Application Note 5

Calculating Polynomials

Micromega Corporation

This application note describes how to use the uM-FPU floating point coprocessor to calculate polynomials.

Introduction

A polynomial is a mathematical expression that is a sum of terms, with each term being a constant times the product of a variable raised to a power. The general form of a polynomial with a single variable, expressed from the lowest term to the highest term, is as follows:

$$y = c_0 + c_1x^1 + c_2x^2 \ldots + c_nx^n$$

Alternatively, a polynomial can be expressed from the highest term to the lowest term:

$$y = c_nx^n + c_{n-1}x^{n-1} \ldots + c_1x^1 + c_0$$

The c_0, $c_1 \ldots c_n$ values are constants, called coefficients. The subscript on a coefficient matches the exponent of the x variable in each term. The value n is the highest exponent of the variable x and is referred to as the order of the polynomial. This application note shows examples of calculating 1st order, 2nd order and nth order polynomials.

1st order polynomials

$$y = c_1x^1 + c_0$$

A 1st order polynomial has just two terms and is the same as the equation of a line. The variable y has a linear response to the variable x. The common form of the equation for a line is:

$$y = mx + b$$

Where m (the c_1 coefficient) is the slope of a line, and b (the c_0 coefficient) is the y-intercept for a line. Assuming uM-FPU registers X, Y, M and B have been previously defined, the uM-FPU instructions for calculating the equation of a line are:

```
SELECTA+Y               ; select Y as A register
FSET+M                  ; Y = M
FMUL+X                  ; Y = Y * X
FADD+B                  ; Y = Y + B
```

2nd order polynomials

$$y = c_2x^2 + c_1x^1 + c_0$$

Many devices have non-linear responses. To compensate for the non-linearity, an n[th] order polynomial calculation is required, where n > 1. For example, the Sensirion SHT1x/SHT7x Relative Humidity and Temperature Sensors datasheet defines the calculation for Relative Humidity as the following:

$$RH_{linear} = c_1 + c_2 \bullet SO_{RH} + c_3 \bullet SO_{RH}^{\ 2}$$

This as a 2nd order polynomial, since the highest power is 2. If you don't immediately recognize this equation as the being the same as the general equation shown above, do the following:

- replace RH_{linear} with y
- replace SO_{RH} with x
- replace c_1, c_2, c_3 with c_0, c_1, c_2

To implement the 2nd order polynomial you can rewrite the general equation

$$y = c_2x^2 + c_1x^1 + c_0$$

as follows:

$$y = \left(\left(c_2 \bullet x\right) + c_1\right) \bullet x + c_0$$

This has the advantage of only requiring one multiply and one addition per term. The variable x doesn't need to be raised to a power for each individual term, instead it is calculated incrementally as each term is processed. The calculation can proceed from left to right with no temporary values. The parentheses are just added to make the equation easier to read. The intermediate result as each step of the equation is done is as follows:

step 1 : $y = c_2 \bullet x$

step 2 : $y = c_2x + c_1$

step 3 : $y = (c_2x + c_1) \bullet x$

step 4 : $y = c_2x^2 + c_1x + c_0$

Using this method, the uM-FPU instructions for calculating polynomial are very efficient. Assuming uM-FPU registers `X, Y, C0, C1` and `C2` have been previously defined, the uM-FPU instructions for calculating a 2nd order polynomial are:

```
SELECTA+Y               ; select Y as A register
FSET+C2                 ; Y = C2
FMUL+X                  ; Y = Y * X
FADD+C1                 ; Y = Y + C1
FMUL+X                  ; Y = Y * X
FADD+C0                 ; Y = Y + C0
```

n^{th} order polynomials

$$y = c_n x^n + c_{n-1} x^{n-1} \ldots + c_1 x^1 + c_0$$

The method described for calculating 2nd order polynomials can be generalized for nth order polynomials. Only one multiply and one addition is required for each term of a polynomial. The algorithm for calculating an nth order polynomial is as follows:

initialize *y* to zero
for each coefficient starting with the highest and going to lowest
    multiply *y* by *x*
    add the coefficient

The uM-FPU implementation is quite straightforward. Assuming uM-FPU registers `X` and `Y` have been previously defined and the coefficients are stored in memory as 32-bit floating point constants. The uM-FPU instructions for calculating nth order polynomials are:

```
SELECTA+Y                ; select Y as the A register
XOP, LOADZERO            ; set register 0 to 0.0
FSET                     ; Y = 0.0

loop for i = n to 0
{
   FMUL+X                ; Y = Y * X
   WRITEB
   C[i] (1st byte)       ; load next coefficient to register 0
   C[i] (2nd byte)
   C[i] (3rd byte)
   C[i] (4th byte)
   FADD                  ; Y = Y + coefficient
}
```

The XOP, POLY instruction

uM-FPU V2 provides a `XOP, POLY` instruction for calculating n^{th} order polynomials. The instruction can only be used as part of a stored function, since the coefficient values are stored in Flash memory as part of the instruction. This can be a very efficient way to calculate polynomials. The following example defines function 1 to calculate the 1^{st} order polynomial $y = 5x + 1$.

```
Function 1
   XOP, POLY, 2
   5.0
   1.0
```

The `POLY` instruction is followed by a byte specifying the order of the polynomial, and then a 32-bit floating point value for each coefficient starting from c_n (the highest coefficient) to c_0 (the lowest coefficient).

If a term is not used, the coefficient is simply entered as zero. For example, polynomial $y = 5x^2 + 1$, is a 2^{nd} order polynomial with no x^1 term. It would be calculated using the following `POLY` instruction:

```
Function 1
   XOP, POLY, 2
   5.0
   0.0
   1.0
```

Further Information

Sample programs that calculate polynomials using the uM-FPU floating point coprocessor are available for various microcontrollers.

Check the Micromega website at www.micromegacorp.com for up-to-date information.