Fixed-Point Multiplication using the dsPIC Hardware Features: Some Notes and appropriate Assembler Code

Stephan Lehmann
Technical University of Berlin, Wavelet Application Group
stephan.lehmann@tu-berlin.de
August 18, 2008

CONTENTS

I. INTRODUCTION

This document is an addition to the tutorial *Fixed-Point Arithmetic in C: A Tutorial and an Example on Wavelet Filtering* [1] which describes a wavelet transformation algorithm for processors with extremly sparse RAM resources. The tutorial doesn't contain assembly language for flexibility and comprehensibility reasons but due to the sparse resources the algorithm has to do a lot of redundant calculations. A solution to speed up these calculations by using assembly language and some specific features of the dsPIC controller is described here.

II. OPTIMIZATION

The basic computation for the fixed point wavelet transformation is a sequence of three operations. The first step is loading the filter coefficient. The second step is to multiply the pixel value by the filter coeffizient. The last step is fixing the results exponent using a shift. This sequence must be implemented for the high pass filter and the low pass filter respectively.
In the tutorial [1] the first step is implemented in the functions `INT16 AlFix(INT8 index)` and `INT16 AhFix(INT8 index)`. Step two and three is implemented in the macro `MulFixExp`. The compiler doesn't use the dsPIC's DSP- instructions to optimize this code. The assembly code in the appendix merges `INT16 AlFix(INT8 index)` and `MulFixExp` into a new function `MulFixExpLow`.
`INT16 AhFix(INT8 index)` and `MulFixExp` are combined into the new function `MulFixExpHigh`. For information about the instructions used there, see the *dsPIC30F/33F Programmer's Reference Manual* [2]. The new functions perform about 110% faster then their C code equivelants.

REFERENCES

[1] S. Rein, *Fixed-Point Arithmetic in C: A Tutorial and an Example on Wavelet Filtering*, Wavelet Application Group, Technische Universität Berlin, 2008, available at www.mdt.tu-berlin.de/research/wavelets.
[2] R. Filchiero, *dsPIC30F/33F Programmer's Reference Manual*, Microchip, 2005, available at www.microchip.com.

APPENDIX

1) Assembly function MulFixExpLow

```
; This function implements a low pass filter multiply with scaling
;
; extern INT16 MulFixExpLow(INT16,INT16,INT16,INT16,INT16)
;
; parameters:
; filter index        [W0]
; expA                [W1]
; expB                [W2]
; expC                [W3]
; filter input        [W4]
;
; other registers used:
; filter coefficient [w5]
;
; return value        [w0]
;
; instruction words : 25 (75 bytes program memory)
; cycles            : 7+19+3 = 29 for al[0]-al[3]
;                   : 7+17+3 = 27 for al[4]
;
; Annotations       : !!! filter index must be in the range of [-4 .. 4] !!!
;                     !!! this will not be checked !!!
;
; Author            : Stephan Lehmann
; last change       : 22.06.2008
;

.global _MulFixExpLow
_MulFixExpLow:
                BTSC    w0,#15      ; negativ index?
                NEG     w0,w0       ; if yes then abs(index)
                SL      w0,#1,w0    ; w0*=2 for computed branch
                BRA     w0          ; computed branch to filter coeff.
                MOV     #27941,w5   ; load al[0] to w5
                BRA     MULT        ; branch to multiply
                MOV     #12367,w5   ; load al[1] to w5
                BRA     MULT        ; branch to multiply
                MOV     #-3625,w5   ; load al[2] to w5
                BRA     MULT        ; branch to multiply
                MOV     #-781,w5    ; load al[3] to w5
                BRA     MULT        ; branch to multiply
                MOV     #1240,w5    ; load al[4] to w5
MULT:           ADD     w1,w2,w1    ; expA = expA+expB
                SUB     w1,w3,w1    ; expA = expA-expC
                ASR     w1,#1,w2    ; expA/2 -> split shift into two shifts
                SUB     w1,w2,w1    ; expA%2 -> split shift into two shifts
                MOV     CORCON,w3   ; save CORCON register
                BSET    CORCON,#0   ; integer multiply instead of fractional
                MPY     w4*w5,A     ; filter_input*filter_coeff
                SFTAC   A,w1        ; 1st arithmetic shift
                SFTAC   A,w2        ; 2nd arithmetic shift
                MOV     ACCAL,w0    ; lower 16 bit of ACCA to return register
                MOV     w3,CORCON   ; restore CORCON register
                return
.end
```

2) Assembly function MulFixExpHigh

```
; This function implements a high pass filter multiply with scaling
;
; extern INT16 MulFixExpHigh(INT16,INT16,INT16,INT16,INT16)
;
; parameters:
; filter index        [W0]
; expA                [W1]
; expB                [W2]
; expC                [W3]
; filter input        [W4]
;
; other registers used:
; filter coefficient [w5]
;
; return value        [w0]
;
; instruction words : 23 (69 bytes program memory)
; cycles            : 7+19+3 = 29 for ah[0]-ah[2]
;                   : 7+17+3 = 27 for ah[3]
;
; Annotations       : !!! filter index must be in the range of [-3 .. 3] !!!
;                     !!! this will not be checked !!!
;
; Author            : Stephan Lehmann
; last change       : 22.06.2008

.global _MulFixExpHigh
_MulFixExpHigh:
                BTSC    w0,#15      ; negativ index?
                NEG     w0,w0       ; if yes then abs(index)
                SL      w0,#1,w0    ; w0*=2 for computed branch
                BRA     w0          ; computed branch to filter coeff.
                MOV     #25837,w5   ; load ah[0] to w5
                BRA     MULT        ; branch to multiply
                MOV     #-13700,w5  ; load ah[1] to w5
```

```
36                  BRA     MULT        ; branch to multiply
37                  MOV     #-1333,w5   ; load ah[2] to w5
38                  BRA     MULT        ; branch to multiply
39                  MOV     #2115,w5    ; load ah[3] to w5
40  MULT:           ADD     w1,w2,w1    ; expA = expA+expB
41                  SUB     w1,w3,w1    ; expA = expA-expC
42                  ASR     w1,#1,w2    ; expA/2 -> split shift into two shifts
43                  SUB     w1,w2,w1    ; expA%2 -> split shift into two shifts
44                  MOV     CORCON,w3   ; save CORCON register
45                  BSET    CORCON,#0   ; integer multiply instead of fractional
46                  MPY     w4*w5,A     ; filter_input*filter_coeff
47                  SFTAC   A,w1        ; 1st arithmetic shift
48                  SFTAC   A,w2        ; 2nd arithmetic shift
49                  MOV     ACCAL,w0    ; lower 16 bit of ACCA to return register
50                  MOV     w3,CORCON   ; restore CORCON register
51                  return
52  .end
```