High Speed and Power Efficient Compression of Partial Products and Vectors

Chiuwei Pan[1], Zhao Wang[2], Carl Sechen[3]

[1, 2, 3]University of Texas at Dallas, 800 W Campbell Rd, Richardson, TX 75080

[1]cxp053000@utdallas.edu; [2]zxw061000@utdallas.edu; [3]cms057000@utdallas.edu

***Abstract*-A high speed and power efficient compression algorithm for an arbitrarily shaped array of partial products and vectors is presented. Since the full-adder cell is a cornerstone for the carry-save adder (CSA) tree, the most power efficient full-adder cells for building CSA trees for a wide range of delays were ascertained. A minimum hard-ware usage algorithm for an arbitrarily shaped array of vectors was developed. Finally, a new delay-based adder-type selection and CSA-tree wiring algorithm is proposed. This new compression network synthesis (CNS) algorithm was tested on several industrial DSP blocks for a variety of process technologies. CNS produces DSP blocks that consume 20-40% less power for the same delay (throughput) compared to other state-of-the-art designs and compared to the leading commercial synthesis tool.**

Keywords- Partial Product Compression; DSP Networks; Networks of Additions and Multiplications; Accelerators

I. INTRODUCTION

Datapaths and functions in digital signal processing (DSP) systems are getting more and more complex, and therefore implementing them using a general-purpose CPU is neither energy efficient nor high performance. Thus custom ASIC accelerators or DSP processors are often used in today's high performance computing systems. DSP functions are mostly comprised of networks of additions and multiplications, which are the bottleneck of the system performance and dominate the power consumption. In order to maximize the performance of the complex DSP processor, a system level synthesizer will extract the arithmetic datapath into the largest possible sum-of-product (SOP) blocks and reuse the intermediate carry-save results extensively, reducing the number of expensive carry propagation adders [1-3]. Avoiding carry-ripples and introducing more parallelism are the key objective in designing a high performance DSP system.

After each SOP block is determined, the combined arithmetic function, consisting of rows of vectors as well as partial products resulting from multiplications, is aligned into a compression matrix (CM) [4]. The CM needs to be synthesized into a compression network comprised of various types of gates, including full and half adders. The compression network is implemented by large Carry-Save Adder (CSA) trees which perform the compression in a bitwise parallel process. The objective of the CSA tree is to reduce or compress the number of rows (a row being a vector or partial product) down to two rows.

Several methods for compressing partial products and vectors have been published. These methods are based on various Carry-Save Adder (CSA) arrays with basic elements such as full-adders (FAs), half-adders (HAs) and lower-order compressors (e.g., 4:2 compressors). A Wallace tree [5] compresses partial products using log3/2(N/2) levels of FAs, where N is the number of rows to be compressed. A Dadda tree [6] minimizes the adder usage. Use of higher order compressors has also been proposed. The method in [7] reduces the partial products by using log2 (N/2) levels of 4:2 compressors. A fast 5:3 compressor design [4] requires log5/3(N/3) levels of 5:3 compressors and a final FA. These methods all utilize fixed-order compressor architectures to implement the partial product reduction tree. (Fixed-order means that each bit-weight or column is assumed to have the same number of bits to compress.) On the other hand, the three-dimensional reduction method (TCM) in [8] creates directly N-th order compressors for each partial product bit-weight since multiplications usually result in the need to compress varying numbers of bits, depending on the bit weight. Each N-th order compressor is timing-driven connected using the input arrival time information.

As described previously [4-8], the basic elements for a compression network's CSA array are FAs and HAs. In order to design a fast or power efficient compression network, the following must be ascertained:

1) The number of adders to employ (and whether they should be FAs or HAs)
2) The best gate-level structure for each required adder
3) The best way to select and interconnect the various adders for all of the CSA trees

In this paper, we propose an algorithm that computes the minimum possible hardware usage for any irregular CM shape. We also provide the first reported timing-driven adder-type selection and interconnection algorithm. These two compression network synthesis (CNS) algorithms target both high speed and power efficient implementations. The proposed CNS designs are compared with those by the leading commercial synthesis tool using cell-based design techniques. The ability to specifically optimize DSP functions was activated for the commercial synthesis tool, and all of its pre-designed CSA blocks were made available. After the compression process, the final carry-propagation adder (CPA) is a bit-wise serial operation.

Experiments have shown that the leading commercial synthesis tool is quite adept at power efficient designs of CPAs for arbitrary input bit arrival times by varying the structure [1, 2]. Thus CPA design is not addressed here.

The paper is organized as follows: Section II shows the structure of the three leading types of full adders as well as half adders, and Section III selects the best adder structures for CSA trees. In Section IV a minimum adder usage algorithm and a novel timing driven adder selection and interconnection algorithm are proposed. Section V shows results for a CSA array of 16 16b vectors, a second-order digital IIR filter and a H.264 video interpolator core.

II. THREE TYPES OF FULL ADDERS AND HALF ADDER

A. Full and Half Adder Structures

A full adder (FA) has three input bits of equal weight and two output bits, sum and carry, of the same and one greater binary bit weight, respectively. A half adder (HA) receives two input bits of equal weight and the output bits are the same as for a FA. Thus, there are two major parts in both the FA and HA: the carry generator (CARRY-GEN) and the sum generator (SUM-GEN). According to the dependency between CARRY-GEN and SUM-GEN, we divide FAs into three different types.

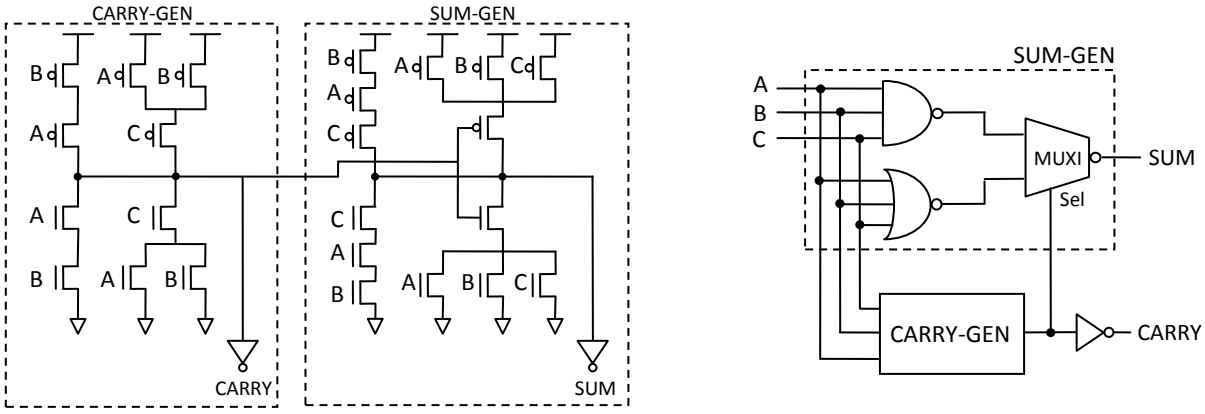
1) Type 1: SUM-GEN is dependent on CARRY-GEN
2) Type 2: CARRY-GEN is dependent on SUM-GEN
3) Type 3: SUM-GEN and CARRY-GEN are independent

Because the logic of a HA is simple, a HA usually has independent SUM-GEN and CARRY-GEN.

In this paper, the focus is on ASIC designs generated by an automated design flow that utilizes a commercial standard cell library for synthesis and timing analysis. In order to correctly ascertain the worst-case delays of contemporary circuit blocks, it is necessary to utilize static timing analysis (STA) tools. The inputs for a cell must be purely capacitive in order to use STA tools [9] and therefore some adder circuits from the literature must be slightly modified in order to ascertain the correct worst-case delay for an entire circuit block.

B. Three Types of Full Adders

Type 1 FA: SUM-GEN is dependent on CARRY-GEN. A commonly used type 1 FA has the structure shown in Fig. 1a. This FA structure is also called the Mirror FA. Another structure is proposed in [10], shown in Fig. 1b. This is termed the MUX-FA. With this type 1 FA architecture, any delayed input arrival (A, B or C) will increase both the sum and the carry delay. However, the carry is generated faster than the sum. Thus, the type 1 FA structure is best when all inputs arrive at the same time and when benefit can be gained from the smaller carry delay. An example of CSA trees constructed using only Mirror FAs is given in [11].

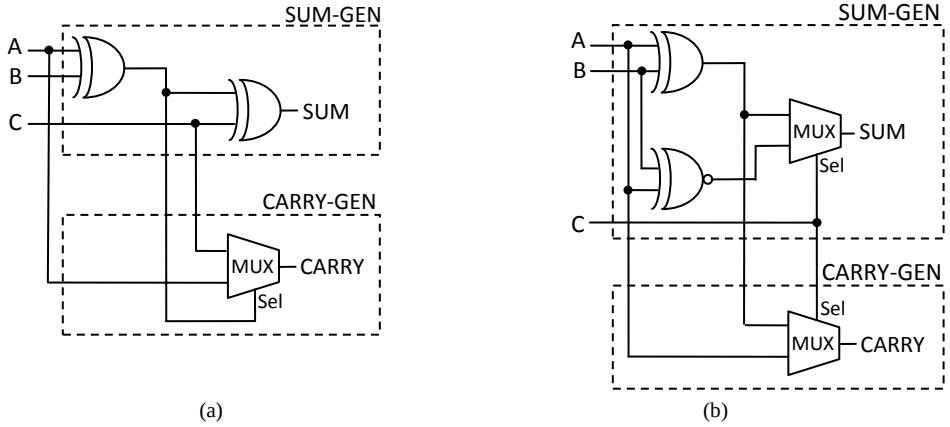


(a) (b)

Fig. 1 Common type 1 FAs (a) mirror FA (b) MUX-FA

Type 2 FA: CARRY-GEN is dependent on SUM-GEN. The logic gate representation is shown in Fig. 2 Type 2 FAs have been extensively researched. In [12] the centralized FA is claimed to benefit from the use of complementary pass-transistor logic (CPL) to implement the XOR-XNOR module while also using static CMOS logic for better drivability. In [13] and [14], the Hybrid-CMOS FA architecture is claimed to be more energy efficient than other type 2 FAs. In this paper, the Hybrid-CMOS FA architecture proposed in [14] was evaluated. In order to adopt this hybrid-CMOS FA into a cell-based standard library and to use it in an STA framework, it was slightly modified as shown in Fig. 3. Because these hybrid-CMOS FAs are connected back-to-back in the CSA tree structure, this modification of moving inverters from the outputs to the inputs will not significantly alter the published performance. For type 2 FAs, assuming the MUX2 delay is similar to an XOR2 delay [4] and [15], if one of the three inputs is delayed within an XOR2 delay, where the delayed input must connect to C in Fig. 2, the sum

and the carry delay will not be affected by the delay of input C. However, both sum and carry are generated at about the same time even if all inputs arrive at the same time. The type 2 FA structure is best utilized when one of three inputs has about an additional XOR2 delay in arrival compared to the others, as was illustrated in the simulation setup for a CSA tree in [13]. The connection method proposed in [16] systemically creates this input delay difference and then only uses type 2 FAs to construct CSA trees.



(a) (b)

Fig. 2 General form of type 2 FAs: (a) XOR-XOR-based, (b) centralized FA

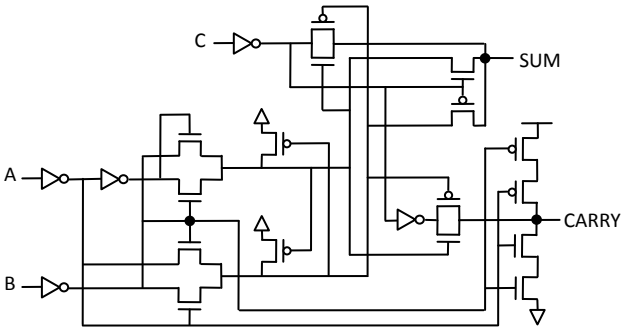
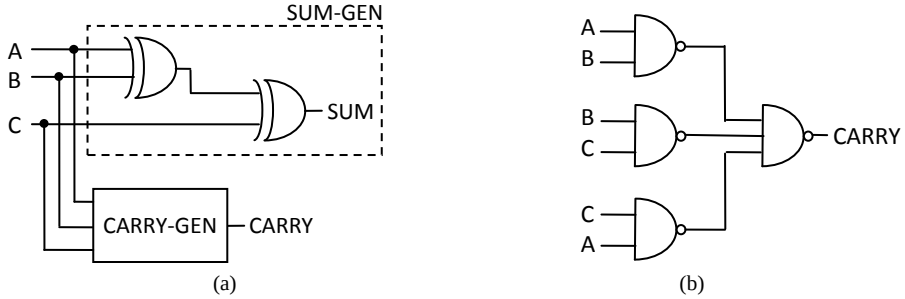


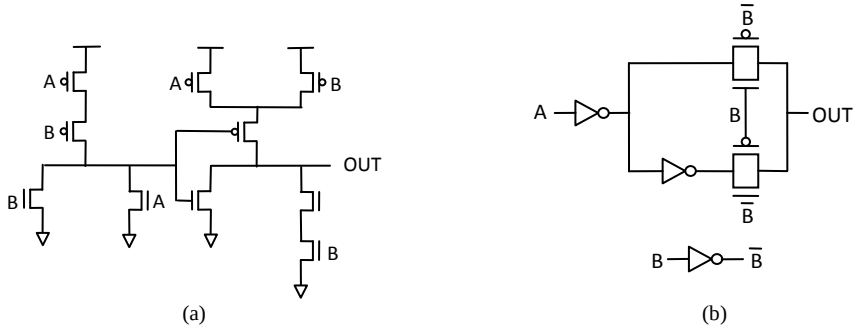
Fig. 3 Hybrid-CMOS FA with purely capacitive input loads

The type 3 FA has independent SUM-GEN and CARRY-GEN; the logic gate representation is shown in Fig. 4a. The CARRY-GEN function is also known as the majority (MAJ) function that can be implemented by four NAND gates shown in Fig. 4b, or the CARRY-GEN in Fig. 1a, referred to as mirror carry (MC) in this paper. The SUM-GEN can be implemented with the structure in Fig. 2a. There are two popular ways to implement the XOR2 gate. The first is shown in Fig. 5a, which uses 10 transistors in the static CMOS style (sXOR). The other utilizes pass transistor logic (PTL). An STA compatible PTL XOR2 is shown in Fig. 5b (pXOR). This design also employs 10 transistors. The two XOR2 circuits in Fig 5a and Fig. 5b are commonly found in a commercial standard cell library. Because of the independent sum and carry generation architecture, SUM-GEN has the same benefit as a type 2 FA in that it can tolerate a C input delay within an XOR2 delay without affecting the SUM delay. CARRY-GEN has the same benefit as type 1 FA in that it generates the carry faster than the sum if all inputs arrive at the same time. Thus, the type 3 FA combines the delay benefits from both the type 1 and type 2 FAs.



(a) (b)

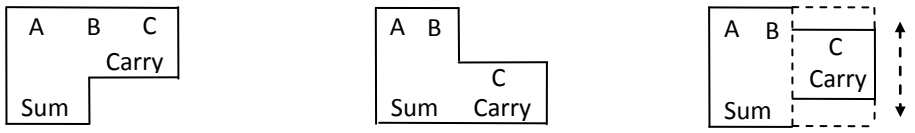
Fig. 4 (a) logic gates of type 3 FA (b) NAND-based CARRY-GEN



(a) (b)

Fig. 5 (a) static XOR2 gate (b) PTL XOR2 with purely capacitive input loads

In summary, the characteristics of different FAs can be represented via delay tiles [16]. The delay tile representation of each FA delay is given in Fig. 6. Only the type 3 FA has an adaptive CARRY-GEN delay, meaning that if an input is up to an XOR delay late, then the carry output will also be delayed by the same amount. In order to determine the best FA structure for building the CSA tree, the most commonly used and state-of-the-art FA structures were selected for each type of FA. The mirror FA in Fig. 1a and MUX-FA in Fig. 1b were selected to represent the type 1 FA. The Hybrid-CMOS FA in Fig. 3 was selected for the type 2 FA. Because of the variety of possible XOR2 cell designs, both Fig. 5a and Fig 5b XOR2 cells are chosen for the SUM-GEN of the type 3 FA. Both NAND-based carry and mirror carry are selected for the CARRY-GEN for the type 3 FA. The relative structural com-parisons of these selected FAs are listed in Table 1.



(a) (b) (c)

Fig. 6 Tile representation of cell delay: (a) type 1, (b) type 2 and (c) type 3 FA

TABLE 1 FULL-ADDER CELLS

Type	Structure	Transistor Count
1a	Mirror FA	28
1b	MUX-FA	34
2	Hybrid-CMOS FA (Fig. 2.3)	26
3a	2 (NOR2+AOI12) + MC	32
3b	2 (NOR2+AOI12) + NAND-based carry	38
3c	2 PTL XOR2 + MC	32

C. Half Adder Structure

The HA is also used in building a CSA tree. The positioning of HAs in a CSA tree will be discussed in Section IV. A HA only has two input bits of equal weight and two output bits, sum and carry, of the same and one greater binary bit weight, respectively. Thus, the SUM-GEN of a HA is equal to an XOR2 gate and the CARRY-GEN of a HA is an AND2 gate. The structure of the XOR2 gate can be either static CMOS as in Fig 5a or PTL as in Fig. 5b. After the comparison in Section III, we will select the best XOR2 structure for both the FA and HA.

D. Other Adder Structures

In the early work on carry-save adder networks, FAs and HAs were implemented less efficiently (in terms of delay, area and power) using only NOR and NAND gates [17, 18]. The FA structure proposed in [17] used 7 NOR3 and 1 NOR2 gates. Implemented in static CMOS, this adder requires 46 transistors and is 40% larger than the power efficient mirror FA. FAs implemented with NOR or NAND gates are not as power efficient for the same delay as the XOR-based (with mirror carry) and mirror-sum, mirror carry. The optimization technique in [18] is based on minimizing the number of NOR or NAND gates and interconnection pins. This technique may be applicable for FPGAs.

III. HIGH SPEED AND POWER EFFICIENT FULL ADDER SELECTION FOR CSA TREE

A. Selecting the Test Circuit

The delay characteristics of a FA used in a ripple adder chain are quite different from a FA used in a CSA tree. The critical path of a ripple adder is dominated by the delay of CARRY-GEN because for an N-bit ripple adder the max-delay is N CARRY-GEN delays and one SUM-GEN delay. However, the CSA tree requires both fast CARRY-GEN and SUM-GEN, because the critical path of the CSA tree varies according to its structure and may proceed through multiple CARRY-GEN and multiple SUM-GEN cells. In order to find the most power efficient FA for building a CSA tree, the CPA delay is neglected and the only focus is on the delay of the compression of partial products and vectors. The critical path of the partial product compression in a single multiplication is different from an SOP block with combined arithmetic functions. Take a 6-to-2 compression process for example; the 6x6 multiplication shown in Fig. 7a created critical paths close to the middle region [5-7], and the center 6-to-2 CSA tree accepts 2 passed carries and 6 bits from partial products. However, a 6x6 CSA array has uniform critical paths after the third bit as shown in Fig. 7b. In this case, the 6-to-2 CSA tree used for the critical paths accepts 3 passed carries and 6 bits from input vectors. Generally, the worst case for a 6-to-2 compressor occurs when it accepts 3 passed carries; thus, we select a CSA array as our testing circuit. (A 6-to-2 compressor can also accept more than 3 passed carries but the delay is dominated by the arrival time of the slowest passed carry and it will pass four or more carries to the next weight instead of three). In [16], the critical path of the 16b 4-operand CSA with final carry-propagate adder is dominated by the 16b ripple adder, as illustrated by the fact that the CSA delay is less than 25% of the CPA delay. Thus, the delay in this case is dominated by carry propagation alone. In contrast, we want to find out which FA is the most power efficient for use strictly in a CSA array.

Fig. 7 Critical path of (a) 6x6 multiplication, (b) 6x6 CSA array

Using different types of FAs greatly impacts the delay of a CSA tree. In order to demonstrate the differences, a 9-to-2 compressor was selected, as shown in Fig. 8a. For a high-speed (indeed, power efficient for this high speed) compressor structure, we need to connect FAs by the timing driving method (TDM) proposed in [8] which only illustrated using type 3 FAs to build the entire CSA tree. Figure 8b shows the timing driven connected 9-to-2 compressor using type 3 FAs only; FAs used in the CSA tree are represented delay tiles. Timing driven connected 9-to-2 compressors built using type 1 FAs only and type 2 FAs only are shown in Fig. 8c and Fig. 8d, respectively. The red dotted paths in Fig. 8 illustrate one possible critical path for each CSA tree structure. At the logic level, the delay of SUM-GEN is roughly twice the delay of CARRY-GEN. This assumption was also adopted in [4], [15] and [16]. In other words, SUM-GEN is roughly two unit delays and CARRY-GEN is roughly one unit delay. With this assumption, the critical path of the 9-to-2 compressor for the type 1, type 2 and type 3 FAs is 7, 8 and 6 unit delays, respectively. Although the unit delay of the critical path for the compressor using type 2 FAs is 25% longer than the compressor using type 3 FAs, the compressor using type 2 FAs can still be faster than the compressor using type 3 FAs if the type 2 FA is 25% faster than the type 3 FA. The total unit delays for the critical path of N-to-2 compressors using the three types of FAs are shown in Fig. 9. Since the type 2 FA structure inherently delays the carry output, more unit delays are accumulated in the critical path. The type 3 FA has the overall minimum unit delay in the critical path because the carry output is adaptively generated according to the incoming input delays. The proposed simulation setup for a CSA tree in [13] which connects two sum outputs and one carry output to a FA will benefit the type 2 FA but slow down the type 1 and type 3 FAs. In order to fairly compare FAs in a CSA tree, each CSA tree of the compression network needs to be custom created for each type of FA. In this paper, a 16-vector, 16b CSA array is selected as the test circuit and the structure is timing driven connected separately for each type of FA shown in Table 1.

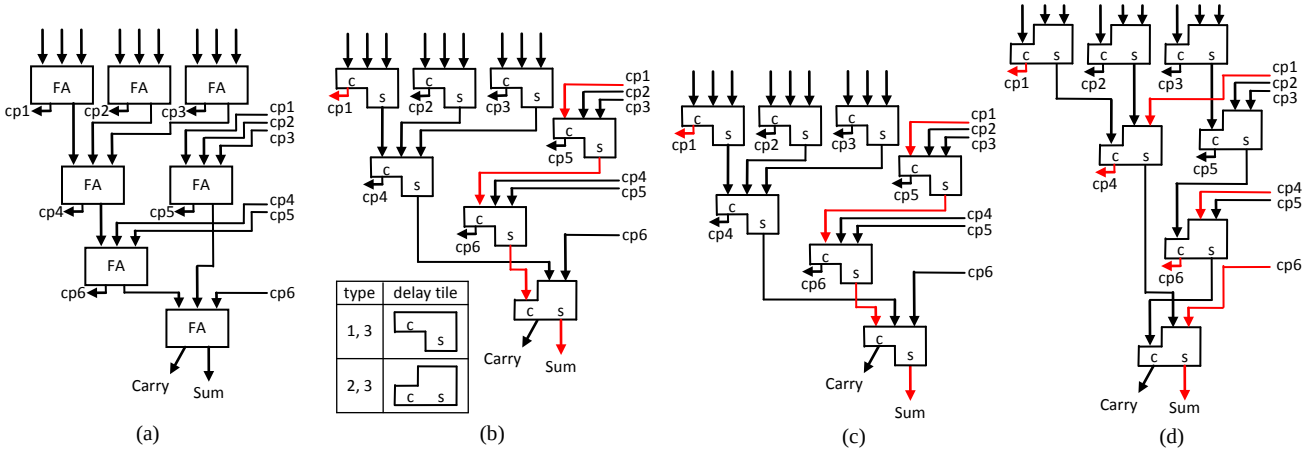


Fig. 8 6-to-2 compressor: (a) general form and delay tile representation with (b) type 3 FA only, (c) type 1 FA only and (d) type 2 FA only

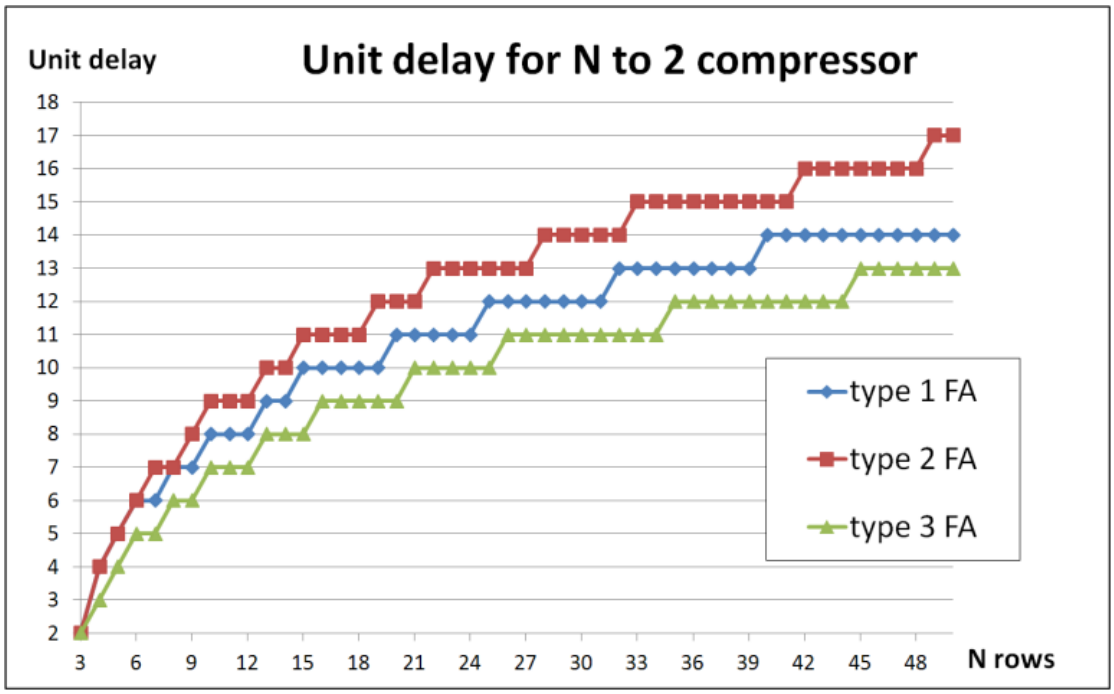


Fig. 9 Unit delay for *N*-to-2 compressors using each FA type, for $N = 3 \sim 50$ rows

B. Simulation Environment

All of the FAs in Table 1 were made available as a separate cell in the characterized standard cell library, and each was made available for a wide range of drive strengths and pull-up to pull-down strength, or beta ratios. In addition, many other types of cells, including INV, NAND, NOR, AOI and OAI cells, were available in many drive strengths and beta ratios. An industrial 130 nm 1.2 V process and Synopsys' Design Compiler were used to synthesize the CSA arrays for a wide range of delay targets, in order to systematically produce a power versus delay plot. Parasitic capacitances and a nominal wire load model were included. The synthesis began with a large delay target and then the delay target was iteratively decreased until the synthesizer failed to reach the target. In order to correctly determine the worst-case path for a complete circuit block (such as a CSA array), we used Synopsys' Prime Time to report the worst-case path delay. Cells were characterized using Synopsys' Liberty NCX. Power results were obtained by simulation with Hspice using 500 random input vectors, since the average power converged in all cases within that number of vectors.

C. Results and Analysis

The most effective way to compare different logic networks for power efficiency is by means of power versus delay plots, since the power efficiency of achieving a particular delay is of crucial significance. In Fig. 10, it is apparent that whatever technique was used to produce curve A is clearly superior to the technique used to produce curve B and C. Without using power versus delay curves, it is impossible to assess the relative power efficiency of different circuit implementations. When two curves intersect, as do curves B and C in Fig. 10, it is possible to specify delay ranges where one technique outperforms

the other.

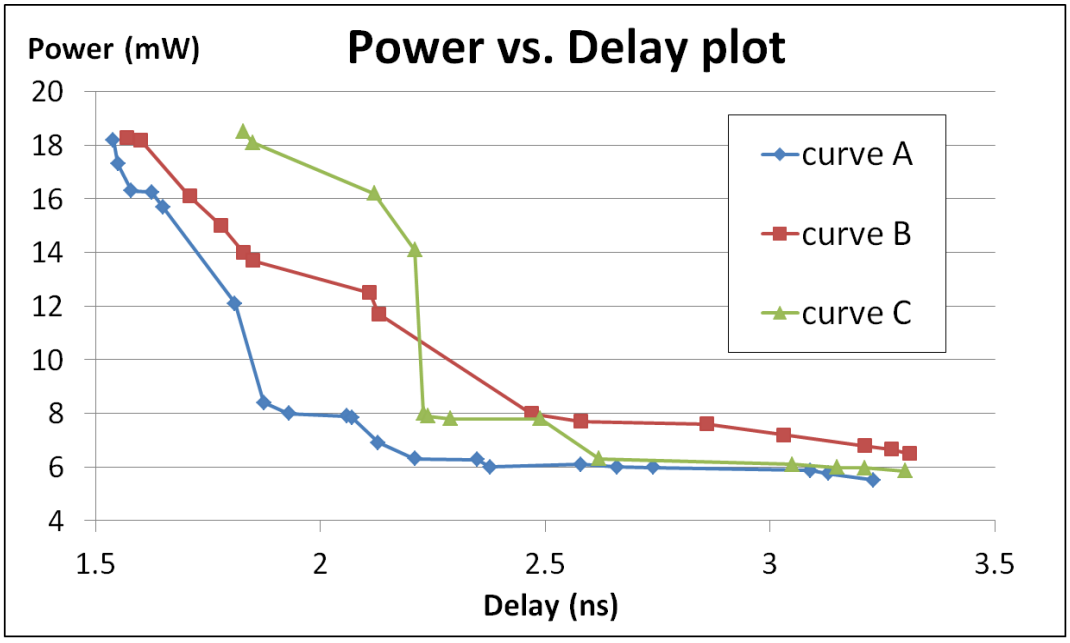


Fig. 10 Power versus delay curves prove that circuit block B and C are of no value compared to block A. However, for long delays, block C is more power efficient than block B

The simulation results for the 16x16b CSA array using each type of FA shown in Table 1 are illustrated in Fig. 11. It is apparent that NAND-based carry (type 3b FA) should not be used (sXOR-NAND). Since the type 2 FA structure inherently delays the carry output, which adds unit delays in the critical path, it is not a suitable structure for high-speed or power-efficient compression network design. The dominant power-delay curves are different for high and low speed regions. In the high-speed region, both type 3a (sXOR-MC) and type 3c (pXOR-MC) FAs have similar performance. In the low power and low-speed region, both type 1a and type 1b FAs are comparable. In order to finalize the selection, the area-delay curves were plotted separately for these four adders in Fig. 12. For high-speed adders, the type 3a FA is 5% smaller in cell area compared to the type 3c FA. Although the type 1b FA saves 8% more power than type 1a FA, the type 1b FA uses 27% more cell area than the type 1a FA. Thus, only type 1a (Mirror FA in Fig 1a) and type 3a (2 cascaded static XOR2 gates with mirror carry) FAs are chosen as the building blocks for our compression network synthesis (CNS) algorithm described in the next section.

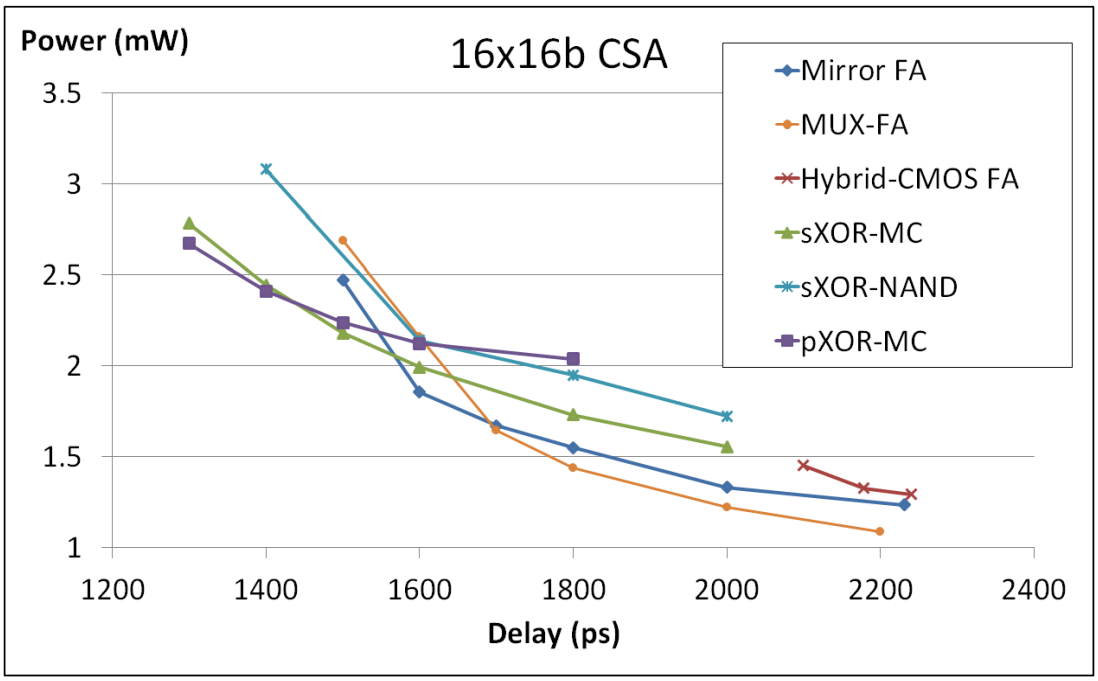


Fig. 11 Power vs. delay plots for the 16-vector, 16b CSA array constructed using Mirror FA (type 1a), MUX-FA (type 1b), hybrid-cmos FA (type 2), static XOR2 with mirror carry (sXOR-MC, type 3a), static XOR2 with nand based carry (sXOR-NAND, type 3b) and PTL XOR2 with mirror carry (pXOR-MC, type 3c)

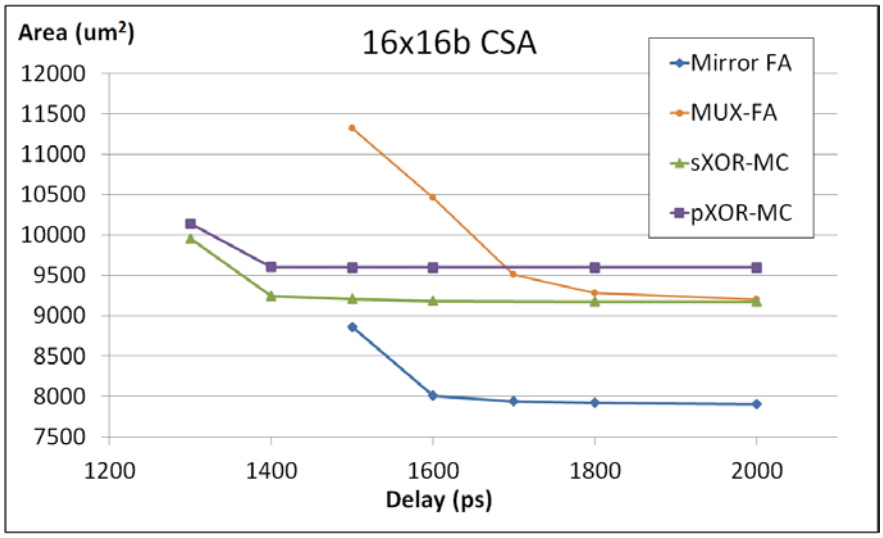


Fig. 12 Area vs. delay plots for the 16-vector, 16b CSA array constructed using Mirror FA, MUX-FA, static XOR2 with mirror carry (sXOR-MC) and PTL XOR2 with mirror carry (pXOR-MC)

IV. CSA TREE BUILDING ALGORITHMS

A. *CSA Tree Hardware Usage Algorithm*

The shape of the CM for a single multiply operand usually resembles a parallelogram. However, it is usually the case that the shape of the CM from combined functions is more irregular. The prior work in the area of CSA tree connection algorithms primarily has focused on a single multiplication or a multi-ply-accumulate (MAC) function [1-5]. In order to build an efficient tree for any irregularly shaped CM, we propose a new adder usage algorithm. The compression network is constructed on a column-by-column basis (from right to left). The compressor for a given column (of equal bit weight) is determined by the number of carries passed into this column (Cin), the number of bits in this column of the CM (Xin) and how many locations (D) we can drop compressed sum bit(s) into. We derived (1), which shows N, the total bits we need to eliminate (compress out) in this column.

$$N = Xin + Cin - D \tag{1}$$

Note that since the compression is down to two rows, D is either one or two. The latter results when there is an empty slot from an unused dropped carry from the column immediately to the right. A full adder (FA) takes three bits and creates one bit in the same column, therefore two bits are eliminated. On the other hand, one bit is eliminated by using a half adder (HA). If N is an odd number, a single HA will be used. The total number of FAs used in the tree is N/2 if N is even or (N-1)/2 if N is odd. An interconnected 8-to-2 compressor structure is shown in Fig. 13. The numbers represent the unit delays at each node (here shown in terms of unit delays, although we use explicit delays in our actual implementation).

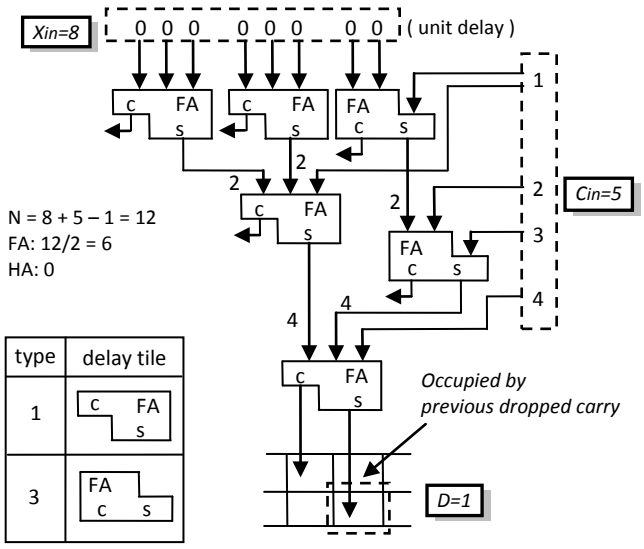


Fig. 13 8-to-2 compressor tree structure with unit delay information

B. Hardware Complexity of the Proposed Algorithm

Although (1) is applicable to all possible shapes for a CM, in order to compare to prior FA usage algorithms for a CSA tree it was necessary to select multiplication as the basis for comparison. Based on the proofs in [8], the CSA structure of a multiplier can be divided into two regions for an unsigned n by n bit multiplication with partial products generated by AND gates. The CM for the multiplication has a regular parallelogram shape as shown in Fig. 14. As shown in Theorem 1 and 2, the total adder usage is therefore equal to (n-1)(n-2). Take a 12b multiplier for example, the proposed method uses 99 FAs and 11 HA matching the same FA and HA count as Dadda's tree [6]. However, unlike prior approaches, the new adder usage algorithm based on (1) handles irregular CM's that often have multiple drop locations, D.

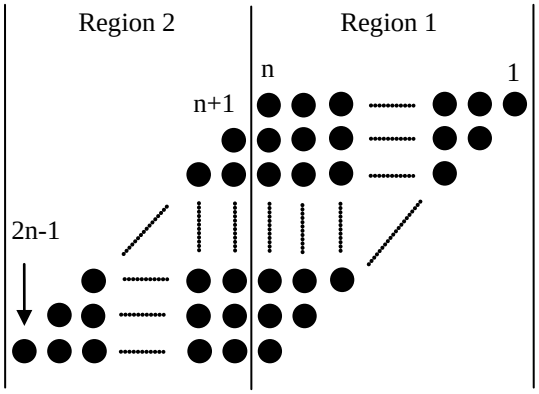


Fig. 14 CM of unsigned NxN bit multiplication

Theorem 1: the adder usage in region 1 is

Proof:

From (1), N = Xin + Cin – D, and thus:

bit 1: Xin= 1, Cin= 0, D =2. $N_{(1)}$ = -1 < 0, No adder needed

bit 2: Xin= 2, Cin= 0, D =2. $N_{(2)}$ = 0, No adder needed

bit 3: Xin= 3, Cin= 0, D =2. $N_{(3)}$ = 1, FA:0, HA:1

bit 4: Xin= 4, Cin= 0, D =1. $N_{(4)}$ = 3, FA:1, HA:1

…..

bit n: Xin= n, Cin= n-4, D =1. $N_{(n)}$ = 2n-5, FA:n-3, HA:1.

From bit 3 to bit n, gate usage increases by one for each bit position. At bit n, the gate usage is (n-2). Thus:

$$\text{Adder usage} = \frac{(1+(n-2))(n-3+1)}{2} = \frac{(n-1)(n-2)}{2}$$

Theorem 2: the adder usage in region 2 is

Proof:

From (1), (N = Xin + Cin – D):

bit n+1: Xin= n-1, Cin= n-3, D =1. N(n+1) = 2n-5, FA: n-3, HA:1

bit n+2: Xin= n-2, Cin= n-3, D =1. N(n+2) = 2n-6, FA: n-3, HA:0

bit n+3: Xin= n-3, Cin= n-3, D =1. N(n+3) = 2n-7, FA: n-5, HA:1

…..

bit 2n-2: Xin= 2, Cin= 1, D =1. N(2n-1) = 2, FA:1 HA:0

bit 2n-1: Xin= 1, Cin= 0, D =1. N(2n) = 0, No adder needed.

From bit n+1 to 2n-2, gate usage decreases by one for each bit position. At bit 2n-2, the adder usage is 1. Thus:

$$\text{Adder usage} = \frac{((n-2)+1)((2n-2)-(n+1)+1)}{2} = \frac{(n-1)(n-2)}{2}$$

C. Timing Driven Tree Construction Algorithm

After the adder usage is determined, the selection of FA structures and connections between the adders is determined next. A delay-optimized tree is created according to the input arrival times as well as the input pin-to-output pin delays of the adders, and in addition the best FA structure for each adder in the tree is determined. Initially all inputs, Xin and Cin in (1), are sorted in ascending order of delays (arrival times) and put into a connection list. If N is an odd number, a HA will be inserted into the tree first in order to gain the benefit from using the faster sum and carry generation provided by a HA. In this case the first two inputs in the connection list (i.e., the two with the earliest arrival times) will be taken out and connected to the HA. The generated sum output will put back into connection list and the carry becomes an input at the next higher weight (immediately to the left). Each tree has at most one HA. Before we connect FAs, the connection list will be updated and sorted in ascending order again (if necessary). Similar to the HA connection, the first three inputs in the connection list will be taken out and connected to a FA. First the delay information is examined for these three inputs. If one delay is at least a unit delay later than

the other two, a type 3 FA will be connected to the tree; otherwise a type 1 FA will be chosen. The carry and sum outputs of the FA are treated in the same manner as for the HA. The process iterates until all adders are connected. The compression network synthesis (CNS) algorithm is given in Fig 15. Compared to TDM [8], CNS has been generalized to handle any irregularly shaped CM, and in addition to performing timing driven connections as [8] does, the new algorithm also selects the best FA circuits to use.

```
Algorithm: Tree construction algorithm (TCA) and FA selection
Inputs: Xin, Cin, D, CP;   // CP is the set of carries passed in
connection_list = { Xin, Cin };
Calculate adder usage (N = Xin + Cin – D)
if ( N is odd)
        FA_usage = (N-1) / 2
        HA_usage = 1
else
        FA_usage = N / 2
        HA_usage = 0
Put X_in and CP into connection list
Sort (connection_list)   // in increasing order, using actual delays
if (HA_usage = 1)
        Take out first two inputs in connection list and connect to a HA, put SUM back to
        connection list and send CARRY to the next weight
while (FA_usage > 0) do
                Sort (connection_list)
        Extract first three inputs from the connection list
        if (one delay is at least an unit delay larger)
                Connect a type 3a FA
                The third FA input receives the latest arriving input
        else
                Connect a type 1a FA
        Put SUM back into conn. list; send CARRY to next weight
        Decrement FA_usage
```

Fig. 15 Compression Network Synthesis (CNS) algorithm

All of the constituent inverting gates of a FA, for example, SUM-GEN and CARRY-GEN for the type 1a FA, as well as its inverters, are sized independently by the synthesis tool for a particular delay target and using the sizes available in a cell-based standard cell library. Similarly, the constituent in-verting gates comprising the XOR2's (Fig. 5a) and mirror carry gate, as well as its inverter, of the type 3a FA are sized independently. By optimizing the selection of FA structures, optimizing the components of these FA structures, as well as the wiring of those FAs, substantial reductions in power and delay can be achieved compared to the leading commercial synthesis tool.

V. DESIGN EXAMPLES AND COMPARISON

A. 16-vector 16b CSA Array

The power-versus-delay results for a 16-vector 16b CSA array using the CNS algorithm were obtained using an industrial 130 nm 1.2 V library with a nominal wire load model. As before, the average power was obtained by simulation with Hspice with 500 random input vectors, since the average power converged in all cases with this number of vectors (or fewer). In order to correctly determine the worst-case path for a complete circuit block (such as a CSA array), we used Synopsys' Prime Time to report the worst-case path delay. Cells were characterized using Synopsys' Liberty NCX. For comparison, we synthesized the 16-vector 16b CSA array (with the result in carry-save format) using the best scripts of the leading commercial synthesis tool. In addition, designs using only type 1a FAs (Mirror FA) or only type 3a FAs (sXOR2-MC) were also generated. Note that the Mirror FA result is equal to [11] which only uses the Mirror FA to build the entire compression network and the sXOR-MC result is equal to [8] which only uses type 3 FAs.

The results are shown in Fig. 16, where the red squares represent the optimized commercial synthesis results. Our CNS results are given by the orange curve. The power reduction for the same delay is about 25% on average over the common delay range compared to the commercial synthesis tool. The main reason for this improvement is our new proposed CNS algorithm which makes opportunistic use of the type 3a FA when input arrival times differ sufficiently, and otherwise uses the more area efficient type 1a FA.

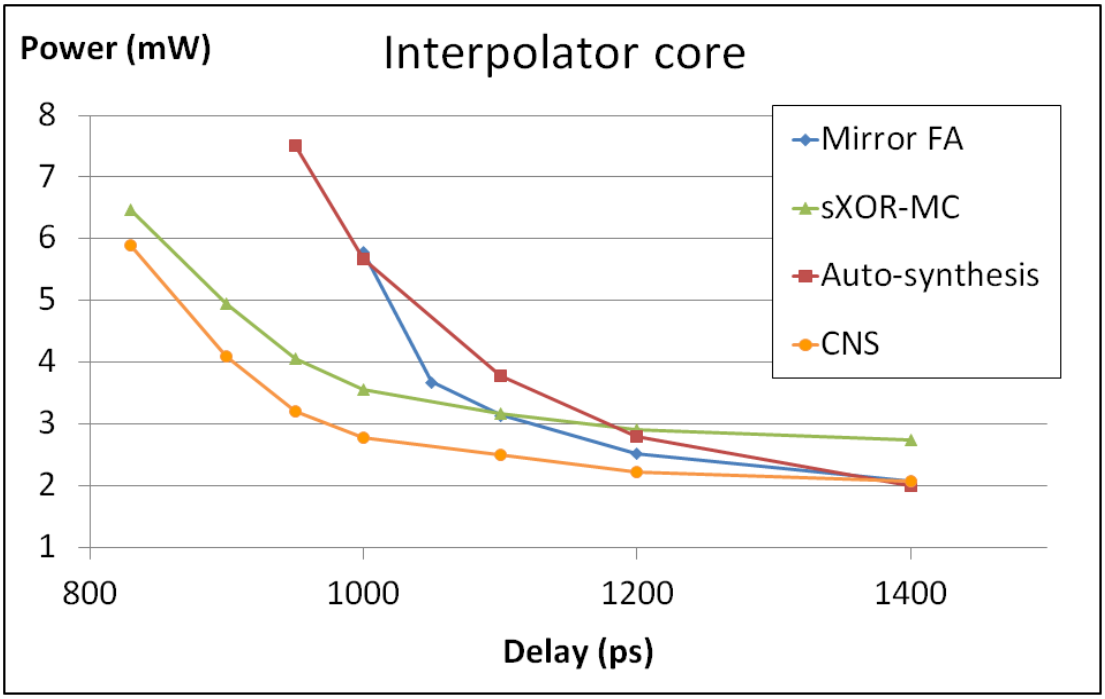


Fig. 16 Power vs. delay plot for a 16-to-2 CSA array constructed using: Mirror FA, static XOR2 with mirror carry (sXOR-MC), commercial synthesis tool (Auto-synthesis) and the CNS algorithm

B. Second-Order Digital IIR Filter

A second-order section (SOS) of a digital Infinite Impulse Response (IIR) filter is shown in Fig. 17 [19]. To maintain stability with the use of fixed precision arithmetic, the filter employs a quantizer (Q). The expression for the output y (n) can be written as

$$y(n) = x(n) + c_1 y(n-1) + c_2 y(n-2) \tag{2}$$

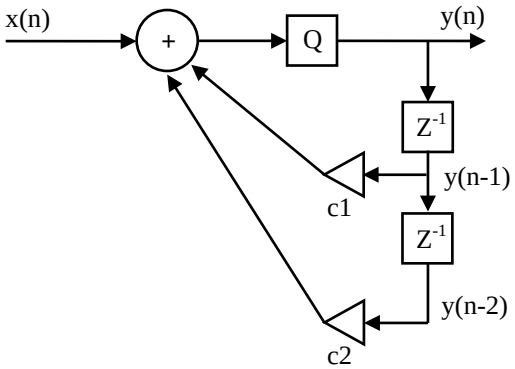


Fig. 17 Second-order IIR digital filter

In (2) c1 and c2 are coefficients, x (n) is the input vector, and y (n-1) and y (n-2) are feedback vectors. The input vector for the filter x(n), the coefficients c1, c2, and the output y(n) are specified in fixed-point arithmetic, in 2's complement, in the form of (t, w). Here t is the number of integer bits and w is the number of fractional bits. The input x (n) is in the format (1, 21). The output y (n) is in the format of (13, 15). The coefficients c1, c2 are in the format of (2, 16), with values: 1.98416137695313 and -0.99127197265625 as given by a leading semiconductor manufacturer.

Fig. 18 shows the CM for the second-order IIR filter design. To compare the efficiency, a second-order IIR filter with the same coefficients was designed in behavioral Verilog and synthesized using the leading commercial synthesis tool along with its best scripts. An industrial 40 nm 0.9 V technology was used, along with a nominal wire load model. The results comparing our CNS algorithm with the leading commercial synthesis tool are shown in Fig. 19. The results are reported by Synopsys' Prime Time with the given industrial 40 nm standard library. For this design, our CNS algorithm yielded an average power reduction (over the common delay range) of 37% for the same delay compared to the leading commercial synthesis tool.

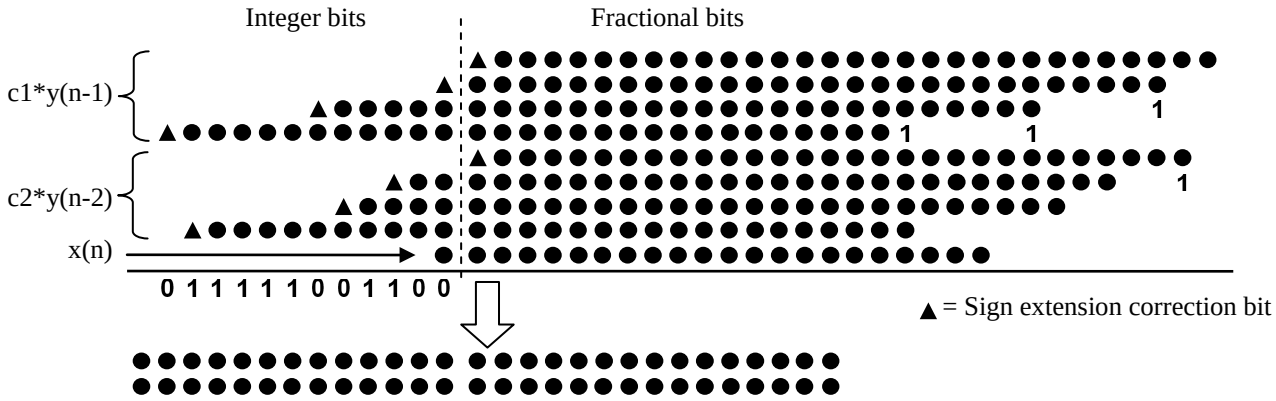


Fig. 18 CM of a second order IIR filter

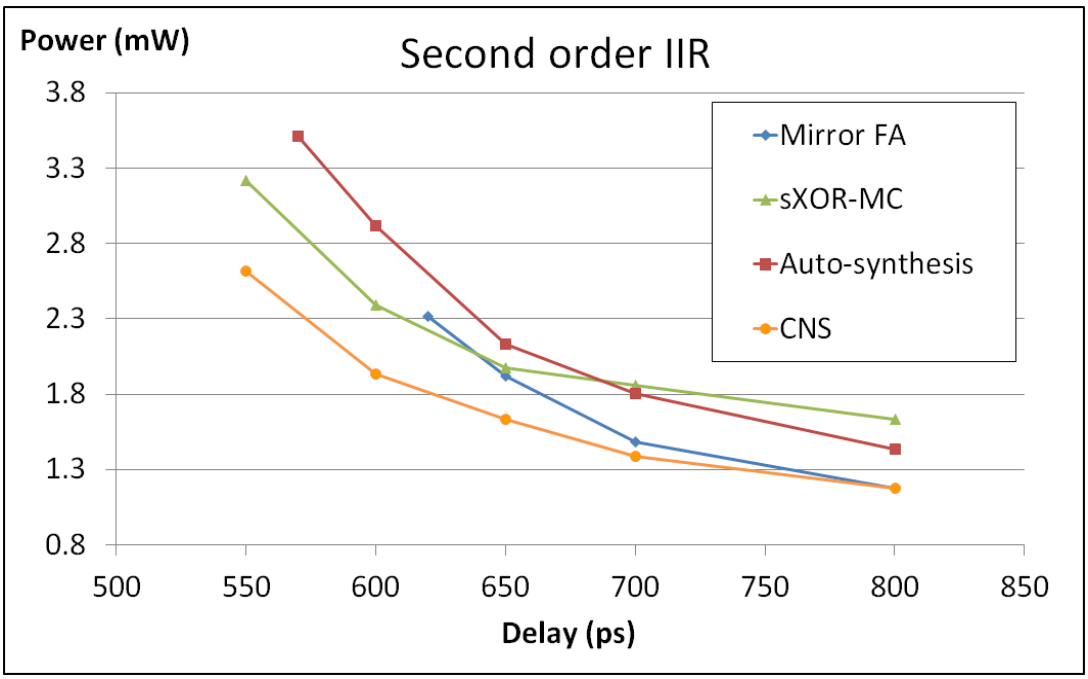


Fig. 19 Power vs. delay plots for a second order IIR filter design

Using only type 3a FAs (green triangles in the Fig. 19) yields fast designs but they are not power efficient. The mirror FA (blue squares) is good for very low power but not competitive speed-wise. Again, the main reason for the improvement by our CNS algorithm is that it exploits the difference in FA arrival times whenever possible to gain a speed advantage and when not possible, it uses the most area-efficient mirror FA.

C. Pixel Interpolator for H.264/AVC Decoder

The pixel interpolator is used for video coding standards with sub-pixel accuracy [20]. With this ability, High Definition (HD) video can be transmitted with a smaller bit rate. The interpolator is one of the most power-consuming modules in the H.264 codec. In order to design an energy efficient interpolator, we utilized the CNS algorithm for the interpolation filters, combined with an effective exploitation of parallelism. H.264/AVC uses a 6-tap Wiener interpolation filter [21] to generate half-pixels from the reference integer pixels [20]. Pixels in the reference image are 8-bits for the basic H.264/AVC. All values are represented in 2's compliment arithmetic. For example, the unit for interpolating the half-pixel position h, named filter 1, are given by (3) and (4). A, C, G, M, R and T are reference integer pixels in Fig. 20.

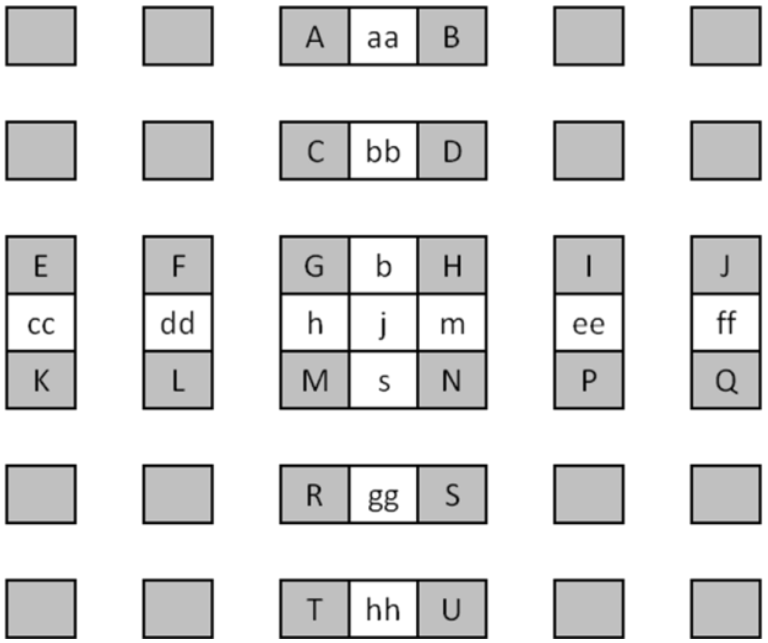


Fig. 20 Reference pixels (shaded blocks with upper-case letters) and fractional-pixel positions (un-shaded blocks with lower-case letters) for fractional-pixel interpolation

$$h1 = (A - 5*C + 20*G + 20*M - 5*R + T) \quad (3)$$

$$h = Clip((h1 + 16) >> 5) \quad (4)$$

h1 is the intermediate value of h. The value of h1 plus 16, right-shifted by 5 digits (divided by 25), is sent to a Clip () function. The Clip () function quantizes the intermediate value to the data range of 0 to 255. All intermediate values less than zero will become zero, and those larger than 255 will become 255. The unit for interpolating the center fractional-pixel position j is named filter 2, whose equations are given by (5) and (6). This center fractional-pixel j is created by intermediate half-pixel values, cc1, dd1, h1, m1, ee1 and ff1 which are generated by (3) with corresponding reference pixels in Fig. 20.

$$j1 = (cc1 - 5*dd1 + 20*h1 + 20*m1 - 5*ee1 + ff1) \quad (5)$$

$$j = Clip((j1 + 512) >> 10) \quad (6)$$

The center fractional-pixel calculation passes through two consecutive filters and comprises the critical paths for both filter 1 and filter 2, as shown in Fig. 21. The intermediate carry-save result generated from the filter 1 is passed to filter 2. With this architecture, the CPA between filter 1 and filter 2 is eliminated.

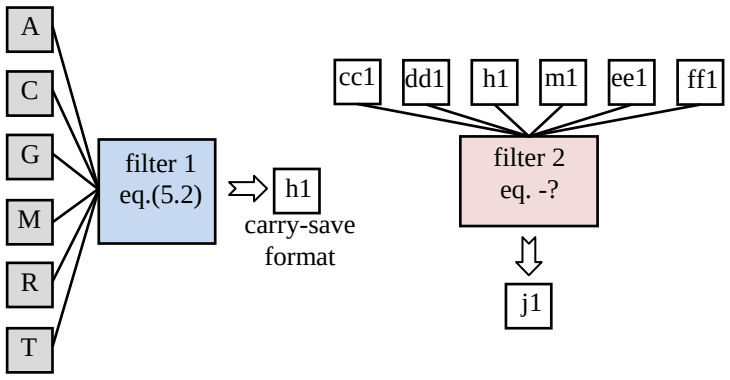


Fig. 21 Critical path for generating center fractional-pixel

The CM for (3) and (5) are illustrated in Fig. 22 and Fig. 23. The interpolator core was designed using an industrial 40 nm 0.9 V technology, along with a nominal wire load model. The results are reported by Synopsys' Prime Time, using an industrial 40 nm standard library. For comparison, this interpolation core was also designed in behavioral Verilog and synthesized with the leading commercial synthesis tool along with its best scripts. Compared to the result from the leading commercial synthesis tool, the proposed CNS design was both faster and lower power as shown in Fig 24. For the same power

of 3 mW, the CNS design is 28% faster. For the same delay of 1 ns, the CNS design reduces the power by 55%.

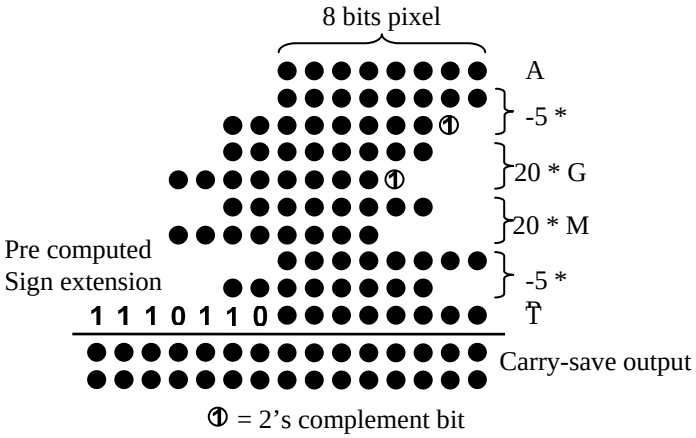


Fig. 22 CM of filter Eq. 3? please reorder the Equations

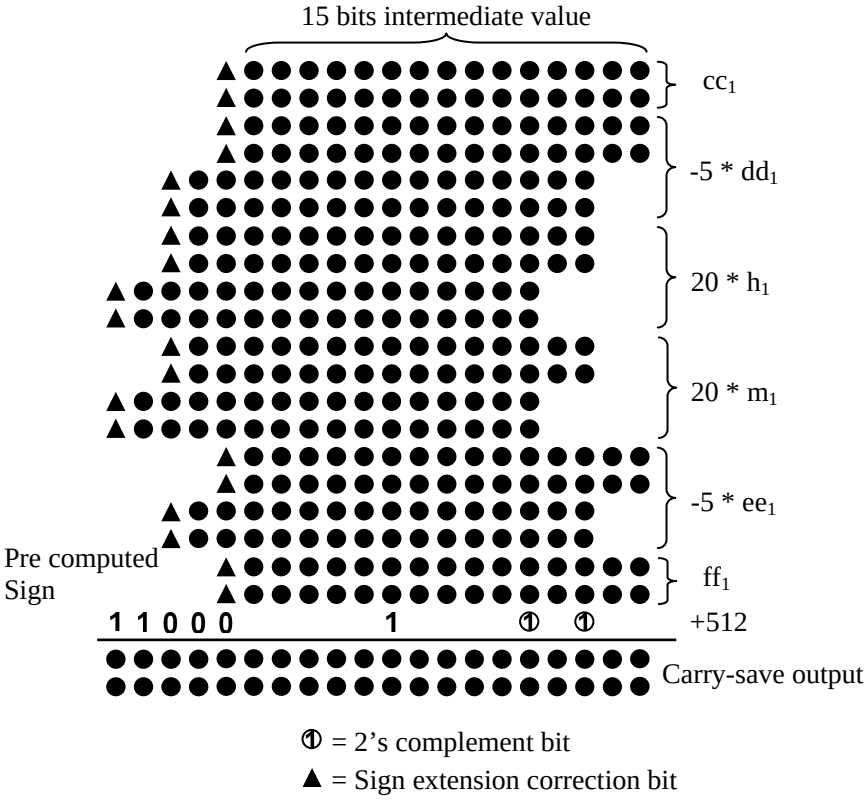


Fig. 23 CM of filter Eq. ?5

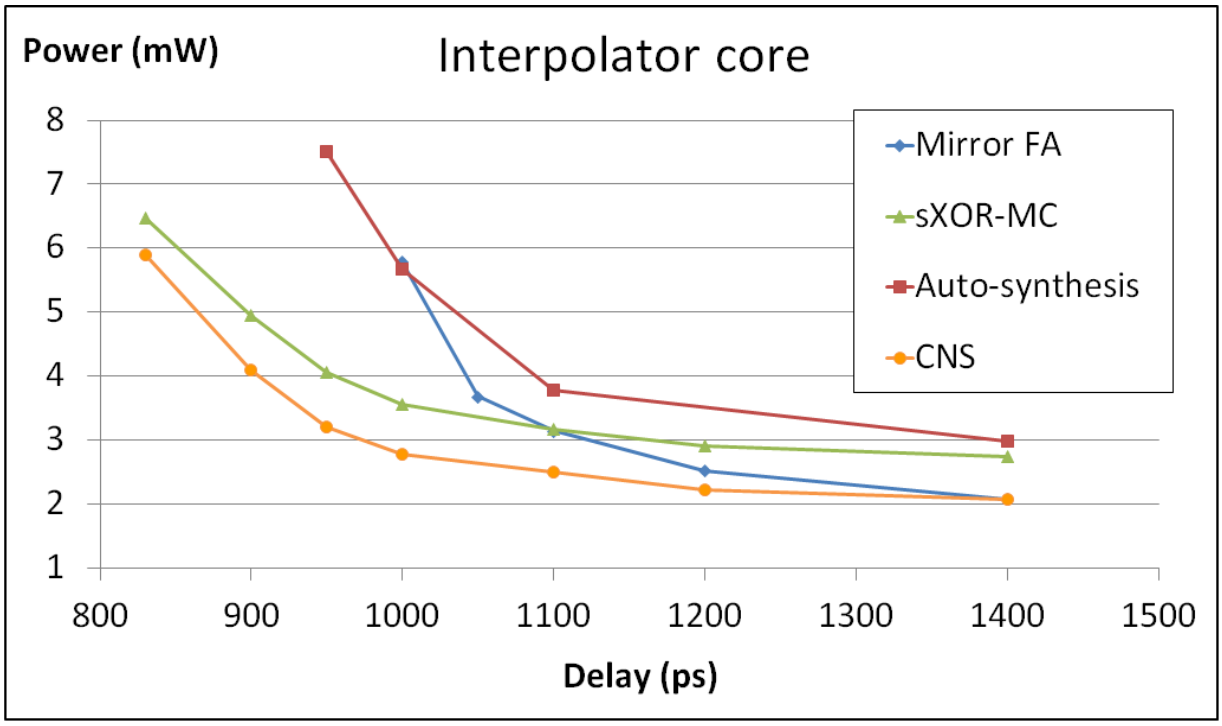


Fig. 24 Power vs. delay plot for a H.264 interpolator core design

D. Results Summary

The average power savings over the common delay range in Figs. 16, 19 and 24 for each of the three designs is shown in Table 2. The power for the CNS designs (the last row) was compared to the use of only Mirror FAs, and versus using the leading commercial synthesis tool. Note that the power is reduced by an average of 18% compared to using only Mirror FAs and by an average of 41% compared to the leading commercial synthesis tool, for exactly the same worst-case delays. Note that the Mirror FA result is similar to [13] which only used the Mirror FA to build the entire compression network in a timing driven fashion; the main improvement is that the CNS algorithm optimizes the use of multiple drop locations (D in Eq. 1) and selects the FA structure according to the timing information.

TABLE 2 AVERAGE POWER REDUCTION RELATIVE TO THE CNS ALGORITHM

	16-vector 16b CSA	**Second order IIR**	**Interpolator core**
Technology	130nm	40nm	40nm
Voltage	1.2V	0.9V	0.9V
Mirror FA [13]	111%	112%	132%
Auto-synthesis	130%	137%	155%
CNS	100%	100%	100%

VI. CONCLUSIONS

A new compression network synthesis (CNS) algorithm, an improved methodology to compress partial products or vectors for power efficient implementations of high performance DSP blocks, was presented for arbitrary arrays of partial products and vectors. Since the full-adder cell is a cornerstone for such compression network, it was determined that the most power efficient full-adder cell for moderate to high speeds was the XOR-based sum function and mirror carry for carry function. Meanwhile, for long delays, the mirror carry-and-sum was more power efficient. A new CSA hardware usage calculation algorithm for any irregularly shaped compression matrix (CM) and a new delay-based wiring and adder selecting algorithm for the CSA tree that used the faster XOR-based FA when one input was an unit gate delay slower in arriving and otherwise used the more area efficient mirror-based FA were developed. The new CNS algorithm was substantially more power efficient than the use of only XOR-based FAs or only mirror-based FAs. Using the best scripts for the leading commercial synthesis tool, CNS produces DSP block implementations that are both faster and consume less power.

REFERENCES

[1] R. Zimmermann and D. Tran, "Optimized synthesis of Sum-of-Products," Asilomar Conf. Sig., Sys. and Comp., Nov. 9-12, 2003.

[2] R. Zimmermann, "Datapath Synthesis for Standard-Cell Design," *19th IEEE international Symposium on Computer Arithmetic*, 2009.

[3] A. Verma, P. Brisk and P. Ienne, "Data-Flow Transformations to Maximize the Use of Carry-Save Representation in Arithmetic Circuits," IEEE Trans. on CAD, vol. 27, iss. 10, Oct. 2008.

[4] O. Kwon, K. Nowka, and Swartzlander Jr., "A 16-Bit by 16-Bit MAC Design Using Fast 5: 3 Compressor Cells," *Journal of VLSI Signal Processing*, vol. 31, pp. 77-89, 2002.

[5] C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Trans. Electron Comp., vol. 13, pp. 14-17, 1964.

[6] L. Dadda, "Some schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, 1965.

[7] A. Weinberger, "A 4:2 carry-save adder module," *IBM Tech. Disclosure Bull*., vol. 23, Jan. 1981.

[8] V. G. Oklobdzijia, D. Villeger and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," IEEE Transactions on Computers, vol. 45, pp. 294-306, 1996.

[9] R. Chadha and J. Bhasker, *Static Timing Analysis for Nanometer Designs*, Springer, 2009.

[10] D. Baran, M. Aktan, and V. G. Oklobdzija, "Energy Efficient Implementation of Parallel CMOS Multipliers with Improved Compressors," *Proc. ISLPED'10*, Aug. 2010.

[11] S. K. Hsu, S. K. Mathew, M. A. Anders, B. R. Zeydel, V. G. Oklobdzija, R. K. Krishnamurthy and S.Y. Borkar, "A 110 GOPS/W 16-bit multiplier and reconfigurable PLA loop in 90-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 41, iss. 1, pp. 256-264, 2006.

[12] D. Radhakrishnan, "Low-voltage low-power CMOS full adder," *IEEE Proc. Circuits Devices and system*, vol. 148, iss. 1, pp. 19-24, Feb 2001.

[13] C. Chang, J. Gu, and M. Zhang, "A review of 0.18-/spl mu/m full adder performances for tree structured arithmetic circuits," IEEE Transactions on VLSI, vol. 13, iss. 6, pp. 686-695, Jun. 2005.

[14] S. Goel et al., "Design of Robust, Energy-Efficient Full Adders for Deep-Submicrometer Design Using Hybrid-CMOS Logic Style," IEEE Trans. on VLS*I*, vol. 14, iss. 12, Dec. 2006.

[15] W. Yeh and C. Jen, "High-speed Booth encoded parallel multiplier design," IEEE Trans. on Computers, vol. 49, iss. 7, pp. 692-701, Jul. 2000.

[16] S. Mahant-Shetti, P. Balsara, and C. Lemonds, "High Performance Low Power Array Multiplier Using Temporal Tiling," IEEE Trans. on VLSI, vol. 7, iss. 1, Mar. 1999.

[17] H. C. Lai and S. Muroga, "Minimum Parallel Binary Adders with NOR (NAND) Gates," IEEE Trans. on Computers, vol. 28, pp. 648-659, Sept. 1979.

[18] H. C. Lai and S. Muroga, "Logic Networks of Carry-Save Adders," IEEE Trans. on Computers, vol. C-31, pp. 870-882, Sept. 1982.

[19] A.V. Oppenheim, R.W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd Edition. Prentice-Hall, Upper Saddle River, NJ, 1999.

[20] JVT of ISO/IEC & ITU-T, Draft ITU-T Recommendation H.264 and Draft ISO/IEC 14496-10 AVC, Doc JVT-Go50. Pattaya, Thailand, 2003.

[21] O. Werner, "Drift analysis and drift reduction for multiresolution hybrid video coding," *Sig. Proc.: Image Commun*., Jul. 1996, vol. 8, p. 387.