Optimization techniques for Digital / DSP circuits

N. Nithyakalyani[*1], Mrs. B. Bala Tripura Sundari[#2], Mr. S.R. Ramesh[#3]

[*]*M.Tech VLSI Design, Amrita School of Engineering, Amrita Vishwa Vidyapeetham University,*

[#]*Department of ECE, Amrita School of Engineering, Amrita Vishwa Vidyapeetham University*

Abstract – **The fast pace of growth of the semiconductor industry has been both a blessing and as well as a biggest challenge to its future. The increase in the number of transistors that can be packed in a single wafer is expected to come to a standstill by 2020. Therefore large number of researches is going on to exploit science in such a way so that we bring out newer and efficient designs with the existing technology. The need for optimization of designs in terms of speed, power or area is the most looked upon field. We present in this paper few techniques which when used in combination has been already proved to give area and speed optimized designs. The techniques presented are 1) Compressor tree using Carry save adders and 2) Common Subexpression Elimination. The large scattering of logic operations over the arithmetic operations is the main target for applying Compressor tree after a series of Rewriting and Sorting rules. Common Subexpression Elimination involves identification of redundant terms in expressions and careful restructuring of resources. So far the above Optimization algorithms have been implemented for FIR filters and few benchmarks. We have presented a systematic analysis after comparison of the above methods with conventional methods. FIR filters with different orders were taken and Common Subexpression Elimination method was implemented on them. The efficiency of the method is brought out. Appropriate benchmark circuits were chosen for implementing Compressor tree technique. The comparison of these methods with their conventional mechanisms is presented.**

Keywords **– Carry save adders, compressor trees, Three-greedy technique, Common Subexpression Elimination, Horizontal Common Subexpression Elimination, Vertical Common Subexpression Elimination.**

I. INTRODUCTION

There are tremendous innovations in the field of computers each day owing to the increased demands. Technology is advancing in such a rapid phase that we always want better and better devices for our applications. The computers used in the olden times are replaced now by more compact and efficient devices. Therefore the need of the hour is to choose optimized designs which provide number of applications within the same or even lesser area. We are also looking out for the speed of the operations in this fast world. All these demands have paved way for research in the fields of optimization of datapath circuits. Most of these datapath processors involve the use of arithmetic circuits for its operation. The target for area reduction usually is the multiplier. There are many optimization algorithms in the recent years which concentrate on the area reduction of multipliers. This paper discusses few of the many optimization algorithms targeted for processors, compares them and arrives at the best possible combination to yield optimized results.

There is always a scattering of logic operations over arithmetic nodes making the data flow complex. There are many sorting techniques available which sort the dataflow graphs such that the logic operations are separated from the arithmetic operations. These arithmetic nodes are combined in a systematic way to obtain compact dataflow graphs without any loss in its original functionality. The compressor trees are used to combine the arithmetic nodes which have the potential to reduce the number of arithmetic / logic nodes needed. The most commonly used compressor tree is the Wallace-like compressor tree. The next optimization algorithm is the Common Subexpression Elimination. It leads to numerical transformation of constant multiplication leading to efficient hardware utilization and increased speed.

The optimization algorithms discussed provide the following merits compared to the conventional methods:

1) Decrease in Critical path delay
2) Reduction in the overall area
3) Increased computation speed
4) Efficient reutilization of resources

The tools used for this purpose are MATLAB for Common Subexpression Elimination and Turbo C for Compressor trees. A set of various orders of FIR filter are used for implementing the Common Subexpression Elimination method using MATLAB. This is then followed by application of Compressor trees after taking the data flow graph through various steps of sorting. This is coded using C language. The resulting data flow graph is synthesized using High level synthesis tool – SPARK and area report is obtained using Quartus II. The benchmark circuits used for Compressor tree technique are Elliptic Wave Filter (EWF), Wave Digital Filter (WDF) and MPEG Motion Vector (MPEG-MV). Future work comprises of applying both the techniques in a benchmark data flow graph and obtained area optimized designs.

Section II deals with Common Subexpression Elimination and then a discussion on the Compressor tree technique in section III. An implementation of Compressor tree and Common Subexpression Elimination along with its comparison with conventional methods is given in section IV. Results and discussion are also presented here.

II. COMMON SUBEXPRESSION ELIMINATION

Strength reduction at the algorithm level reduces the number of additions and multiplications in arithmetic circuits. One such numerical technique is called Sub expression elimination [1]. This technique improves speed, power and area of the circuit greatly. This strength reduction reduces the total capacitance and therefore reduces the power consumption. Sub expression elimination method is used over expressions which have a set of common multiplicands. It identifies recursive occurrences of identical bit patterns that are present in the coefficients using *Iterative Matching Algorithm* [2]. It reduces the number of redundant multiplications and thus maximizes the use of a common expression multiple times to obtain the desired design. Here the number of shifts and additions required for multiplication is effectively minimized.

The Iterative matching algorithm used in implementing CSE uses the following steps:

- Each constant in the set is expressed in binary format or Canonic Signed Digit format.
- The number of bitwise matches between each pairs of the constants in the set is determined. Only non-zero matches are considered because only they have the potential to increase the number of adders/shifters.
- The best match is chosen.
- The redundancy from the best match is eliminated. The remainders and the redundancy are returned to the set of coefficients.
- The number of bitwise matches continues to be determined until no more improvement is achieved.

The applications of Common Subexpression Elimination are as follows:

A. Common Subexpression Elimination in Linear problems

As already discussed before, the Subexpression elimination can be applied to constant multiplications. Therefore we can extend its use to linear transformation also. The multiplicands are expressed in binary format and then the iterative matching algorithm is applied to extract the common subexpressions.

While considering Linear Transformations, the sub expression elimination problem consists of three basic steps:

- The number of shifts and adds required to compute the product, t_{ij} x_j are minimized using Iterative matching algorithm.
- Unique products are formed using the sub expressions determined in the previous step.
- The additions are shared among the various y_i's thus reducing the number of individual hardware and process time required for computing the additions.

B. Common Subexpression Elimination in polynomial evaluation:

In a similar way, Sub expression elimination can be applied to polynomial evaluation also. This technique is best suited to reduce the computational complexity [3].

For e.g., considering the polynomial shown below,

$$a^7 + a^4 + a^2 + a \quad (1)$$

A conventional realization of the above expression would require 10 multiplications. However on careful examination we can notice that the number of multiplications required can be significantly reduced. For eg. a^7 as $a^4 * a^2 * a^1$. Therefore after applying sub expression evaluation, the polynomial evaluation problem effectively reduces as:

$$a^2 * (a^4 * a) + a^4 + a^2 + a \quad (2)$$

The terms a^2, a^4 and a^8 each require only one multiplication:

$$a^2 = a * a, \quad a^4 = a^2 * a^2 \quad (3)$$

Thus we see that on exploiting the redundancy which is not conceptually very difficult we arrive at a much simpler polynomial evaluation problem.

C. Common Subexpression Elimination in filters:

The idea of Common Subexpression Elimination is made use of in FIR filters to reduce the hardware complexity of the filter implementation [4], [5]. In order to realize the sub expression elimination transformation for an N-tap filter given by the following expression,

$$y(n) = h_0 x(n) + h_1 x(n-1) + \ldots + h_{N-1} x(n-N+1) \quad (4)$$

must be realized using its transposed direct form structure. This type of structure is also referred to as a broadcast filter structure. With this structure, one variable is multiplied to multiple constant coefficients. Sub expression elimination can then be applied. To systematically obtain the sub expression elimination, a filter operation is represented in a matrix form. The rows are indexed by delay i and the columns by shift j. The row and column indexing starts at 0. The entries in the table are 0 or 1 if two's complement representation is used (except the sign bit) or they are from the digit set {0, 1, -1} if CSD is used.

An improvement over Common Subexpression Elimination is Binary Common Subexpression Elimination. This technique uses binary representation [0 1] instead of the Canonic signed digit representation [-1 0 1] in Common Subexpression Elimination [6]. The other variants of Common Subexpression Elimination are *Vertical Common Subexpression Elimination (VCSE)* and *Horizontal Common Subexpression Elimination (HCSE).* The VCSE technique identifies subexpressions between different coefficients

whereas the HCSE identifies subexpressions within the same coefficient. An example of VCSE and HCSE is shown in Fig. 3 and 4 respectively.

D. Illustration of CSE in Digital Filters:

E.g. y(n)= 1.0001`00000*x(n) + 0.1`01`010010*x(n-1) + 0.000100001`*x(n-2)

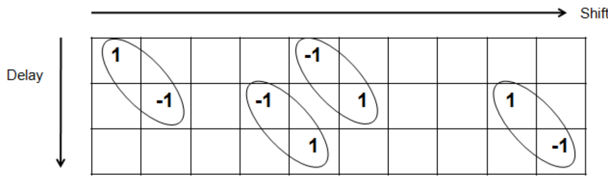

Fig. 1 Common Subexpression Elimination in Digital Filters

Subexpression: x2 = x1 – x1[-1] >> 1
y = x2 – (x2 > > 4) – (x2[-1] >> 3) + (x2[-1] >> 8)

E.g. y(n) = 1.000í00010 * x(n) + 1.000í00010 * x(n-1) + 0.0001000í0 * x(n-2)

VCSE:

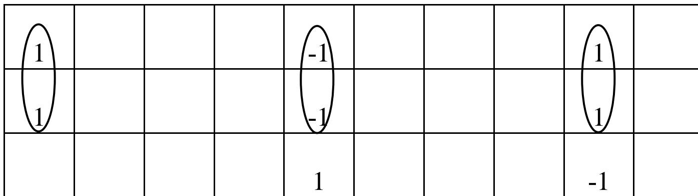

Fig. 2 Vertical Common Subexpression Elimination

Subexpression: x2 = x1 + x1[-1]
y(n) = x2 – (x2 >> 4) + (x2 >> 8) + (x1[-2] >> 4) – (x1[-2] >> 8)

HCSE:

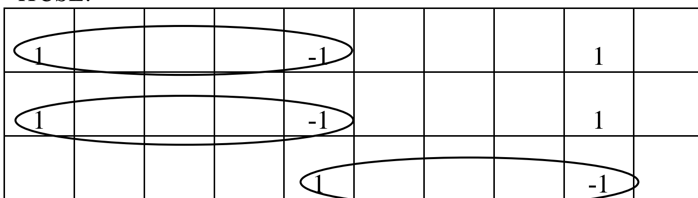

Fig. 3 Horizontal Common Subexpression Elimination

Subexpression: x2 = x1 – (x1 >> 4)
y(n) = x2 + x2[-1] + (x2[-2] >> 4) + (x1 >> 8) + (x1[-1] >> 8)

The CSE technique is followed by the Compressor tree method. A large number of data flow transformations are applied before applying compressor trees. This is because better efficiency is obtained in terms of area after such transformations.

III. COMPRESSOR TREES

The idea of parallel addition and multiplication was a major breakthrough from the conventional approach. It helped to save a lot of time though there was so save in the resources utilized. The processor was never idle in such a situation.

Fig. 4(a) shows a conventional approach for adding 18 numbers. It is noted that it takes 16 time steps to obtain the result. In the modified tree height reduction approach shown in Fig. 4(b), it takes only 7 time steps to finish the same operation. This method uses parallel addition using carry save adders to compute and produce faster results.

It is evident from Fig. 4(b) that the parallel addition method holds well as long as all the inputs arrive at more or less equal times. Therefore such an architecture would be best suited if all its inputs were primary inputs. However if we intend using such an architecture in the middle or towards the end of a data flow graph, then the input arrival time poses a major threat to the usage of such a technique. In the worst case, it can even produce the same results as conventional approach (Fig. 4(a)). Therefore knowledge of input arrival times is very important for working out a solution towards the same.

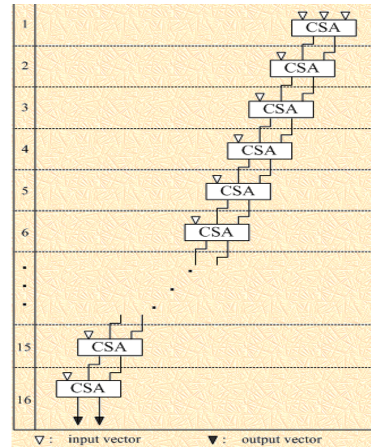

Fig. 4(a) Conventional approach to add 18 numbers

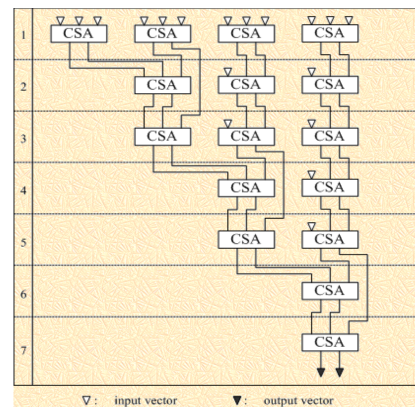

Fig. 4(b) Parallel architecture for adding 18 numbers

A. *Three-greedy technique:*

The Three greedy technique is a heuristic to reduce the vertical slice [7]. The steps are shown in Fig. 5. A set, T is defined for each vertical slice of the graph. It contains the arrival delay of all input bits initially. In each step, three smallest values are chosen from T and a full adder is used to calculate the sum and carry from the corresponding inputs. Then the three values are removed from the set T. The sum of the bits is now added to the set while the carry is added to the set T of the next vertical slice. The process is repeated until the set T contains the final three elements in it. These three inputs are again fed to a full adder which calculates the final sum and carry. The assumption under this process is that the difference between the arrival times of the input signals is not extremely large and the use of compressor trees is beneficial. The three greedy techniques essentially checks if any addition operation can be performed before the next input arrives. This way we can use the hardware efficiently.

The processor design requires a large scatter of logic operations over the arithmetic operations. It is important to sort these nodes in such a manner that they are more understandable. It then becomes easy to apply optimization techniques to this sorted dataflow graph. Compressor trees are then applied to suitable arithmetic nodes after careful analysis. There are two points to be observed before applying compressor trees. The first is that the function of the design before and after optimization should remain the same. The second is that the application of compressor trees should provide some advantage compared to the traditional design.

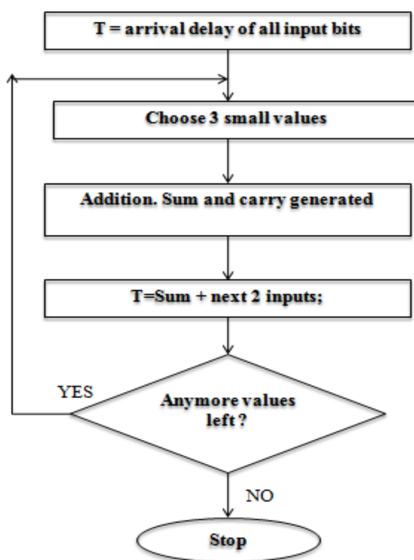

Fig. 5 Three Greedy technique

A compressor tree is shown in Fig. 6. It is basically a circuit which takes more than 3 inputs and produces 2 outputs, sum and carry [8].

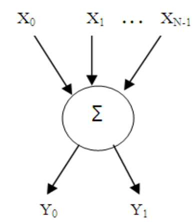

Fig. 6 General form of a Compressor tree

Though the compressor trees efficiently reduce the area when compared to the traditional designs, there arises a problem when there is a drastic difference in the arrival times between its various inputs. Therefore they make use of Three greedy technique for overcoming this problem.

The steps involved in the application of compressor trees are briefed in the flowchart shown in Fig. 7 ([9], [10]).

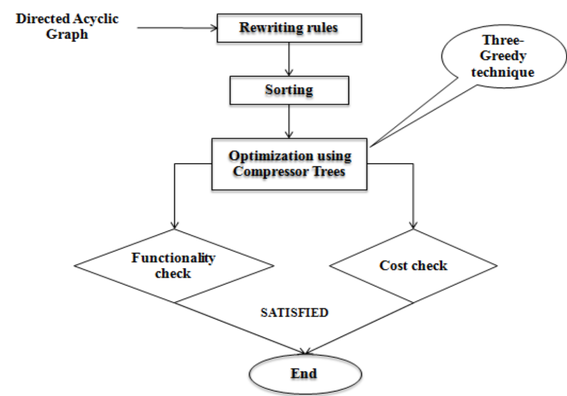

Fig. 7 Steps involved in the application of compressor trees

B. *Steps involved in Compressor tree application:*

The steps involved in the Compressor tree technique are summarized below. The scattered natures of the logic and arithmetic nodes are the main target.

1) Rewriting rules: The first step under this consists of rewriting all the arithmetic operations in terms of additions so that whole of the arithmetic circuit consists of only adders. The table 1 gives a list of rewriting rules. The first operation computes the negative by an integer by finding the 2's complement of the integer. The second operation generates the difference of two integers by calculating the sum of one of the integers and the 2's complement of the next integer. The third rule implements a parallel multiplier. PP ( ) generates a set a partial products and sum ( ) represents addition of the partial products which will be implemented using the compressor trees. The fourth operation calculates the relation between two operands by finding the difference between the two and then checking the sign bit.

After this step the dataflow graph is rearranged in such a way that multi-input adders are incorporated as much as possible. Then we replace a multi-input adder with a

compressor tree followed by a single carry-propagate final adder.

TABLE I
REWRITING RULES

From	To
-X	$X^C + 1$
X – Y	$X + Y^C + 1$
X * Y	SUM (PP(X,Y))
X (relation) Y	F_{REL} (SIGN (X – Y))

2) Sorting rules: This is an important step before applying compressor trees. Sorting of a dataflow graph is done to improve the effectiveness of the carry save representation. After rewriting the dataflow graph according to Table 1, the resulting directed acyclic dataflow graph is represented as G (V, E). Here nodes V's represent primitive operations and edges E's represent data dependences. The function Ord (.) returns the position of a node in the ordering. The nodes can be in one of the two classes: Arithmetic (A) where all the nodes consist of only adders as they have already been rewritten using Table 1. The function class (.) returns the class of a node. There is an important fact to note. A sorted graph is sufficient though not absolutely necessary to be able to produce an optimal implementation with the use of compressor trees. Table 2 summarizes transformation rules for advancing class L operations over class A operations.

The table gives a clear idea that in only some cases swapping of nodes is possible. The last sorting rule i.e. advancing a Partial Product (PP) node over addition is based on the distributive property of multiplication over addition but is considerably more complex than the other sorting rules. It is to be noted that unlike the other sorting rules, this rule is sometimes not quite practical as it imposes a major cost in terms of hardware. Therefore a careful analysis has to be performed before applying this rule for it to be beneficial. After applying the last sorting rule, for any PP node, there exists an addition node such that from all outputs of the PP node, there are identical paths of logic nodes to the addition node.

TABLE II
POSSIBILITIES FOR ADVANCING CLASS L OPERATIONS OVER CLASS A OPERATIONS

Operations	Can be advanced over addition
Bitwise NOT	Yes
Multiplication	Yes
Partial Product generator	Yes
Selector	Yes

3) Application of Compressor tree: After applying the rewriting rules, all the arithmetic nodes have been transformed to include only adders. After advancing the appropriate Logic nodes over the addition nodes, we can get a cluster of addition nodes. We can then apply multi-input adders for all independent nodes. This method reduces the number of adders thereby reducing the hardware utilization of the design. By applying the three-greedy technique also, we can obtain efficient time utilization.

4) Functionality and Cost check: There are 2 checks to be satisfied before applying the compressor trees. They are the functionality check and the cost check. The Functionality check accounts for the fact that the basic function of the design should not be compromised due to the application of any number of optimization techniques. The next check reveals if there is any advantage either in terms of area or in terms of speed after applying the compressor trees. It is necessary to identify *Useful Movable nodes* for this purpose. These are nodes which produce an advantage when moved.

IV. RESULTS AND DISCUSSION

Appropriate benchmark circuits shown in Table 3 were taken and the rewriting and sorting rules were applied to them to obtain a sorted Data flow graph with a cluster of addition nodes towards the end. These nodes can then be replaced by a multi input adder which improves speed and reduces the hardware utilization. Some of the benchmark circuits were implemented using this technique and results were obtained as shown in Table 3. High level synthesis tool – SPRAK is used for this purpose. It takes C file as input and generates a VHDL output. This HDL file can then be simulated using ModelSim and synthesized using Quartus II. Thus the area report is obtained which can then be compared with the conventional method.

TABLE III
COMPARISON BETWEEN ORIGINAL DFG AND SORTED DFG AND THE SAVE IN AREA OBTAINED USING HIGH LEVEL SYNTHESIS TOOL – SPARK

Benchmark circuit	No of nodes in original	No. Of nodes in	Area saving (in %)
EWF	35	29	12.2
WDF	35	26	15.7
MPEG-MV	33	23	17.28

A conventional FIR filter whose coefficients have been specified in Canonic Signed Digit representation was implemented and the number of adders required was noted. The same filter was implemented using Vertical Common Subexpression Elimination and Horizontal Common Subexpression Elimination techniques using Iterative Matching Algorithm and the results were tabulated as shown below. Matlab was used to implement the algorithms.

TABLE IV
COMPARISON CHART ON THE HARDWARE UTILIZATION BETWEEN VARIOUS TYPES OF FIR FILTERS USING MATLAB

Filter order	Coefficients	No. of Adders utilized		
		Conventional	**VCSE**	**HCSE**
3	C0=1.000Ī00010 C1=1.000Ī00010 C2=0.00010000Ī	7	5	5
4	C0=Ī010Ī00010 C1=0.Ī01000000 C2=0.010Ī00010 C3=1.000010Ī00	11	9	8
5	C0=Ī010Ī00010 C1=0.Ī01000000 C2=0.010Ī00010 C3=1.000010Ī00 C4=0.00010000Ī	13	11	10

Common Subexpression Elimination technique was implemented for various FIR filter orders in a high level synthesis tool – SPARK using C language. It was synthesized using Quartus II, implemented using FPGA device – Cyclone III. The following results were obtained.

TABLE V
LOGIC OPERATOR AND REGISTER UTILIZATION OF FIR FILTERS USING HIGH LEVEL SYNTHESIS TOOL – SPARK

FIR filter order	Logic operators used	Flip flops / latches
10	120	45
20	243	91
28	312	103
35	401	165
55	560	217

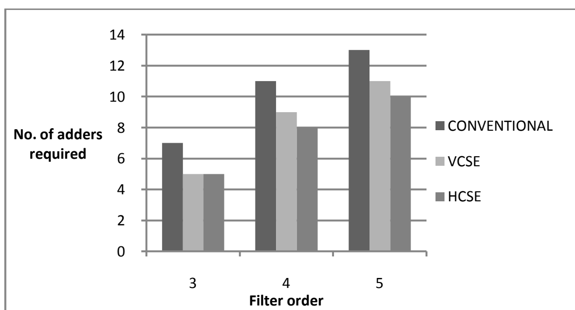

Fig. 8 Hardware requirement of Conventional, VCSE and HCSE type FIR filters

V. CONCLUSION

The following conclusions were derived after implementing Compressor trees and Common Subexpression Elimination methods:

1) Using the compressor tree method, an average of about 15% save in area was obtained for 3 benchmark circuits (Elliptic Wave Filter, Wave Digital Filter, MPEG Motion Vector).
2) The Common Subexpression Elimination method was implemented for various FIR filter orders and it was concluded that it required atleast 2 adders less than that required in the conventional implementation. It was also derived that the method would yield better results as the order of the filter is increased.

The above Optimization algorithms when used together in a single benchmark circuit will provide the advantages of both the methods – Decrease in area and reduction in critical path delay in case of Compressor trees and increase in reutilization of hardware resources through Common Subexpression Elimination. The above along with the Distributed Arithmetic method integrated together on the benchmarks and the FIR filters is envisaged as the future work.

ACKNOWLEDGEMENT

We would like to thank the management of Amrita Vishwa Vidyapeetham for providing us the required infrastructure and lab facilities for our work.

REFERENCES

[1] Mahesh, R.; Vinod, A.P. (2008): *A New Common Subexpression Elimination Algorithm for Realizing Low-complexity Higher order Digital Filters*, *IEEE Trans. Comput. Aided Des.* Vol 27, No. 2, pp 217-229.
[2] Keshab K.Parhi, : *VLSI Digital Signal Processing Systems, Design and Implementation*, *ISBN: 978-81-265-1098-6*.
[3] Sivaram, G.; Priyank, K. (2009): *Algebraic techniques to enhance Common Sub-expression Elimination for polynomial system synthesis*, *Electronic design Automation Association*, pp 1452 – 1457.
[4] Pasko, R.; Schaumont, P.; Derudder, V.; Vernalde, S.; Durackova, D. (1999): *A new algorithm for elimination of common subexpressions*, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 1, pp. 58–68.
[5] Anup, H.; Farzan, F.: *Common Subexpression Elimination involving multiple variables for Linear DSP synthesis*, *15th IEEE International Conference on Application Specific Architectures and Processors (ASAP).*
[6] Hartley, R.I. (1996): *Subexpression sharing in filters using Canonic signed digit multipliers*, *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.* vol. 43, no. 10, pp. 677–688.
[7] Stelling, P.F.; Martel, C.U.; Oklobdzija, V.G.; Ravi, R. (1998): *Optimal circuits for parallel multipliers*, *IEEE Trans. Comput.,* vol. 47, no. 3, pp. 273–285.
[8] Verma, K.; Ienne, P. (2008): Data flow *transformations to maximize the use of carry save representation in arithmetic circuits*, *IEEE trans, Comput.-Aided Des.* Vol. 27, No. 10, pp 1761-1774.
[9] Kim, T.; Jao, W.; Tjiang, S. (1998): *Circuit optimization using carry-saveadder cells*, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.,* vol. 17, no. 10, pp. 974–984.
[10] Oklobdzija, V.G.; Villeger, D.; Liu, S.S. (1996): *A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach*, *IEEE Trans. Comput.,* vol. 45, no. 3, pp. 294–306.