FIR Filtering on ARM Cortex-M3

ALEXANDRU BÂRLEANU, VADIM BĂITOIU, ANDREI STAN
Faculty of Automatic Control and Computer Engineering
Gheorghe Asachi University of Iași
700050, Romania
alexb@cs.tuiasi.ro, vadim.baitoiu@gmail.com, andreis@cs.tuiasi.ro

Abstract: - This paper describes how to implement efficient FIR filters on ARM Cortex-M3 microprocessors. Two multiply-accumulate instructions are discussed: MLA – multiply with accumulate, using 32-bit operands and producing a 32-bit result, and SMLAL – signed multiply with accumulate, using 32-bit operands and producing a 64-bit result. There are advantages and disadvantages for each instruction. It is possible, for example, to reuse the data loaded in memory and compute multiple output samples per iteration. MLA is more suitable for this technique than SMLAL. But SMLAL can provide higher accuracy than MLA. This paper gives also an insight on how to increase the filtering accuracy with non-uniform coefficient fractional wordlength. This can be useful if there are very small and very large floating-point constant coefficients.

Key-Words: - embedded software, fixed-point arithmetic, filtering algorithms, ARM Cortex-M3

1 Introduction

ARM Cortex-M3 is a modern, high-performance (32-bit) processor designed for embedded applications ("M" comes from microcontroller). Though it is not primarily a DSP processor, it has multiply-accumulate instructions that are very convenient for signal processing algorithms [1]. Two of them are discussed in this paper: MLA – multiply with accumulate, using 32-bit operands and producing a 32-bit result, and SMLAL – signed multiply with accumulate, using 32-bit operands and producing a 64-bit result. These instructions are not something specific only to the ARM Cortex-M3 processor. MLA was introduced in the ARMv2 architecture and SMLAL was introduced in the ARMv3M architecture. (Cortex-M3 is based on the ARMv6M architecture.)

ARM Cortex-M3 processors do not have floating-point coprocessors, which means that C compilers must emulate floating-point operations. This can be a problem, because even if the compiler libraries for floating-point are highly-optimized, for standard IEEE-754 accuracy results the execution time can be very long. In such cases it is important to consider if the calculations can be done in fixed-point format [2,3]. That would mean using integer datatypes and operations instead of floating-point. It is possible to download libraries available on the Internet [4,5,6] or write custom routines (which can be made very efficient for a particular problem).

2 Existing Libraries

NXP Semiconductors and STMicroelectronics provide DSP libraries for ARM Cortex-M3 microprocessors [4,5]. Both libraries contain FIR filtering routines. There are some minor differences between the two implementations like, for example, the format of input data, but in essence the implementations are the same.

ARM Cortex-M3 processors cannot load from memory and perform arithmetic operations in parallel. This aspect was taken into account in the NXP and STM FIR filtering routines. In both cases the signal is filtered in blocks, in order to minimize the number of load and store operations (the samples and coefficients read from memory are reused). This technique is described in [7]. But there are some disadvantages. The signal to be filtered is processed entirely: if M is the length of the filter and N is the length of the input signal then in one call M by N multiply-accumulate operations are performed. The number of coefficients and the number of input/output samples must be multiples of 4 for MLA or multiples of 3 or 2 for SMLAL. And the MLA instruction produces 32-bit results – this is fine for applications in which the degree of accuracy needed is not very high. But if, for example, a filter with 24-bit coefficients is wanted, then MLA is insufficient. Switching from MLA to SMLAL is not very easy, because with SMLAL the register pressure is increased and the signal must be filtered in smaller blocks.

The NXP FIR filtering routine has the following C declaration:

```
void  vF_dspl_blockfir32(int  *pi_y,  int  *pi_x,
tS_blockfir32_Coeff *pS_Coeff, int i_nsamples);

typedef struct
{
    int *pi_Coeff;
    int NTaps;
} tS_blockfir32_Coeff;
```

The input signal and coefficients are declared as 32-bit signed integers, but their values must be limited to a much smaller range than full 32-bit integer range.

The STM FIR filtering routine has the following C declaration:

```
void   fir_16by16_stm32(int   *a,short   *x,struct
COEFS *p,unsigned int N);

typedef struct {
    short *h;
    unsigned int nh;
} COEFS;
```

The input signal and coefficients are declared as 16-bit signed integers, which eliminates the possibility of multiplication overflow. The summation overflow is however possible, so the signal and/or the coefficients must be properly scaled.

As an aside, the CMSIS 2.0 library provided by ARM [6,9] contains FIR filtering routines too. This library is written entirely in C for Cortex-M3 and Cortex-M4. There are two types of FIR filtering routines: **normal** (with SMLAL) and **fast** (with MLA). The source code does not tell anything directly about MLA or SMLAL; nevertheless, it is understandable what the compiler will do. The size of integer variables and explicit casts that appear in expressions are forcing either MLA or SMLAL.

3 Coefficient Truncation with MLA and SMLAL

To understand how much the coefficients of a FIR filter must be truncated for MLA and SMLAL instructions, below is presented an example: a 32-tap filter, designed with Matlab FDATool, and a 12-bit input signal (unsigned). The filter coefficients are scaled *uniformly*.

3.1 MLA - multiply with accumulate, using 32-bit operands and producing a 32-bit result

Fig. 1 shows that with MLA the filter coefficients must be scaled in the integer interval [-16388; 16388]. This means that maximum 14 bits can be kept from the floating-point representation significand.

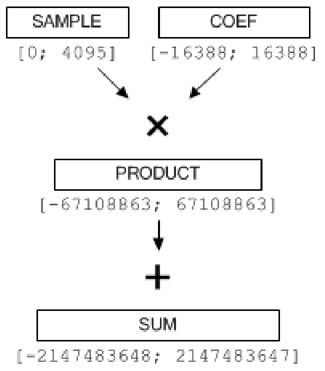

Figure 1: Finding the permitted interval of coefficient fixed-point values

In Fig. 1 the intervals are computed from bottom (SUM) to top (COEF). The SUM interval is set as the maximum signed 32-bit integer interval. The PRODUCT interval is computed by dividing the SUM interval low/high values by 32 (number of filter coefficients).

In Fig. 1 the COEF interval is computed for the worst case, but in practice it can be two or more times wider, without the risk of overflow, because most coefficients have small amplitude. In [7] are given two inequalities for determining the accumulator (SUM) precision from the power of input signal and the power of filter coefficients.

In general, if the filter order is great, then it will probably contain coefficients with very small amplitude. Such coefficients are almost lost as the fixed-point coefficient resolution is decreased. Fig. 2 illustrates this.

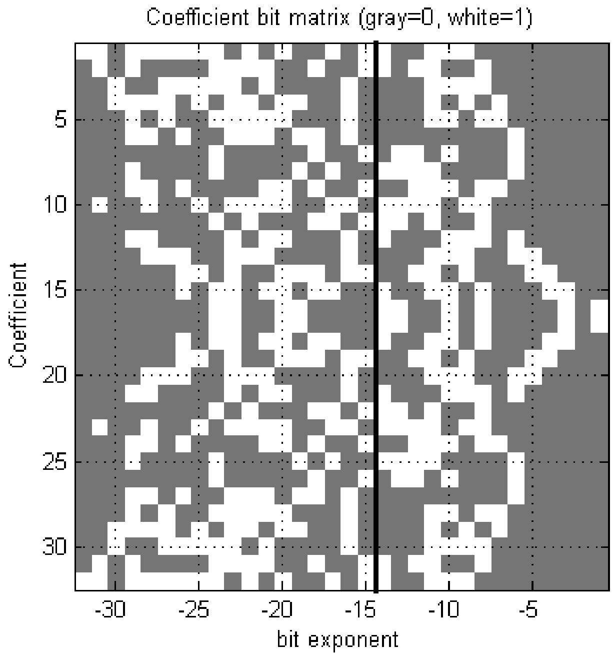

Figure 2: Coefficient bit matrix

Fig. 2 is constructed by extracting the significand bits from single precision floating-point coefficients. A row represents a coefficient. Each row is displaced horizontally for alignment (as for summation). White squares represent „1” bits. Most significant bits are located in the right hand side of the image. The black vertical line that separates bit exponent -15 from bit exponent -14 represents the truncation margin computed as shown in Fig. 1. What is located on the left of this line is lost by truncation.

In Fig. 2 it can be observed the symmetry of coefficients. This property can be used to reduce somehow the number of memory accesses and multiplications, but here this possibility is not exploited.

In Fig. 1 the resolution of the signal to be processed is12 bits – as if from a typical Cortex-M3 analog to digital converter. But if the resolution would be greater, then the coefficients would have to “step back” – for every additional bit in the input signal the fractional wordlength of the coefficients must be decreased.

3.2 SMLAL - signed multiply with accumulate, using 32-bit operands and producing a 64-bit result

The FIR filter coefficients can use the full 32-bit signed integer range ([-2147483648; 2147483647]). The multiplication results cannot overflow. The summation results cannot overflow too, because the signal to be filtered has a resolution of 12 bits. It is interesting that with SMLAL the precision of coefficients can be greater than in the floating-point single precision format. This is easy to see: for SMLAL a coefficient can have up to 31 significant bits and the floating-point single precision format contains 24 significant bits (1 hidden/implicit).

Below is given a simple FIR filtering routine that uses the SMLAL instruction. It is computing one output sample per call (unlike the NXP and STM FIR filtering routines). The ATPCS [8] calling convention is used. There is a loop inside the routine that contains 4 SMLAL instructions one after another – the purpose is to have less loop break tests. This puts a condition on the filter length, but is not a big problem. If the filter has, for example, 41 coefficients, then the solution is to add one more SMLAL instructions after the loop body.

Listing 1 (GCC assembly):

```
# C declaration:  int64_t fir64(const int32_t* 
a,
#       int32_t* x, uint32_t n);
# Parameters:
#     r0: coef ptr
#     r1: input ptr
#     r2: filter length (must be 4, 8,
#                          a multiple of 4)

fir64:
    push {r4-r11}

    mov r12, r2

    mov r3, #0
    mov r2, #0

fir64_loop_start:
    ldmia r0!, {r4-r7}
    ldmia r1!, {r8-r11}

    smlal r2, r3, r4, r8
    smlal r2, r3, r5, r9
    smlal r2, r3, r6, r10
    smlal r2, r3, r7, r11

    subs r12, r12, #4
    bgt fir64_loop_start

    mov r0, r2
    mov r1, r3

    pop {r4-r11}

    bx lr
```

Additional FIR filter implementations with SMLAL can be found in [7].

4 Speed and Accuracy Measurements

A LPC1343 microcontroller is used for speed measurements. The test application is compiled with GCC with -O2 optimizations (the filtering routines are written in assembly, but the routines calling them are written in C). The filter coefficients are placed in SRAM (not in Flash).

An ARM Cortex-M3 processor can have a so-called Data Watchpoint and Trace (DWT) unit. This is something optional – not available on all processors (on LPC1343 yes). A DWT unit contains a 32-bit cycle counter register, named CYCCNT, which can be used for time measurements. The following code shows how to read the CYCCNT register:

```
uint32_t cyccnt = *((uint32_t*)0xE0001004);
```

Table 1. Performance results obtained with a 32-tap filter

Code	CYCLECOUNT	SQNR (dB)
Implementation with MLA instruction (STM DSP library)	3936 (one call)	75.65
Implementation with SMLAL instruction (Listing 1)	10323 (32 calls)	177.00
Single-precision floating-point expression	175540-183764 (32 calls)	156.48

The signal to quantization noise ratios (SQNR values) in Table 1 are computed on a Intel T7250 processor (using equivalent C code – MLA and SMLAL cannot be used) with the following formula:

$$SQNR = 10 \cdot \log_{10}\left(\frac{S}{N}\right) \quad (1)$$

where S represents the power of the ideal signal (double precision floating-point code) and E represents the power of the error – which is computed as the difference between the ideal and the actual signal. The accuracy can also be estimated analytically [10].

The input data used to test the filtering routines influences the execution time. This might seem like a paradox at first sight; because there is no if-then-else branching (always the same number of instructions is executed). The explanation lies in the SMLAL instruction timing [1] – it can take from 4 to 7 cycles, depending on operands (this is called early termination). The cycles shown in Table 1 are obtained with random input data.

The routine in that uses the SMLAL instruction (see Listing 1) takes 2.6 times more time than the STM FIR routine. On the other hand, it provides very high accuracy – greater than even the equivalent single precision floating-point code. The routine with the SMLAL instruction takes approximately 17 times less time than the equivalent single-precision floating-point code.

The CYCLECOUNT values shown in Table 1 correspond to filter coefficients placed in SRAM memory. It is possible to place them also in the Flash memory and save SRAM, but this will slightly decrease the execution speed. For example, the routine from Listing 1 executes with SRAM coefficients in 10323 cycles, but with Flash coefficients in 11571 cycles (12% more). The number of cycles obtained to access the Flash memory is the same for different system frequency values: 9, 18, 36, and 72MHz.

The accuracy of the STM routine that uses the MLA instruction (in Table 1: 75.65dB) is obtained with coefficients scaled as shown in Fig. 1 and Fig. 2. It is possible to increase the fractional wordlength and obtain a higher SQNR, but only by one bit. More than one bit is not possible, because the STM FIR filtering routine takes `short int` coefficients and the greatest coefficients would overflow. The NXP FIR filtering routine takes `int` coefficients.

NOTE: There are two forms of floating-point code for signal filtering: as an expression (which can be very long if there are many coefficients) or as a loop (one multiplication and one summation per iteration). It has been observed that the accuracy of a loop is much lower than the accuracy of an expression – though in both cases the coefficients are identical (single-precision floating point). This accuracy difference might be due to accumulator width. In a loop the summations are performed consecutively using a single precision floating-point accumulator (declared as a `float`). But in an expression the compiler is probably using another type of accumulator (maybe a 32-bit integer). This accuracy anomaly has been observed with two compilers: GCC and IAR.

5 Non-uniform Coefficient Wordlength

In case there are very small and very large floating-point coefficients it might be better, if possible, to group them by exponent, in order to increase the accuracy [11]. (Fixed-point constants must have the same fractional wordlength within a group.) In this way more coefficient information is used for multiplications, but the multiplication results must be aligned for summations. The cost of aligning is not very big, because ARM processors have 32-bit barrel shifters. A shift operation takes one cycle no matter the shift length.

Fig. 3 illustrates the key positions of shift operations in a filter tap.

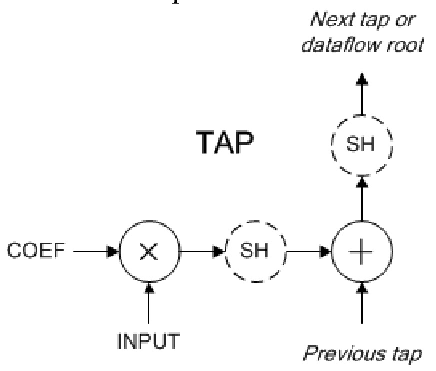

Figure 3: There can be two bitwise shift operations per filter tap at runtime (SH nodes)

When shifting integers for alignment to right it must be considered that for negative values this is not equivalent with division [12]. If the filter input data is non-negative, then one solution to this problem is to change the sign of negative coefficients, perform multiplications, shift operations, and then one or more substractions instead of additions.

6 Conclusion

The main idea is that with the MLA instruction it is possible to write high speed code and with the SMLAL instruction it is possible to write high accuracy code. A routine based on the SMLAL instruction is slower than a routine based on the

MLA instruction, because (1) SMLAL takes more cycles than MLA and (2) the result of SMLAL is 64-bit and the result of MLA is 32-bit – which makes MLA more convenient than SMLAL for block filter implementations [7]. The difference in number of cycles is great, but the accuracy that can be obtained should be regarded too when deciding which instruction to use. The SMLAL instruction is ideal for high accuracy signal processing – as has been shown, it can be better than equivalent single precision floating-point C code. The accuracy with MLA is not so high. It can be good enough in many applications like, for example, to filter 10-bit or 12-bit ADC data. However, it is easy to observe that as the dynamic range of the input samples grows, the filter coefficients must be downscaled.

Control algorithms, though not every time non-recursive (FIR), deserve a brief discussion. If there are not so many coefficients, then the code can be fully unrolled. By doing this, the routine becomes simpler and the execution time is reduced. If there are integrators whose values can become very large, then SMLAL might be the only usable instruction. And if the control command must be computed at each step, then there is no way for block filter implementations.

References:

[1] ARM, Cortex-M3 Technical Reference Manual, 2010.

[2] D. Menard, D. Chillet, F. Charot, O. Sentieys, Automatic Floating-point to Fixed-point Conversion for DSP Code Generation, *in Proc. of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Oct. 2002.

[3] I. Konvalinka, A. Quddus and D. Asraf, An efficient floating-point to fixed-point conversion process for biometric algorithm on DaVinci DSP architecture, *Proc. SPIE 7306*, 73062A, 2009.

[4] NXP Semiconductors, AN10913 - DSP library for LPC1700 and LPC1300, 2010.

[5] STMicroelectronics, UM0585 - STM32F10x DSP library, 2010.

[6] ARM Press Release, ARM Extends Software Interface Standard with DSP Library, 13 December 2010.

[7] Andrew N. Sloss, Dominic Symes, Chris Wright, ARM System Developer's Guide, Elsevier, 2004.

[8] ARM, The ARM-THUMB Procedure Call Standard, 2009.

[9] Reinhard Keil, Digital Signal Processing with Cortex™-M Microcontrollers, *Information Quarterly Magazine*.

[10] D. Menard, R. Rocher, O. Sentieys, Analytical Fixed-Point Accuracy Evaluation in Linear Time-Invariant Systems, *in IEEE Trans. On Circuits and Systems I*, vol. 55, issue 10, pp. 3197-308, 2008.

[11] Bolton, G. and Stewart, R.W., Non-uniform wordlength delay lines for FIR filters, *EUSIPCO*, 2009, Glasgow.

[12] R. J. Mitchell and P.R. Minchinton, A Note on Dividing Integers by Two, *The Computer Journal*, 32, No. 4, Aug 1989, 380.