Improving Area Efficiency of Residue Number System based Implementation of DSP Algorithms

M.N.Mahesh, Satrajit Gupta
Electrical and Communication Engg.
Indian Institute of Science
Bangalore - 560012, INDIA
maheshmn,satrajit@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

Residue Number System based applications involve modulo-arithmetic which is typically implemented using look-up-tables (LUTs) for a small value of modulus. In this paper, we present a data coding technique to minimize the area of these LUTs when implemented using two level logic structures such as PLAs. We also present a technique that exploits the symmetry in these computations to further optimize the LUTs. Results show that area improvement of upto 66% can be achieved using these techniques.

1. Introduction

High speed signal processing applications require dedicated hardwired implementations to meet the stringent performance requirements. Residue Number System (RNS) based implementations of DSP systems have been presented in the literature [1, 2, 3] as a technique for high speed realization. RNS achieves this by breaking an operation (such as addition, multiplication etc.) into smaller operations that can be executed in parallel. In this paper, we present techniques for improving area efficiency of RNS based implementations of DSP algorithms. While these techniques are generic, they have been discussed in the context of PLA based implementations.

The paper is organized as follows. We start with a brief introduction to RNS in section 2. We present techniques for area efficient PLA implementation of residue arithmetic in section 3. section 4 presents the results that highlight the effectiveness our techniques. Finally we conclude in section 5 with a brief discussion on the future scope of our work.

2. Residue Number System

In RNS, an integer is represented as a set of residues with respect to a set of integers called the Moduli. Let $(m_1, m_2, m_3, ..., m_n)$ be a set of relatively prime integers called the Moduli set. An integer X can be represented as $X = (X_1, X_2, X_3, ..., X_n)$ where

$$X_i = (X)\ modulo\ m_i \quad for\ i = 1, 2, .., n \tag{1}$$

we use notation $|X|_{m_i}$ to represent X_i the residue of X w.r.t m_i. Given the moduli set, the dynamic range(M) is given by the LCM of all the moduli. If the elements are pair-wise relatively prime, the dynamic range is equal to the product of all the moduli [4]. The bit-precision of a given moduli set is

$$bits = log_2(M) \tag{2}$$

where M is the dynamic range of the given moduli set. So, the moduli set is determined based on the bit-precision needed for the computation. Let X,Y and Z have the residue representations $X = (X_1, X_2, X_3, ..., X_n)$, $Y = (Y_1, Y_2, Y_3, ..., Y_n)$ and $Z = (Z_1, Z_2, Z_3, ..., Z_n)$ respectively and $Z = (X\ op\ Y)$ where *op* is any operation in addition, multiplication or subtraction. Thus we have in RNS,

$$Z_i = |X_i\ op\ Y_i|_{m_i} \quad for\ i = 1, 2, .., n \tag{3}$$

since, X_i's and Y_i's require lesser precision than X and Y, the computation of Z_i's can be performed faster than the computation of Z. Moreover, since the computations are independent of each other, they can be executed in parallel resulting in significant performance improvement.
For example, Consider the moduli set (5,7,11)
Let $X = 47 = (|47|_5, |47|_7, |47|_{11}) = (2, 5, 3)$
$Y = 31 = (|31|_5, |31|_7, |31|_{11}) = (1, 3, 9)$
and $Z = X + Y = 47 + 31 = 78 = (|78|_5, |78|_7, |78|_{11}) =$

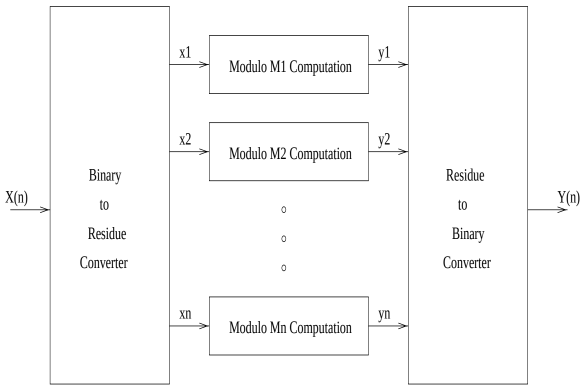

Figure 1. RNS based Implementation of DSP Algorithms

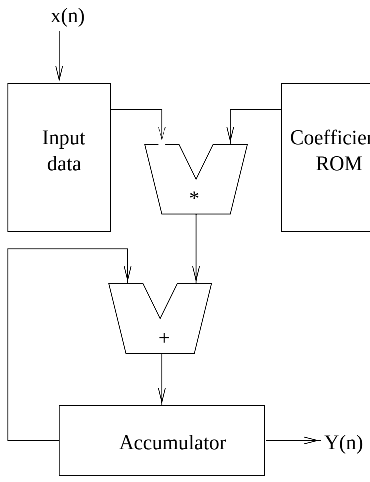

Figure 2. Conventional FIR filter Implementation with a Multiplier and an Adder

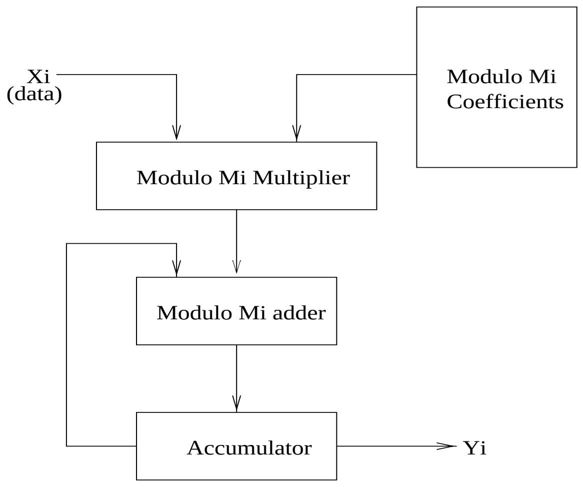

Figure 3. Modulo M_i filter

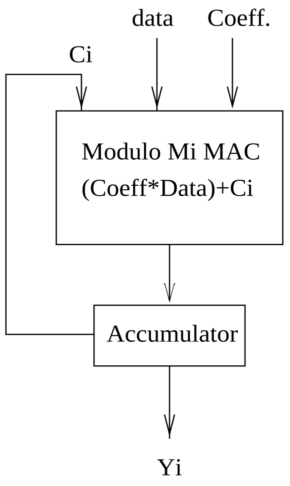

Figure 4. MAC based Modulo M_i filter

$(3,1,1)$ Z_i's can be computed from (3) as $Z = (|2+1|_5, |5+3|_7, |3+9|_{11}) = (3,1,1)$

While in RNS, multiplication, addition, and subtraction can be performed with high speed, the operations like division, scaling, and magnitude comparision require several basic operations. They are slower and more complicated to implement. Since most DSP algorithms (such as FIR and IIR filtering, FFT, correlation, DCT etc) do not have these operation to be performed, this limitation does not apply. Figure 1 shows a generic structure of the RNS based implementation of such DSP algorithms. As an example, consider a typical FIR filter implementation shown in Figure 2. In the RNS domain, it is implemented as multiple modulo m_i filters. Figure 3 shows the structure of a modulo m_i filter.

Generally, modulo-arithmetic operations like addition and multiplication are performed using Look-Up Tables (LUTs) when modulo is small. Thus for the FIR filter shown in Figure 3, the operations modulo m_i multiplication and modulo m_i are implemented as LUTs. Each tap of the FIR filter thus requires two look-ups. The performance can be further improved by merging the two LUTs into a module m_i Multiply-Accumulate (MAC) LUT. The resultant structure of the FIR filter is shown in Figure 4.

In this paper we present area efficient implementation of LUTs for addition, multiplication and the multiply-accumulate operation. For two-level implementation of LUTs, PLAs are preferred to ROMs because of their area efficiency. We now present two techniques for improving area efficiency of PLA based implementation of LUTs.

3. Techniques for Area-efficient PLA Implementation

We present the following techniques for improving the area efficiency of PLA based modulo arithmetic computation:

1. Residue encoding: Since all the computation in the RNS domain is performed using LUTs, the residue values can be treated as symbolic variables/constants. The residues hence need not be encoded in the binary form (i.e., 0 - 000, 1 - 001 etc.). Since encoding has the direct impact on the area of the PLA, optimum encodings can be derived as to achieve maximum area reduction.

2. Inherent symmetry of modulo arithmetic: Since the modulo addition and the modulo multiplication operations are commutative, the LUT need not store results for the cases where the inputs are swapped w.r.t. another entry in the LUT. For example, the LUT need not store results for both A+B and B+A computations. The inherent symmetry of the modulo arithmetic can be used to minimize the area of the PLA.

3.1. Residue Encoding

Let us consider the truth table for modulo-3 multiplication.

Table 1.

Input1	Input2	Output
0	0	0
0	1	0
0	2	0
1	0	0
1	1	1
1	2	2
2	0	0
2	1	2
2	2	1

The area of the PLA implementing Table 1 is different if the residues 0,1,2 are encoded as (00,01,10) vs (11,01,00) vs (10,01,11) and so on. The aim of residue encoding is to come up with an assignment of unique binary vectors to the residues, so as to minimize the area of the PLA. We refer to this as the modulo encoding problem. This problem is similar to the well studied input-output encoding problem for PLA minimization [5, 6]. The distinguishing feature of modulo encoding problem is that symbols in all the input columns(2 in case of addition and multiplication, 3 in case of MAC) and the output column are members of the same set. So we have to ensure that the code assigned to a symbol be same across different columns. Currently there is no encoding tool that considers more than two input columns at a time for encoding. Moreover none of the tools ensures consistency of the code assigned to the same symbol across more than two columns(the input and the output column for PLA based FSM encoding) [7, 8]. Because of these differences with the conventional encoding problem, we used some heuristics and modified a FSM encoding tool, NOVA [9], to generate the residue encodings.

We start with multi-valued output disjoint minimization to get the initial set of input constraints. Partial order relations among the output states are developed by constructing a directed acyclic graph(DAG) where each vertex represents an output state(say u) and a directed edge (u,v) means that state u bit-wise dominates state v. Symbolic minimization is done using these partial order relations(The ON set of each output state is expanded using the don't care sets. After this multiple valued minimizer is invoked to minimize the cardinality of the ON-set corresponding to the output). The final set of input constraints is obtained from this symbolically minimized cover. In addition to this, we have to ensure that the input constraints derived from the different input columns are satisfied for the same encoding of states across the columns. So we combine the input constraints derived from the different columns and apply the ordered face hyper-cube embedding algorithm. The pseudocode for our encoding algorithm is presented below.

```
Encoding()
{
    Mini();
    Symbolic_minimization();
    Combine_constr();
    IO_hybrid_code();
}
```

In the pseudocode given above, Mini does the initial output-disjoint multi-valued minimization of the symbolic truth table. Symbolic_minimization reduces the truth table further by using the partial order relationships among the output states and gives the final set of input constraints for all the columns. To maintain consistency of codes across columns, the input constraints for all the input columns are combined together in the procedure Combine_constr. Finally, the IO_hybrid_code assigns codes to each state such that the constraints developed by the Combine_constr are met. The mini, Symbolic_minimization and IO_hybrid_code routines are similar to the ones used in NOVA, while the Combine_constr routine has been developed to take care of the unique nature of the residue coding problem.

The gain in area over conventional binary encoding('0'-'000','1'-'001',.) using our encoding algorithm is presented in section 4.

3.2. Inherent Symmetry of Modulo Arithmetic

An important property that can be exploited for implementing area efficient PLAs for modulo arithmetic is the inherent symmetry among the operands. The symmetry is due to the commutativity of multiplication and addition. Using this, the truth table can be minimized. For example, both a+b and b+a give the same output. So, one of these entries can be eliminated in the truth table. With this a significant reduction in the truth table can be achieved. For example, the LUT for modulo 3 multiplication can be reduced to the truth-table shown in Table 2 which has 6 entries compared to 9 entries of the initial truth-table shown in Table 1.

Table 2.

Input1	Input2	Output
0	0	0
0	1	0
0	2	0
1	1	1
1	2	2
2	2	1

However, to exploit this symmetry we need additional logic that detects the redundancy in input operand patterns and feeds the arithmetic block (PLA implementing an adder/multiplier/MAC) with an irredundant input pattern. In Figure 5, we show a block diagram of a system that exploits symmetry in implementing modulo arithmetic.

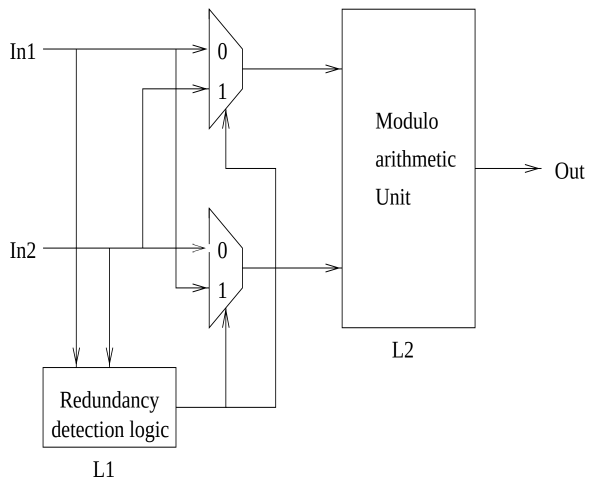

Figure 5. Block schemetic of redudancy detection based modulo-arithmetic

In addition to the area overhead of L1 and mux's this scheme also increases the delay. The area and delay overhead of L1 can be eliminated by choosing a simple redundancy detection scheme. For example, we can select the LSB of an input operand as the mux select line. Then the reduced truth table for modulo 3 multiplication (under conventional binary encoding) is shown below.

Table 3.

Input1	Input2	Output
0	0	0
0	2	0
1	0	0
1	1	1
1	2	2
2	0	0
2	2	1

The LUT shown in Table 3 has 7 rows compared to 9 rows of the initial truth-table shown in Table 1 1 and 6 rows of the truth-table shown in Table 2. It can thus be noted that this input bit-based selection scheme cannot exploit the full symmetry and hence does not result in as much reduction as the scheme shown in Figure 5. The reduction in the truth table that can be achieved with this scheme depends on the encoding, the bit chosen to be the mux select line and the modulo value.

We thus have two alternatives to exploit the symmetry of modulo arithmetic:

1. Exploit the full symmetry with elaborate redundancy detection scheme.

2. Exploit the partial symmetry with simple redundancy detection scheme.

In the following subsections we give the implementation details for both the schemes.

3.2.1 Exploiting full symmetry with elaborate redundancy detection

In this scheme the residue encoding should be done such that the area of (L1+L2) (Figure 5) is minimized. Here we have to encode the inputs of L1 and boththe input and output of L2. As all the symbols belong to the same set, we have to maintain the compatibility of code assigned to a symbol across L1 and L2.

To arrive at the encoding we get the input and output constraints for blocks L1 and L2 separately. Then the constraints are combined and the encoding algorithms of NOVA are used to get the final encoding. The results for the area of L1+L2 with this encoding are given in the next section.

3.2.2 Exploiting the partial symmetry with simple redundancy detection scheme

In this scheme, the area of L2 depends on the encoding and the bit chosen to be the mux select line. These two factors - the encoding and the bit line that gives maximum reduction are dependent on each other. So we need to employ a heuristic to come up with an encoding first and then choose the bit that gives the maximum reduction. We have adopted the heuristic of starting with a code that is optimal with respect to residue encoding (i.e. we neglect symmetry to come up with the encoding followed by selecting the bit that gives the maximum reduction in the truth table.

For a given an encoding, we can use the following results for bit selection:

1. We can select the bit from any input as the truth table is commutative for both the inputs(for addition, multiplication and the two multiplicand columns of MAC)

2. For maximum reduction, the bit-position that has the most balanced distribution of 0's and 1's among the residue codes should be chosen. i.e., if a_i is the number of codes for which the i-th bit is 1 and b_i is the no. of codes for which the i-th bit is 0, then the bit position i for which maximum reduction is possible is the one where $|a_i - b_i|$ is minimum

Using the above mentioned results we can select a bit for a particular encoding so as to achieve maximum area reduction.

4. Results

We now present results that highlight the effectiveness of these techniques in reducing area of the PLAs implementing modulo addition, modulo multiplication and modulo MAC operations. The results are presented for the modulo set of (5, 7, 9, 11, 13). The metric chosen for PLA area is the product of the no. of rows and columns in the PLA. The no. of columns in a PLA is equal to the 2*(no. of input bits) + the no. of output bits. The number of rows are obtained from the residue encoded truth table which is minimized using Espresso [10] .

Tables-4, 5 and 6 present results for the following cases:
column 1: The area of PLA with conventional binary coding('0'-'000', '1'-'001',.).
column 2: The area of PLA with coding obtained using the algorithm(residue coding) given in section 3.1.(This encoding is referred to as code1).
column 3: The area of PLA implementation in case of elaborate redundancy detection with conventional coding(referred to as PLA(L1+L2)).
column 4: The area of PLA implementation in case of elaborate redundancy detection with combined encoding(referred to as PLA(L1+L2),code2).
column 5: The area of PLA implementation in case of elaborate redundancy detection with code1(referred to as PLA(L1+L2),code1).
column 6: The area of the PLA with simple redundancy detection with conventional coding(referred to as PLA(bit)).
column 7: The area of the PLA with simple redundancy detection with code1(referred to as PLA(bit),code1).

The area overhead of the mux's is ignored for cases in columns 3,4,5,6 and 7.

Table-4: Area estimates of PLA based mod-add implementation

Modulo	PLA (conv)	PLA (code1)	PLA (L1+L2)	PLA (L1+L2) (code2)	PLA (L1+L2) (code1)	PLA (bit)	PLA (bit) (code1)
5	270	240 (11%)	217 (19%)	172 (36%)	217 (19%)	225 (16%)	195 (27%)
7	540	420 (22%)	363 (32%)	368 (31%)	355 (34%)	420 (22%)	315 (42%)
9	1060	1220	736 (30%)	673 (36%)	804 (24%)	900 (15%)	980 (7.5%)
11	1580	1340 (15%)	1050 (33%)	1172 (25%)	1066 (32%)	1320 (17%)	1280 (19%)
13	2200	1740 (21%)	1364 (38%)	1443 (34%)	1198 (45%)	1820 (17%)	1480 (33%)

Table-5: Area estimates of PLA based mod-mult implementation

Modulo	PLA (conv)	PLA (code1)	PLA (L1+L2)	PLA (L1+L2) (code2)	PLA (L1+L2) (code1)	PLA (bit)	PLA (bit) (code1)
5	225	135 (40%)	202 (10%)	170 (24%)	157 (30%)	165 (27%)	165 (27%)
7	285	330	258 (9%)	325	314	225 (21%)	330
9	960	540 (43%)	656 (31%)	644 (32%)	547 (43%)	760 (21%)	720 (25%)
11	1440	1000 (31%)	990 (31%)	1046 (27%)	884 (38%)	1200 (16%)	980 (32%)
13	1860	920 (51%)	1324 (28%)	1448 (22%)	886 (52%)	1540 (17%)	1160 (38%)

As can be seen from the results, the PLA area can be reduced by as much as 45% (corresponding to modulo-13 addition using elaborate redundancy detection with residue encoding). The results also show that no one combination of techniques results in the most area reduction across all moduli. While elaborate redundancy detection gives maximum area reduction in most cases, it has an associated delay

Table-6: Area estimates of PLA based mod-MAC implementation

Modulo	PLA (conv)	PLA (code1)	PLA (L1+L2)	PLA (L1+L2) (code2)	PLA (L1+L2) (code1)	PLA (bit)	PLA (bit) (code1)
5	1428	1113 (22%)	892 (38%)	884 (38%)	766 (47%)	1197 (16%)	1029 (27%)
7	4284	2604 (39%)	2346 (45%)	2133 (50%)	1781 (58%)	3381 (21%)	2667 (37%)
9	10696	5124 (52%)	6240 (42%)	3838 (64%)	3648 (66%)	8876 (17%)	7224 (32%)
11	20636	17388 (16%)	11454 (45%)	10748 (48%)	10150 (51%)	16744 (18%)	14728 (28%)
13	31892	32088	17670 (45%)	18415 (42%)	17264 (46%)	26320 (17%)	25228 (20%)

overhead. For modulo-7 addition, the simple redundancy results in the minimum area and it comes with minimal delay overhead.

In modulo multiplication implementation, the PLA area can be reduced by as much as 52% (corresponding to modulo-13 multiplication using elaborate redundancy detection with residue encoding). The results also show that for modulo-5 multiplication, just the residue encoding (i.e. with no redundancy detection) gives the most area efficient PLA.

In case of MAC computation, the results show that the PLA area can be reduced by as much as 66%. In this case, the scheme of elaborate redundancy detection with residue encoding, gives the minimum area for all the moduli (i.e. 5, 7, 9, 11 and 13).

5. Conclusions and Future work

We have presented techniques of residue encoding and redundancy elimination for area efficient PLA implementation of modulo addition, modulo multiplication and modulo MAC operations. These two techniques complement each other and together result in upto 45% reduction for modulo-addition, upto 52% reduction for modulo-multiplication and upto 66% reduction for modulo-MAC operations.

For the simple redundancy detection scheme we select a bit that results in the minimum number of rows in the truth table. However, a truth table with more number of rows may result in reduced number of product terms after minimization by Espresso. We plan to analyze whether this indeed is the case for modulo arithmetic.

Our residue encoding scheme implies that the binary-to-residue and the residue-to-binary converters also support the encoding. In our analysis, we have assumed that the residue encoding has minimal impact on the area of binary-to-residue and residue-to-binary converters. This however may not be the case across all moduli sets. We plan to work on an optimization strategy that aims at area minimization of the entire system shown in Figure 1.

The symmetry properties of modulo arithmetic have been exploited for a significant improvement in area. The identity properties of addition and multiplication can be also used for further reducing the number of entries in the truth-table. This however has an overhead of 0-detect logic in case of modulo addition and 1-detect logic for modulo multiplication. We plan to evaluate the impact of these techniques on the overall area of the modulo computation.

Finally we also plan to look at area minimization for multi-level logic implementation of modulo computation. We are in the process of deriving a cost function that can be used to arrive at an optimal residue encoding.

References

[1] 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 Syst., vol. CAS-31, pp. 1065-1067, July 1977.

[2] W.K.Jenkins, "A highly efficient residue combinatorial architecture for digital filters," Proc. of IEEE, Vol. 66, pp. 700-702, June 1978.

[3] 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.

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

[5] A.Saldanha, T.Villa, R.Brayton and A.L.Sangiovanni Vincentelli, "A framework for satisfying input and output encoding constraints," DAC, Proceedings of DAC, pp. 170-175, 1991.

[6] S.Yang and M.Ciesielski, "Optimal and suboptimum algorithms for input encoding and its relationship to logic minimisation," IEEE Trans. on Computer Aided Design, vol. CAD-10, No. 1, pp. 4-12, Jan. 1991.

[7] G.De Micheli, R.Brayton, and A.L.Sangiovanni Vincentelli, "Optimal state assignment for finite state machines," IEEE Trans. on Computer-Aided Design, vol. CAD-4, No. 3, pp. 269- 284, July 1985.

[8] G.De Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," IEEE Trans. on Computer-Aided Design, vol. CAD-5, No. 4, pp.597-616, Oct. 1986.

[9] T.Villa and A.L.Sangiovanni Vincentelli, "NOVA : State assignment of finite state machines for optimal two-level logic implementation," IEEE Trans. on Computer Aided Design, vol. CAD-9, No. 9, pp. 905-924, Sept 1990.

[10] G.De Micheli, *Synthesis and optimization of digital circuits.* McGraw-Hill, 1994.