

Optimization of FIR filter implementation for FMT on VLIW DSP

Sachit Vashisht, Nikhil Chhabra, Tushar Tanwar
Dronacharya college of engineering, Gurgaon

Abstract- The paper summarizes the FMT modulation prototype filter design and its efficient implementation on DSP. The optimum design of algorithms for digital signal processors with VLIW architecture is described. Using this new approach it was, for example, possible to optimize compilation from the C language into the assembler of TMS320C6414 digital signal processor for implementation of FMT modulation with prototype FIR filter. The method consists in a closer linkage between the theory of digital signal processing, software tools and hardware.

Index Terms- Assembler Programming, Digital Signal Processor, Filtered Multi Tone Modulation, Prototype Filter, State-Space Representation, Very Long Instruction Word.

I. INTRODUCTION

The filtered multitone modulation (FMT) is a multitone modulation technique, proposed by Giovanni Cherubini in 1999. Another type of this modulation is the half overlapped FMT. It is an interesting alternative to standard modulations in xDSL systems, particularly to digital multitone (DMT) modulation, as well as to wireless systems using orthogonal frequency division multiplexing (OFDM) modulations. This modulation is using the fast fourier transform (FFT) algorithm in combination with bank of filters for frequency spectrum separation. These filters are polyphase components of prototype filter, low-pass finite impulse response (FIR) filter. The first step of implementation of this modulation on digital signal processor is an implementation of these filters. The advantages of the modern architecture of type very long instruction word (VLIW) digital signal processors cannot be made full use of as long as the algorithms to be implemented require sequential data processing. Fortunately, this type of algorithms appears in the area of digital signal processing only rarely. Much more frequent are the algorithms of

processing data flows (FFT, digital filter banks, wavelet and homomorphous analyses, etc.). In this type of processing the algorithm can be realized for several input signal values simultaneously, and making use of parallel processing will considerably increase the algorithm processing speed and the computation performance.

II. PROTOTYPE FILTER DESIGN

The fundamental element of the FMT system is the prototype filter. We typically attempt to design it to reach the best frequency characteristic. It mainly concerns the suppression of side lobes, the orthogonality of derived filters, and the frequency separation of particular subchannels. In this way we can obtain an ideal suppression of inter-channel interference (ICI). Inter-symbol interference (ISI), on the other hand, will not be limited by these actions; it will appear even in the ideal channel without interference. Its source is the implementation of filters. But such a distortion is easy to remove via equalization.

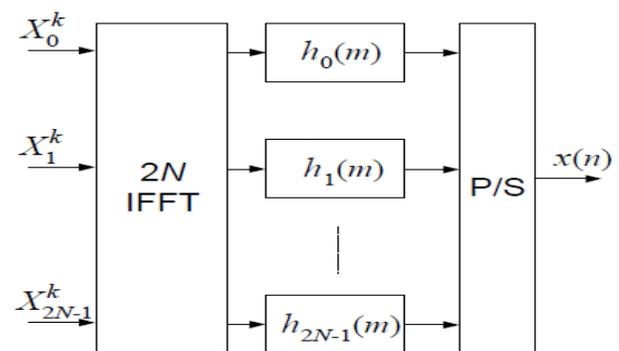


Fig. 1 Block diagram of FMT modulator and transmitting filters.

The prototype filter can be designed using any FIR filter design method with the limitation that the filters derived must be orthogonal to each other. The following methods meeting the orthogonality condition appear the most convenient for the design.

The first method is approximation of IIR filters, where ρ is the main parameter. Parameter ρ controls the shape of filter transition bandwidth and it is within the range from 0 to 1. For greater ρ the final filter is of a greater steepness but it decreases the stop-band attenuation. If we increase ρ above 0.6, the ripple in the pass-band will increase significantly. For $\rho = 0.9$ it can be up to 12 dB. Side lobes are suppressed to -63 dB for $\rho = 0.1$. The other option is to shape the filter transition bandwidth using the square-root raised cosine filter. In this filter, the lobes are suppressed to -38 dB for $\alpha = 0.5$. The third method is the windows method. The final characteristic is then formed by the properties of the window used. The Blackman, Hamming or Hanning windows appear to be sufficient, possibly also some other window retaining the orthogonality with sufficient side lobe fall-off and broadness of the main lobe. Another method uses the modified Parks-McClellan algorithm.

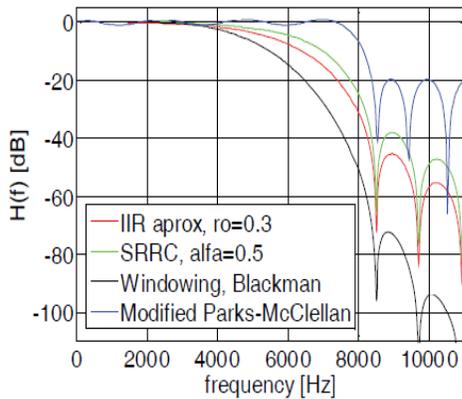


Fig. 2 Prototype filters frequency responses using different design methods.

III. DIGITAL SIGNAL PROCESSORS TMS320C6414

The digital signal processor TMS320C6414 by Texas Instruments belongs to digital signal processors with VelociTI core based on VLIW architecture. The core of VelociTI digital signal processors is made up of two data paths (A and B), each of which contains four functional units (L, S, M and D) and a data register file (Fig 3). The registers are of 32 bits and there 32 of them in the TMS320C64xx processor. The registers are always designated by a data path letter and the respective serial number 0-31 (for example, A0-A31).

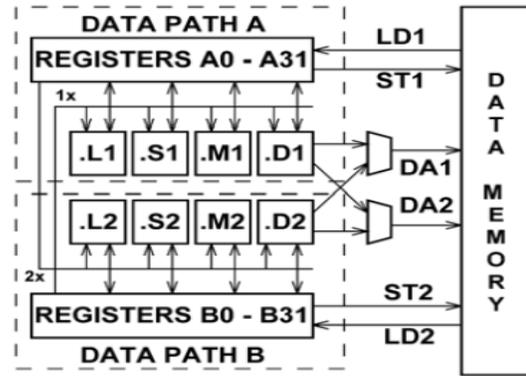


Fig. 3 Block diagram of the VelociTI core of DSP of TMS320C64xx series.

Functional units can read values from the registers or store the results of operations in the registers of the corresponding data path. In each data path in an instruction packet it is possible to read the contents of one register of the other data path. Reading is then realized along one of the two cross paths (1x and 2x). In each data path one address (DA) and two data buses (LD and ST) are available for moving the values between the registers and the data memory. The functional units of the core of digital signal processor are optimized for a certain type of operation. Functional unit L is designed for arithmetic operations, functional unit S processes the instructions of logic operations and program branching instructions, functional unit M is a hardware multiplier, and functional unit D is used to calculate the address and to transfer values between the data memory and the registers.

IV. IMPLEMENTATION OF DISCRETE CONVOLUTION

FIR filters will be implemented using algorithm of discrete convolution (1) of input signal $x[n]$ and filter coefficients $h[n]$ or samples of impulse response, respectively.

$$y[n] = h[n] * x[n] = \sum_{m=0}^{N-1} h[m]x[n - m], \tag{1}$$

```

void fir_filter(
    short in[],
    short coef[],
    short out[],
    int N_out,
    int N_h)

```

Fig. 4 Declaring the fir_filter function in the CCS.

where N is the number of coefficients or impulse response length, respectively. A function that will realize a discrete convolution can be declared in the Code Composer Studio (CCS) integrated development environment (IDE) for TMS320C6xxx digital signal processor as shown in Fig. 4. This function takes over as parameters the array in of input samples, the array coef of filter coefficients, and the array out for output samples storage. The number of output samples is read as parameter N_out and the number of filter coefficients is read as parameter N_h, respectively.

V. OPTIMIZATION OF DISCRETE CONVOLUTION

Compilers designed for digital signal processors are part of the IDE. Texas Instruments' CCS can be quoted as examples. These compilers differ from the ANSI-C or C++ standard in a few details, which in the ultimate result have a considerable effect on the speed and stability of algorithm implementation. The basic difference lies in that the defined data types are fully adapted to the architecture of digital signal processor. The number of data bits and the format of storing numbers in a given code (mostly the two's complement) correspond to the actual storage of numbers in digital signal processor registers. When optimizing the source code it is convenient either to enter the instructions of digital signal processor assembler directly into the C-language source code or to use the *intrinsic* functions, which are assembled as a single instruction. In Fig. 5, the intrinsic function `_smpy` will be compiled as `SMPY` instruction; function `_sadd` will be compiled as `SADD` instruction. Most IDEs for digital signal processors support these activities. In this way the programmer can optimize the critical parts of source code that the assembler is not able to analyse correctly. This is a kind of intermediate stage between optimizing in the

C language and optimizing in the assembler of digital signal processor.

The CCS defines macro instructions and compiler directives by means of which the programmer defines in the source code additional information. The data in question concern, for example, mutual relations between variables, rounding of values in memories, etc. This set-up information is used by the compiler in the optimization process and if used properly, this information can greatly increase the compilation effectiveness as measured by the computation demand of the compiled binary code. Conversely, incorrect application yields a binary machine code, which is potentially dangerous and can cause run-time errors. For example, two independent variables X and Y , stored in different parts of data memory can be stored or read in parallel. If the variables shared a common memory space, then writing a value in variable X would entail a change also in the value of variable Y . In that case the value read from Y depends on whether the reading operation is executed before or after the operation of writing into X . In the case that variables X and Y are the arguments of a function passed on by a reference, it is not possible at the time of compilation to find out whether or not the two variables share the memory space. The compiler assumes they do and provides a more secure binary machine code, which, however, requires longer and more intensive computation.

```

void fir_filter(
    const short in[restrict],
    const short coef[restrict],
    short out[restrict],
    const int N_out,
    const int N_h)

```

Fig. 6 Optimized declaration of the fir_filter function in the CCS.

Individual instructions of the instruction packet do not execute different stages of the same sum element but different stages of three elements are performed simultaneously due to the delay slot of instruction execution. This is the result of pipelining and parallel processing of the loop iterations. For the element with serial number $m = 7$ reading from the memory is started, for the element $m = 2$ the multiplication of the coefficient and the sample is performed, and

finally the element $m = 0$ is accumulated. The execution of the instruction packet of the loop kernel is illustrated in Fig. 7.

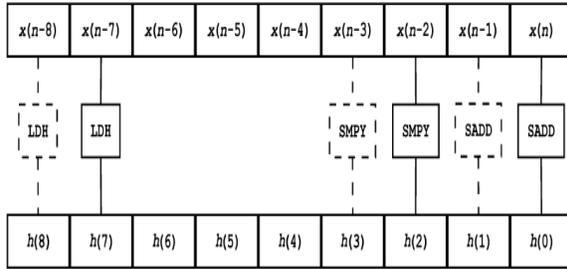


Fig. 7 Illustration of the calculation of discrete convolution in VelociTI digital signal processor.

VI. CONCLUSION

Non-optimised and optimised versions of function `fir_filter` were tested in the CCS environment. The filter for testing was a pass-band filter of the 39th order. The compiled binary code size and the computation demands of both versions are shown in Table I. The computation demands of the optimised version per one output sample is approximately three times less than that of the non-optimised version, but the binary code size is approximately three times greater than the binary code size of the non-optimised version. This is caused by the addition of special program sections, i.e. *prolog* and *epilog*. In the course of optimization we must compromise between the calculation demands and the binary code size of compiled binary code.

TABLE I
AVERAGE COMPUTATION DEMANDS PER ONE SAMPLE AND BINARY CODE SIZE
OF FUNCTION `FIR_FILTER` FOR 39TH FILTER ORDER.

<code>fir_filter</code>	clock cycles (-)	code size (B)
non-optimised	412	448
optimised	125	1 712

Writing algorithms in the assembler of digital signal processors of the type VLIW is very demanding. Several instructions are being processed in every clock cycle, their number being given by the number of active parallel units. Executing any instruction

takes a different number of clock cycles. This is due to the high degree of pipelining. The program thus contains several parallel computation paths, which the programmer must follow incessantly. Under these conditions it is very easy to make a mistake. All this strongly depends on the particular type of digital signal processor. By contrast, the development of programs for processors with superscalar architecture (Pentium from Intel, etc.) is simpler from this viewpoint since parallel instruction grouping is performed by the hardware unit in the processor structure (Schedule Unit). In spite of the above difficulties we often cannot avoid writing the algorithm directly in the assembler of digital signal processor since this is the only way how to achieve the maximum speed of calculating the critical parts of the source code.

REFERENCES

[1] J. Eyre, The Digital Signal Processing Derby, in *IEEE Spectrum*.
 [2] Z. Smékal and P. Sysel, Architecture-Dependent Algorithm Optimization for VLIW Digital Signal Processor.
 [3] P. Sysel, Optimization of FIR Filter on VLIW Digital Signal Processors, in *Proc. of Research in Telecommunication Technology*.
 [4] N. Benvenuto, S. Tomasin, L. Tomba, Equalization methods in DMT and FMT Systems for Broadband Wireless Communications. In *IEEE Transactions on Communications*