

Signed Multi-Byte Multiplication

By: Dave Van Ess

Associated Project: *signedmath.zip*

Associated Part Family: CY8C25xxx, CY8C26xxx

Summary

A built in MAC (Multiply/Accumulate) assists the PSoC™ MCU with digital signal-processing applications. The MAC takes two signed single-byte inputs as multiplicands and generates a signed 2's compliment result. But what if multi-byte signed multiplication is needed? This Application Note will show that with a little understanding of the interaction between signed and unsigned values, fast signed multi-byte multiplication algorithms can easily be developed.

Introduction

Publication of [Application Note AN2032](#) (Unsigned Multiplication) generated a lot of interest with users. However, many requested information, and actual routines, for signed multi byte multiplication. Please view this as a continuation of AN2032. It is recommended that you read it first.

The interaction between signed and unsigned values is discussed and techniques will be shown that allow the PSoC MCU to quickly multiply multiple-byte signed multiplicands and give a full resolution signed output. Software to multiply two 16-bit signed values will be developed.

This application uses the following PSoC microcontroller resources:

- MAC (Multiply/Accumulate)

The PSoC Multiplier

Figure 1 shows that four registers are used to access the multiplier.

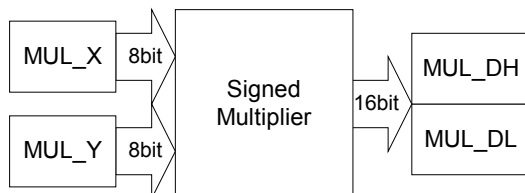


Figure 1: PSoC Multiplier Block Diagram

These four registers are all located in register Bank 0, having the following names and addresses:

- MUL_X (Bank 0 E8h)
- MUL_Y (Bank 0 E9h)
- MUL_DH (Bank 0 EAh)
- MUL_DL (Bank 0 EBh)

MUL_X and MUL_Y are “write only” registers where signed, 2's-complement 8-bit values are input to the multiplier. The signed, 2's-complement 16-bit output is deposited into two 8-bit “read only” registers, MUL_DH (upper) and MUL_DL (lower).

Part Signed, Part Unsigned

The two bytes that make up a 16-bit signed word are a combination of a signed upper byte and an unsigned lower byte. Equations (1) and (2) show how these multiplicands are represented:

$$x_{16} = 256x_u + x_l \quad (1)$$

$$y_{16} = 256y_u + y_l \quad (2)$$

Equations (3) and (4) show that the multiplication of two, 16-bit values can be broken down into the four multiplications of two, 8-bit values.

$$x_{16} \cdot y_{16} = (256x_u + x_l) \cdot (256y_u + y_l) \quad (3)$$

$$x_{16} \cdot y_{16} = 256^2 x_u y_u + 256(x_u y_l + x_l y_u) + x_l y_l \quad (4)$$

Of the four multiplications...

- One is a signed byte times signed byte operation.
- Two are signed byte times unsigned byte operations.
- One is an unsigned byte times unsigned byte operation.

The first is already in the form the multiplier requires. The third has already been covered in Application Note AN2032. This just leaves the case of a signed byte and unsigned byte multiplication.

The difference between signed and unsigned values is how the most significant bit is interpreted. Equation (5) defines the relationship between unsigned and signed values:

$$x_{unsigned} = x_{signed} + 256f(x) \quad f(x) = \begin{cases} 1 & x_{bit7} = 1 \\ 0 & x_{bit7} = 0 \end{cases} \quad (5)$$

Equation (6) shows that the multiplication of a signed and an unsigned byte yields a signed result.

$$Result_{signed} = y_{signed} x_{unsigned} \quad (6)$$

Combining Equations (5) and (6) produces an expanded Equation (7).

$$Result_{signed} = 256 f(x) y_{signed} + x_{signed} y_{signed} \quad (7)$$

As stated in Equation (6), multiplication of signed byte and unsigned byte produces a signed 2-byte result. Parsing Equation (7) yields the following results:

- $[X_{signed}Y_{signed}]$ is the 2-byte output of the signed multiplier.
- $[256f(y)X_{signed}]$ has the effect of adding X_{signed} to the upper byte of the result if y_{bit7} is 1.

Equation (8) and (9) puts this together to show how the resultant upper byte and lower byte for an unsigned multiply are calculated:

$$(Result_u)_{signed} = MUL_DH + 256f(x)y_{signed} \quad (8)$$

$$(Result_l)_{unsigned} = MUL_DL \quad (9)$$

For either signed or unsigned operands, the lower byte of the resultant is always the same.

The macro below calculates the signed MSB of the multiplication of an unsigned value “x” and of a signed value “y”:

```
macro GetXuYsMSB
; @0 @1 & MUL_DH determine value
mov A, reg[MUL_DH]
tst [0], 80h
jz .+4
add A, [01]
;endif
endm
```

To simplify matters, the macro below calculates the signed MSB of the multiplication of a signed value “x” and of an unsigned value “y”:

```
macro GetXsYuMSB
; @0 @1 & MUL_DH determine value
mov A, reg[MUL_DH]
tst [01], 80h
jz .+4 ;
add A, [00] ;endif2
endm
```

These macros are found in “*signedmath.inc*,” located in the project file associated with this Application Note. These routines are presented as macros to simplify the explanation of the algorithm development. Conversion of these macros to subroutines or to distinct code is left as an exercise for the reader.

16-Bit Signed Multiplication

If two signed, 2-byte operands are multiplied together the result is a signed, 4-byte answer. Figure 2 shows that this multiplication is just the combination of four smaller 8-bit, unsigned multiplications.

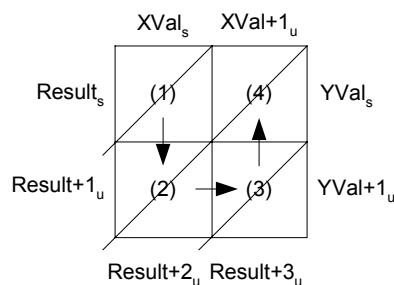


Figure 2: Napier Matrix for 16-Bit Signed Multiply

The following four-step algorithm illustrates four, smaller 8-bit unsigned multiplications:

1. Multiply $XVal_s$ and signed byte $YVal_s$. Place the lower byte resultant stored in $Result+1_u$ and the upper byte stored in $Result_s$.

2. Move down the matrix and multiply $XVal_s$ and $YVal+1_u$. Place the lower resultant byte in $Result+2_u$, and add the upper resultant byte with $Result+1_u$. Because the upper result is signed but the storage is unsigned, the sign must be extended to $Result_s$. By moving only one space horizontally each time, only one multiplicand needs to be reloaded, cutting the overhead of loading the multiplier nearly in half.
3. Move right and multiply $XVal+1_u$ and $YVal+1_u$. Place the lower upper resultant byte in $Result+3_u$ and add the upper resultant byte to $Result+2_u$.
4. Move up the matrix and multiply $XVal+1_u$ and $YVal_s$. Add the lower byte to $Result+2_u$ and the upper byte into $Result+1_u$. As in step 2, a negative result must be sign-extended in to $Result_s$.

The following macro implements this algorithm. Macros internal to it can be found in "signedmath.inc" and "unsignedmath.inc," located in the project file associated with this Application Note.

```
macro Multiply32s_16s_16s
; result = XVal * YVal
; @0 = @1 * @1
; 16bit by 16 bit signed multiply
; with 32 bit signed result
;
; (1)
Multiply16s_8s_8s (@0), (@1), (@2)
; (2)
PushMulY (@2 + 1)
GetXsYuMSB (@1), (@2+1)
cmp A, 128
jc . + 4 ;pass on carry
    dec [@0]
    add [@0 + 1], A
    adc [@0], 0
    GetLSB
    mov [@0 + 2], A
; (3)
PushMulX (@1 + 1)
GetUnsignedMSB (@1+1), (@2 + 1)
    add [@0+2], A
    adc [@0+1], 0
    adc [@0], 0
    GetLSB
    mov [@0+3], A
; (4)
PushMulY (@2)
GetXUsMSB, (@1 + 1), (@2)
    push A
    cmp A, 128
    jc . + 4
        dec [@0]
    GetLSB
    add [@0 + 2], A
    pop A
    adc [@0 + 1], A
    adc [@0], 0
endm
```

Code 1 is a program to exercise this macro. Two 16-bit signed operands ($XVal$ and $YVal$) are multiplied and a 32-bit resultant ($Result$) is calculated.

```
-----
; Program to test the "signedmath.inc" macros
;
; Copyright (c) Cypress Microsystems 2002
; All rights reserved. (every single one)
-----

export _main
include "m8c.inc"
include "unsignedmath.inc"
include "signedmath.inc"
export Result
export XVal
export YVal
area bss(RAM)
    Result:   BLK 4   ;32 bit signed Resultant
    XVal:    BLK 2   ;16 bit signed operand
    YVal:    BLK 2   ;16 bit signed operand
area text(ROM, REL)
_main:
    mov [XVal+0], ffh
    mov [XVal+1], ffh
    mov [YVal+0], ffh
    mov [YVal+1], ffh
    loop:
        Multiply32s_16s_16s Result, XVal, YVal
        nop ;good place to halt and view data
    jmp loop
ret
```

Code 1

When this program is run on the ICE and halted at the end of calculation, the contents of RAM will show that the "Result" is **-1*-1 or 1**. The time it takes to perform the retrieval of data, the actual multiply, and place the signed answer in memory is 293 CPU cycles. For a CPU speed of 24 MHz this works out to 12.2 usec.

Conclusion

A signed multiplier does not limit a user to single byte-signed math. An understanding of the differences between signed and unsigned values makes fast, multi-byte multiplication possible. A two-byte example (**Multiply32s_16s_16s**) has been shown, Table 1 gives a summary of the CPU cycles and size in bytes for the routine developed in this Application Note.

Ideas discussed herein will allow the user to develop their own multiplication routines customized for their particular data-processing requirements.

Table 1: Multiplication Routine Summary

Operands	Result	CPU Cycles	Size
16-Bit Signed	32-Bit Signed	293	99 Bytes

Cypress MicroSystems, Inc.
22027 17th Avenue S.E. Suite 201
Bothell, WA 98021
Phone: 877.751.6100
Fax: 425.939.0999

<http://www.cypressmicro.com/> / http://www.cypress.com/aboutus/sales_locations.cfm / support@cypressmicro.com

Copyright © 2002 Cypress MicroSystems, Inc. All rights reserved.

PSoC™ (Programmable System on Chip) is a trademark of Cypress MicroSystems, Inc.

All other trademarks or registered trademarks referenced herein are property of the respective corporations.

The information contained herein is subject to change without notice.