Optimization of FIR filter implementation for FMT on VLIW DSP

Petr Sysel, Ondřej Krajsa

***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.

***Keywords*—**Assembler Programming, Digital Signal Processor, Filtered MultiTone 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 [9]. Another type of this modulation is the half-overlapped FMT [11]. 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 (ODFM) 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. This is to say that individual operations are linked up directly and prior to any next step in the algorithm the results of all the previous operations must be known. 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 [2]-[3].

Manuscript received June 31, 2010. This work was supported in part by the Ministry of Education, Youth, and Sports of Czech Republic under research programme MSM 0021630513 and also by the Grant Agency of Czech Republic under Grant 102/09/1846.

P. Sysel is with the Brno University of Technology, Faculty of Electrical Engineering and Communication, Purkynova 118, Brno, 602 00, Czech Republic; e-mail: sysel@feec.vutbr.cz.

O. Krajsa is with the Brno University of Technology, Faculty of Electrical Engineering and Communication, Purkynova 118, Brno, 602 00, Czech Republic; email: krajsao@feec.vutbr.cz.

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 [10]. But such a distortion is easy to remove via equalization.

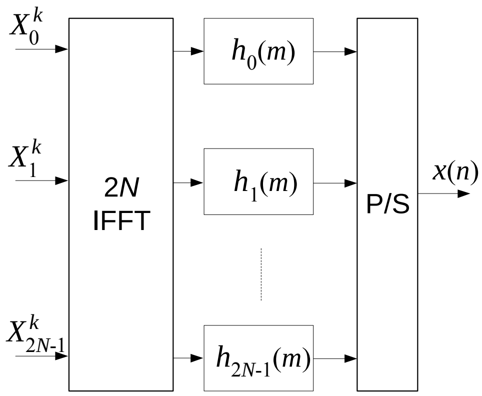

Fig. 1 Block diagram of FMT modulator and transmitting filters.

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 [9], 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 [9]. 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.

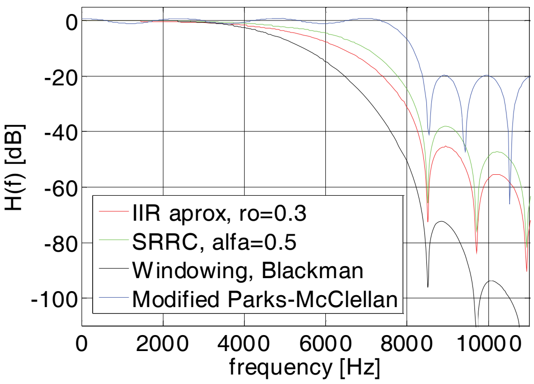

Fig. 2 Prototype filters frequency responses using different design methods.

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`).

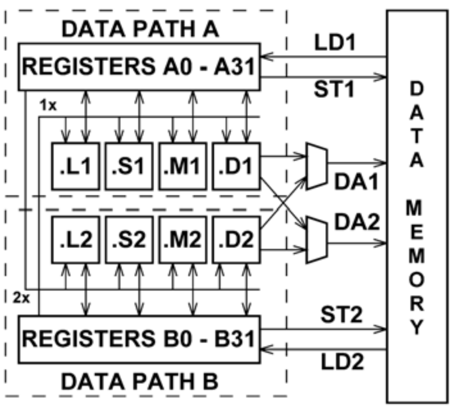

Fig. 3 Block diagram of the VelociTI core of DSP of TMS320C64xx series.

The instruction notation in the assembly is made up of the instruction mnemonic, the specification of the unit the instruction is designed for, and the definition of instruction operands, i.e. source and target registers or direct data. As many as eight instructions can be composed in parallel in one instruction packet. The `||` characters signify that an instruction is to execute in parallel with the previous instruction in one *instruction packet*. Example of instruction mnemonic is shown in Fig. 7.

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.

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

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 parametr `N_h`, respectively.

```
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.

The equation (1) of discrete convolution can be programmed as shown in Fig. 5. The variable `sum1` is an auxiliary variable of the type of `int` for temporary storage of

ISSN: 1792-4324

ISBN: 978-960-474-208-0

the output sample with double precision. Expressions `_sadd` and `_smpy` are *intrinsic* compiler functions of environment CCS. The first function, `_sadd`, realize addition with saturation of two fractional numbers in the two's complement format, the second function, `_smpy`, realize multiplication with saturation of two fraction numbers in the two's complement format. The arguments of both functions are assumed to be of the `int` type (i.e. 32-bit arguments). However, the multiply operation performs only the product of two 16-bit arguments (16x16 bits). The function `_smpy` multiplies 16 least significant bits of both arguments. Similarly, there are further intrinsic functions, `_smpylh`, that multiplies 16 least significant bits of the first argument by 16 most significant bits of the second argument, `_smpyhl` and `_smpyh`, that realize the product of the remaining parts. To conclude the calculation, 16 most significant bits of the output sample are written in the `out` address in the memory.

```
int sum1;

for( i = 0; i < N_out; i++)
{
 sum1 = 0;
 for( j = 0; j < N_h; j++)
 {
  sum1 = _sadd( sum1,
  _smpy( coef[j], in[i+j]));
 }

 *out++ = sum1 >> 16;
}
```

Fig. 5 Defining the `fir_filter` function while making use of the intrinsics.

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.

In parallel processing the given algorithm can be realized simultaneously for several values of the input signal. Using parallel processing will greatly increase the speed of algorithm processing. The condition is that the algorithm should be written by an experienced programmer directly in the assembler of digital signal processor or that the source code written in a higher programming language (e.g. ANSI-C or C++, etc.) should be translated by a first-rate compiler.

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.

The input sample pointer `in` and the pointer to the field of FIR coefficients `coef` can be declared by the key word `const` since in the course of calculation the input sample value and the values of individual filter coefficients will not change. The output sample value and the values of state-space variables will, on the contrary, change during calculation and thus they cannot be declared by the key word `const`. It is obvious from the algorithm structure that the individual input arguments represent mutually independent data structures, which will be stored in separate memory locations. In that case it is of advantage to use the key word `restrict`, which informs the compiler about the memory-independence of the variables. In case the output sample was entered into the same memory field as the input samples (in-place processing), there would evidently be a dependence relation between the `in` and `out` pointers and the `restrict` keyword could not be used in declaring the two arguments. Optimized declaration of function `fir_filter` is in Fig. 6.

After implementation of these optimizations the calculation of each element of sum (1) is splitted into several separate stages, which will be executed in parallel. The loop kernel is formed by one instruction packet given in Fig. 7. In the first stage it is necessary to read the value of coefficient $h[m]$ and the value of sample $x[n–m]$ from the memory and store it data registers A3 and B3, respectively (instructions LDH). Address of the input sample and address of the coefficient are stored in the register A4 and B4, respectively. In the next stage, instruction SMPY is used to multiply the sample and the corresponding coefficient. In the end, instruction SADD accumulates the value of the product in register A6. The remaining BDEC instruction for program branching provides for the whole instruction packet to be repeated [6].

```
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. 8.

```
LOOP    LDH    .D1T1    *A4++,A3
||      LDH    .D2T2    *B4++,B3
||      SMPY   .M1x     B3,A3,A5
||      SADD   .L1      A5,A6,A6
||      BDEC   .S2      LOOP,B0
```

Fig. 7 Loop kernel instruction packet of discrete convolution implementation on VelociTI digital signal processor.

For the next loop kernel iteration the value of m is increased by one and the stages of the following sum elements are processed. Before entering the loop kernel it is necessary to start, sufficiently in advance, to read successively the values of samples $x[n]$ to $x[n–6]$ and the values of coefficients $h[0]$ to $h[6]$ in order that they be prepared before entering the loop kernel. This must be ensured by the special section of program called *prolog*. Similarly, after leaving the loop kernel it is necessary to finish correctly the processing of the last samples as reading them from the memory is successively finished. This must be ensured by the special section of program called *epilog*.

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.

fir_filter	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. Moreover, grouping instructions into parallel paths is subject to many constraints, which are given by the internal architecture of the given digital signal processor. For example, if only two address buses are available, then only two values can be read from the memory in one clock cycle. All this strongly depends on the particular type of digital signal processor. By contrast,

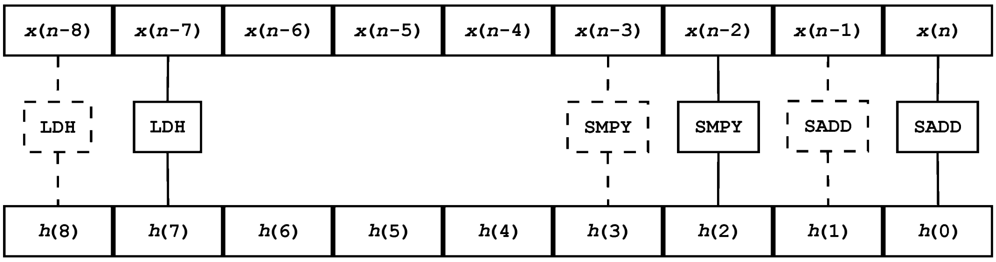

Fig. 8 Illustration of the calculation of discrete convolution in VelociTI digital signal processor.

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*, June 2001, pp. 62-68.

[2] Z. Smékal and P. Sysel, Architecture-Dependent Algorithm Optimization for VLIW Digital Signal Processor, in *Proc. of the 48th International Scientific Colloquium IWK'2003*, Ilmenau, 2003, pp. 143-144. ISBN 1619-4098.

[3] Z. Smékal and P. Sysel, Influence of Signal Processor Architecture on Generating Optimum Algorithm of Digital Signal Processing Methods, in *Proc. of the 5th WSEW/IEEE Multiconference CSCC*. Rethymno, 2001, pp. 4291-4294. ISBN 960-8052-33-5

[4] Z. Smékal and R. Vích, Optimised Models of IIR Digital Filters for Fixed-Point Digital Signal Processor, in *Proc. of the 6th IEEE International Conference on Electronics, Circuits and Systems (ICECS'99)*, September 5-8, 1999, Cyprus, pp. 145-148. ISBN 0-7803-5682-9

[5] Z. Smékal and R. Vích, Cepstral Speech Synthesis Optimised for Dual Harvard Architecture of DSP, in *Proc. of the International Conference on Telecommunications (ICT 2000)*, May 2000, Acapulco, Mexico, pp. 244-248. ISBN 968-36-7762-2

[6] P. Sysel, Optimization of FIR Filter on VLIW Digital Signal Processors, in *Proc. of Research in Telecommunication Technology*, Prague, 2004, pp. 1-5. ISBN 80-01-03063-6

[7] R. Vích, J. Přibil, and Z. Smékal, New Cepstral Zero Pole Vocal Tract Models for TTS, in *Proceedings of the International Conference EUROCON'2001*, July 7-9, 2001, Bratislava, Slovakia, pp. 459-462. ISBN 0-7803-6490-2

[8] M. Vlček, R. Zahradník, and R. Undehauen, Analytical Design of FIR Filters, in *IEEE Transactions on Signal Processing*, September, 2000, pp. 2705-2709.

[9] G. Cherubini, E. Eleftheriou, and S. Olcer, 1999, Filtered multitone modulation for VDSL. Proc. IEEE Globecom'99, Rio de Janeiro,Brazil, pp.1139-1144

[10] N. Benvenuto, S. Tomasin, L. Tomba, Equalization methods in DMT and FMT Systems for Broadband Wireless Communications. In IEEE Transactions on Communications, vol. 50, no. 9. [s.l.] : [s.n.], 2002. s. 1413-1418

[11] P. Šilhavý, Half-overlap subchannel Filtered MultiTone Modulation with the small delay. In The Seventh International Conference of Networking. 1. Cancun, Mexico: IARIA, LCN 2007941923, 2008. s. 1-5. ISBN: 978-0-7695-3106-9.