IMPROVING PERFORMANCE OF HIGH PRECISION SIGNAL PROCESSING ALGORITHMS ON PROGRAMMABLE DSPS

M. N. Mahesh

Electrical Communication Engg.
Indian Institute of Science,
Bangalore - 560012, INDIA
maheshmn@protocol.ece.iisc.ernet.in

Mahesh Mehendale

Texas Instruments (INDIA) Ltd.
Golf View Homes, Wind Tunnel Road,
Bangalore - 560017, INDIA
mhm@india.ti.com

ABSTRACT

In applications, such as high quality audio, that need more than 16 bits of precision, the processing of signals on a 16-bit DSP requires double precision computation and is hence time consuming. Since in Residue Number System (RNS), the high precision data is decomposed to lower precision for its processing, in this paper, we propose RNS for increasing the performance of such applications implemented on a single processor. We apply multirate architectures and also suggest some architectural extensions to the processor to further enhance the performance. The results show that performance improvement of more than 57% can be achieved with these implementations. We also show that the power dissipation can be significantly reduced with such implementations.

1. INTRODUCTION

There has been an increasing demand for the increase in the word length for high quality audio applications such as CD, Digital Audio Tape(DAT), and high quality PC audio. Typically, 24-bit precision is being used for such applications. For precise processing of these 24-bit signals on a 16-bit processor there is an overhead of double precision computations which makes these implementations, computation intensive and time consuming. This problem is addressed by use of multiple processors [1] or by use of floating point or a higher precision fixed point processor. Since, these methods are expensive, we present techniques by which precise processing of these high precision signals can be done on a low precision processor with better performance.

Residue Number System(RNS) has been used in the literature in high speed DSP applications with hardwired implementations using smaller moduli [2, 3]. In software, RNS has been used on multiple processors for performance improvement in DSP applications [1]. To the best of our knowledge, this is the first attempt in which RNS has been used to improve performance of implementations on a single processor.

In RNS, an integer is represented as a set of residues with respect to a set of integers called the moduli set. Let $(m_1, m_2, .., m_n)$ be the moduli set. Then, an integer X in [0,M-1] can be uniquely represented in RNS as $(X_1, X_2, .., X_n)$ where, $X_i = (X)\ modulo\ m_i$ and M is the dynamic range given by LCM of all the moduli [4]. If the operator *op* denotes the binary operation of addition, subtraction or multiplication in the RNS, then $W = X\ op\ Y$ satisfies the property $W_i = |(X_i op Y_i)|_{m_i}$. As can be seen, in RNS the processing is done in low-precision because of the modulo arithmetic computations involved. So, we consider RNS as an effective means by which the performance of high-precision processing on a low-precision DSP can be enhanced. We consider the implementation of FIR filtering using RNS on TMS320C5x DSP [5]. Use of multirate architecture has been proposed in the literature to reduce the computational complexity [6]. We apply multirate architectures in the RNS based implementations for further improvement in performance. The performance of 24-bit precision algorithms using RNS and multirate architectures have execution times much larger than the corresponding 16-bit binary implementations. So, we suggest extensions to TMS320C5x processor with minor overheads to enhance the performance of the RNS based implementations.

2. RNS BASED FIR FILTER IMPLEMENTATION

The execution time of RNS based implementations depends significantly on the the moduli set selected. Different issues involved in the selection of moduli set are discussed below.

2.1. Moduli selection

The selection of moduli is an important aspect as it determines the computation time of RNS based implementation. The issues involved are:

0-7803-5471-0/99/$10.00©1999 IEEE

1. The moduli set should be pairwise relatively prime to enable high dynamic range.
2. The moduli set has to be selected such that residue computations are easy(eg. 2^n,$2^n - 1$) [4].
3. Since the implementation is on a single processor and computations w.r.t. to each moduli has to be done sequentially, it is desirable to have as few moduli as possible.
4. The moduli should have simple multiplicative inverses. This ensures conversion from residue to binary domain with less computations.

Keeping in view the above issues, a moduli set of form $(2^n, 2^n - 1, 2^{n-1} - 1, 2^{n-2} - 1)$ has been employed in our implementations. It has been shown by Szabo and Tanaka [4] that the four moduli are pairwise relatively prime if and only if n is odd. So, for $n = 15$, this moduli set provides a high dynamic range of more than 56 bits and also offers an advantage of simplicity in determining the additive and multiplicative inverse. The additive inverse of a number w.r.t. modulo of form $2^k - 1$ is just the 1's complement representation of the number.

2.2. Algorithm Description of RNS based FIR filter implementation

FIR filtering is achieved by convolving the input data samples with the desired unit impulse response of the filter. Output Y(n) of an N-tap FIR filter is given by the weighted sum of latest N input data samples.

$$Y(n) = \sum_{i=0}^{N-1} A[i] * X[n-i] \qquad (1)$$

where, A[i] represent the filter coefficients and X[i] represent input data samples. The pseudo code of the RNS based implementation of FIR filtering on a TMS320C5x DSP is given below. The moduli set selected in this implementation is $(2^{15}, 2^{15} - 1, 2^{14} - 1, 2^{13} - 1)$.

```
NXTPT:
  Data_read();          /* read 24-bit binary data input
  Bin2res_conv();       /* conversion to residue domain.
  Mod_M1_fir_filter();/* implement modulo M1 filter
  Mod_M2_fir_filter();/* implement modulo M2 filter
  Mod_M3_fir_filter();/* implement modulo M3 filter
  Mod_M4_fir_filter();/* implement modulo M4 filter
  Res2bin_conv();       /* convert back to binary domain.
  Output_data();        /* put filtered data at the out port.
goto NXTPT;
```

The filter coefficients w.r.t. each modulo are assumed to be put in data memory. The function Mod_M_i_fir_filter is implemented by performing a series of multiply-accumulate operations on the data and filter coefficients w.r.t. modulo M_i(N times in case of N-tap filter). If the modulo is of the form 2^k the overflow during the multiply-accumulate operations can be neglected since the residue is just is lower-order k-bits irrespective of the higher order bits. But if the modulo is in the form $(2^k - 1)$, the overflow needs to be considered. In this case so as to minimize the computation time, instead of computing the residue every time, the overflow is accumulated in a temporary register and the residue is computed on the final accumulated sum. Now we apply multirate architectures to enhance the performance.

3. MULTIRATE ARCHITECTURE BASED FIR FILTER IMPLEMENTATION IN RNS

Multirate architectures have been used in the literature for reducing the computational complexity [6] and also low power applications [7]. Multirate architectures involve implementing the FIR filter in terms of its decimated sub-filters [6]. The signal flow graph of a 1-level decimated multirate architecture based FIR architecture is shown in figure 1. In this architecture the input and the filter are decimated by a factor of two. The decimation factors for the input and the filter can be same or different and can be integers higher than two [6]. Each such combination results in a different multirate architecture. In this paper, we restrict our implementation of FIR filtering in RNS with decimation factor of two for both input and the filter. In this implementation, the performance improvement comes from the fact that the number of multiplications involved in the computation of single output sample is reduced from N in the case of usual implementation to $(3N/4)$ in the case of multirate implementation.

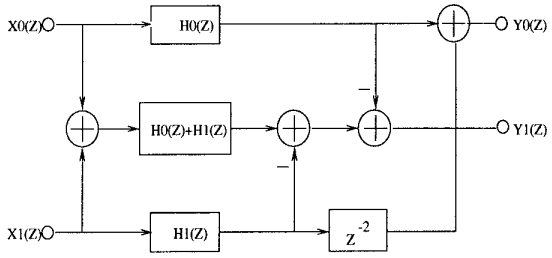

Figure-1: 1-level Decimated Multirate Architecture

As can be noted, the architecture processes two input samples simultaneously to produce the corresponding two outputs. The pseudo code for the RNS based FIR filtering using multirate architecture is similar to the usual filter implementation shown in section (2) except that the function Mod_M_i_MR_fir_filter() is called instead of Mod_M_i_fir_filter(). The results showing the performance improvement are shown in section 5. Now we look into the architectural extensions

to instruction set of the TMS320C5x processor that can further enhance the performance of RNS based implementations.

4. ARCHITECTURAL EXTENSIONS TO TMS320C5X DSP

Performance can be enhanced if optimization is done for the most frequently used block of instructions. The frequency of occurrences of the basic blocks for the RNS based implementation of FIR filtering with and without multirate architectures is analyzed for a 20-tap FIR filter. Figure 2 shows the analysis. It is been observed that optimization can be performed on three of these basic blocks by some extensions to the instruction set. Now we consider each of these blocks and suggest the extensions.

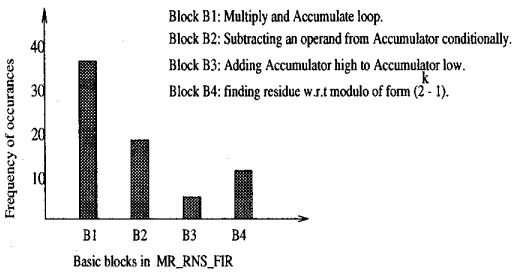

Figure-2: Plot of frequency of occurances of different basic blocks.

4.1. Extending Accumulator precision

For higher order filters, the multiply-accumulate(MAC) loop has high impact on the overall performance. It has been observed that the accumulator range is posing a limitation to the use of a compound instruction like MACD which does the job of accumulating the previous product and multiply the coefficient with data point along with data move. So, a major improvement can be obtained if the dynamic range of accumulator can be increased from 32-bit to 40-bit.

However, increasing the arithmetic precision of accumulator needs a higher precision adder. Also, the data path size increases resulting in further over-head. So, though this extension has the potential for significant performance improvement, the overhead involved due to this extension is significant. We look into some other modifications that can be done in this MAC loop.

4.2. Conditional Increment

The basic block B1 has been analyzed and is found to perform the MAC loop. In the MAC loop, after every summation operation, the accumulator is checked for overflow and a counter is incremented in case of overflow. So, an instruction of the form conditional increment which increments the counter in case of an overflow would enable faster implementation of this loop. With this optimization, the number of clock cycles needed for each MAC loop can be reduced from 5 to 3.

```
INCC   *,AR1 ; Increment the register pointed by ARP
             ; conditionally and change ARP to AR1.
```

4.3. Conditional Subtract from Accumulator

Another block(B2) which occurs frequently is the modulo addition of two residues. So, if the sum of residue happens to be greater than the modulo, the modulo needs to be subtracted from the sum in order to perform modulo addition. This can be implemented on TMS320C5x processor by explicitly comparing the sum with the modulo and then performing the subtraction. This requires at least 4 clock for implementation on TMS320C5x processor. But if the architecture supports a conditional subtract instruction as described below, this operation can be done in just 1 cycle.

SUBC M_i ; subtract M_i from Accm if Accm $\geq M_i$

4.4. Add Accumulator high to Accumulator low

The other basic block(B3) which occurs more frequently, basically adds shifted most significant word of the accumulator to the least significant word of the accumulator. This occurs whenever residue of a number is to be found w.r.t. modulo of the form $(2^k - 1)$. This operation takes 3 cycles for implementation on C5x processor and can be reduced to 1 if the architecture supports the instruction Add Accumulator high(AH) to Accumulator low(AL) with shift(ADAH)

ADAH n ; Add AH to AL with shift n.

Now we present the results showing the performance improvement.

5. RESULTS

The comparison is made in terms of the number of execution time(cycles), program memory size and data memory size required in each implementation in terms of N (no. of filter taps). Comparisons are made between the following implementations.

1. BIN_FIR: FIR filter implementation in binary domain.
2. RNS_FIR: FIR filter implementation in residue domain.
3. MR_BIN_FIR: FIR filter implementation in binary domain using multirate architectures.
4. MR_RNS_FIR: FIR filter implementation in residue domain using multirate architectures.

Table 1 shows the comparisons.

It is to be noticed that the memory size needed for storing the filter coefficients and the data points for RNS_FIR is

Table 1: Comparison of different FIR filter implementations

	Pgm. Mem. size(words)	Data Mem. size(words)	Exec. time (cycles)
BIN_FIR	86	4N+8	32N+20
RNS_FIR	N+269	7N+19	16N+261
MR_BIN_FIR	163	6N+14	51(N/2)+48
MR_RNS_FIR	3(N/2)+647	21(N/2)+24	12N+314

* N - No. of filter taps.

Table 2: Comparison of different operations in FIR filter implementations

	Loads & Stores	Adds	Mults
BIN_FIR	11N+4	10N	4N
RNS_FIR	4N+48	3N+60	4N+1
MR_BIN_FIR	39(N/4)+12	15(N/2)+7	3N
MR_RNS_FIR	3N+45	9(N/4)+30	3N+1

* It is assumed that the memory read and write take 1 cycle.

twice than that of BIN_FIR and data memory requirement for MR_RNS_FIR is thrice than that of corresponding binary implementation. As expected, the number of loads, stores, add and multiplication operations for multirate implementations are (3/4) times the number of operations needed for their binary counterparts. Also, it is observed that for filter lengths more than 18, RNS_FIR and MR_RNS_FIR have better execution times than BIN_FIR implementation. For a 40-tap filter, MR_RNS_FIR is 34% faster than the BIN_FIR and 57% faster in case of 48-tap filter.

The frequency of occurrences of various operations for different implementations is shown in table 2. As can be seen, for a 48 tap filter the number of memory accesses needed for MR_RNS_FIR is 65% less than those needed for BIN_FIR implementation. Due to the reduction in the number of memory accesses, the power dissipation is reduced significantly. Also, the number of additions in the case of RNS based implementations are much less than their binary counterparts. So, this also plays an important role in reducing the power dissipation in the these implementations.

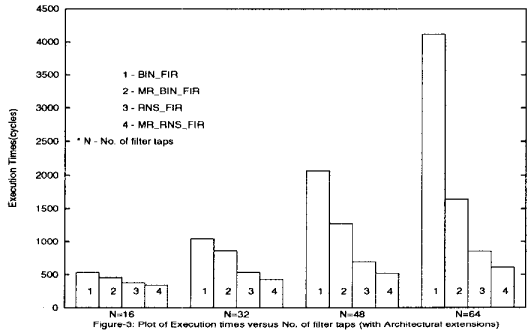

Figure-3: Plot of Execution times versus No. of filter taps (with Architectural extensions)

Now we look into the implication of architectural extensions on the performance of RNS based implementations. The plot of execution times for different FIR filter implementations with architectural implementations versus the number of filter taps is shown in figure 3. It can be seen that the performance increases from 57% to 75% for a 48-tap RNS based FIR filter on a TMS320C5x processor with architectural extensions. It is to be noted that the improvement is more significant for higher order filters.

6. CONCLUSION

We have presented methods by which performance improvement can be achieved for precise processing of high precision signals on a low precision processor. Performance improvement of more than 57% can be achieved for 24-bit precision implementation of a 48-tap filter implemented on a 16-bit TMS320C5x DSP. The suggested architectural extensions can improve the performance from 57% to 75% for a 48-tap filter. Since the overhead to due residue to binary and binary to residue domain conversions is constant, as the processing of signals in residue domain increases there will be drastic improvement in performance as can be seen from figure 3. Due to the reduction in the number of memory accesses and the number of additions in RNS based implementations, there can be significant reduction in the power dissipation of the RNS based implementations compared to the corresponding binary implementations.

7. REFERENCES

[1] W. K. Jenkins, "Techniques for high precision digital filtering with multiple microprocessors", Proc. of Midwest Symposium on Circuits and Systems, 1977.

[2] W. K. Jenkins and B. Leon, "The use of Residue Number System in the design of finite impulse response digital filters", IEEE Trans. on Circuits and Systems, Vol: CAS-24, No.4, pp: 191-201 Apr. 1977.

[3] M. A. Soderstrand and R. A. Escott, "VLSI implementation in multiple-valued logic of an FIR digital filter using residue number system arithmetic", IEEE Trans. on Circuits and Systems, vol. CAS-31, pp. 1065-1067, July 1977.

[4] N. S. Szabo and R. I. Tanaka, *Residue arithmetic and its Applications to computer technology*, New York: McGraw-Hill,1967.

[5] TMS320C5x User's Guide, Texas Instruments, 1993.

[6] Z. J. Mou and P. Duhamel, "Short-length FIR filters and their use in fast nonrecursive filtering", IEEE Trans. on Signal Processing, June 1991, pp. 1322-1323.

[7] M. Mehendale, S. D. Sherlekar, "Low power realization of FIR filters using multirate architectures", VLSI Design conference, 1996, pp. 370-375.