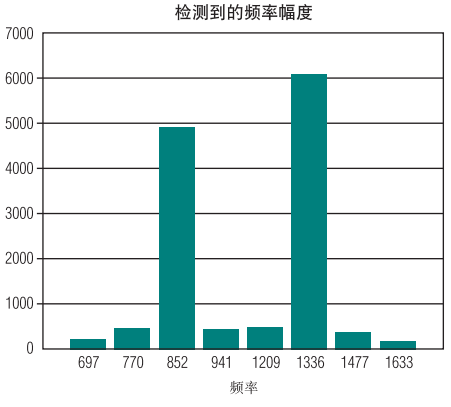

图5. DTMF解码器检测得到的不同频率信号的幅度。

[1] Todd Hodes, John Hauser, Adrian Freed, John Wawrzynek, and David Wessel. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-99, March 15–19, 1999), pp. 993–996.
[2] Alan Oppenheim and Ronald Schafer, Discrete-Time Signal Processing. Prentice Hall.