A Flexible DSP Block to Enhance FPGA Arithmetic Performance

Hadi Parandeh-Afshar[1], Alessandro Cevrero[2], Panagiotis Athanasopoulos[1], Philip Brisk[3], Yusuf Leblebici[2], and Paolo Ienne[1]

[1]*School of Computer and Communication Sciences*
[2]*School of Engineering*
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015, Lausanne, Switzerland
First_name.Last_name@epfl.ch

[3]*Department of Computer Science and Engineering*
Bourns College of Engineering
University of California, Riverside
Riverside, CA, USA 92521
First_name@cs.ucr.edu

Abstract—**We propose a new DSP block for use in modern high-performance FPGAs. Current DSP blocks contain fixed-bitwidth multipliers that can be combined efficiently to form larger multipliers. Our approach is similar, but includes a bypass layer following the partial product generator that exposes the compressor tree used for partial product reduction directly to the user. As a consequence, the proposed DSP block can accelerate multi-input addition operations in addition to multiplication. To increase the flexibility of the device, the partial product reduction tree used within our DSP block uses a fixed-function compression logic along with a field programmable compressor tree (FPCT), the latter of which is user-configurable to meet the needs of the application at hand. Multi-input addition operations can be mapped directly onto the FPCT without compromising any of the other functionality of the DSP block.**

I. Introduction

This paper proposes a new DSP block designed for use in modern high-performance FPGAs. Existing DSP blocks provide area-efficient implementations of multiplication and multiply-accumulate operations; our proposal extends these capabilities with the ability to perform multi-input addition efficiently as well, enlarging the space of computations that can exploit DSP blocks. Due to its increased flexibility, the DSP block introduced here intends to replace the DSP blocks used in existing high-performance FPGAs.

A. Motivating Examples

One computation that could benefit from our proposed DSP block is the sum-of-absolute differences (SAD) used in video coding: $|A_1 - A_2| + |B_1 - B_2| + |C_1 - C_2| + |D_1 - D_2|$. The fastest implementation of SAD does not use a tree of adders and subtractors, but instead uses a compressor tree implemented using carry-save arithmetic [7, 23].

More generally, Verma et al. [22] introduced a set of systematic data flow transformations that reorganize operations in a datapath to maximize the use of compressor trees. These transformations fuse disparate addition operations together and with the partial product reduction trees of parallel multipliers. Unfortunately, a multiplier that has been fused can no longer use current DSP blocks; however, the compressor tree in the DSP block proposed in this paper is directly accessible and configurable, and is capable of accelerating fused multiply-addition operations.

Fig. 1 shows an example: the *fmult* function from the *g721* benchmark. Fig. 1(a) shows a datapath derived from a C-language implementation. Fig. 1(b) shows the datapath after applying the transformations. The transformed datapath contains a fused multiply-add operation, and two fused addition-subtraction operations. The multiply-add operation cannot map onto existing DSP blocks, because there is no mechanism by which the third input (to the adder) can bypass the *partial product generator (PPG)* of the multiplier in the DSP block. The addition-subtraction operations can also be realized as compressor trees, but existing DSP cannot perform multi-operand addition in general, once again, because there is no way to bypass the PPG of the multiplier.

Fig. 1 motivates the need for a new type of DSP block that can still perform multiplication and MAC, but contains compressor trees that are exposed to the user. The transformations that produced the circuit shown in Fig. 1(b) provide a generalized mapping methodology for applications described in high-level languages, where the programmer is unlikely to understand the finer points of computer arithmetic—such as the benefits of compressor trees over adder trees. The compressor trees could be synthesized on FPGA soft logic instead, as is done today [18-20].

B. Related Work

Modern FPGAs are built from lookup tables (LUTs). The LUTs are connected by a fully custom programmable routing fabric that consumes more than half of the on-chip silicon area. Routing delays may account for more than 50% of the critical path delay of circuits synthesized on an FPGA.

LUTs have been augmented with carry chains. The primary benefit of the carry chain is that operations that utilize it bypass the programmable routing network [5, 9, 10, 14, 16].

978-1-4244-4377-2/09/$25.00 © 2009 IEEE

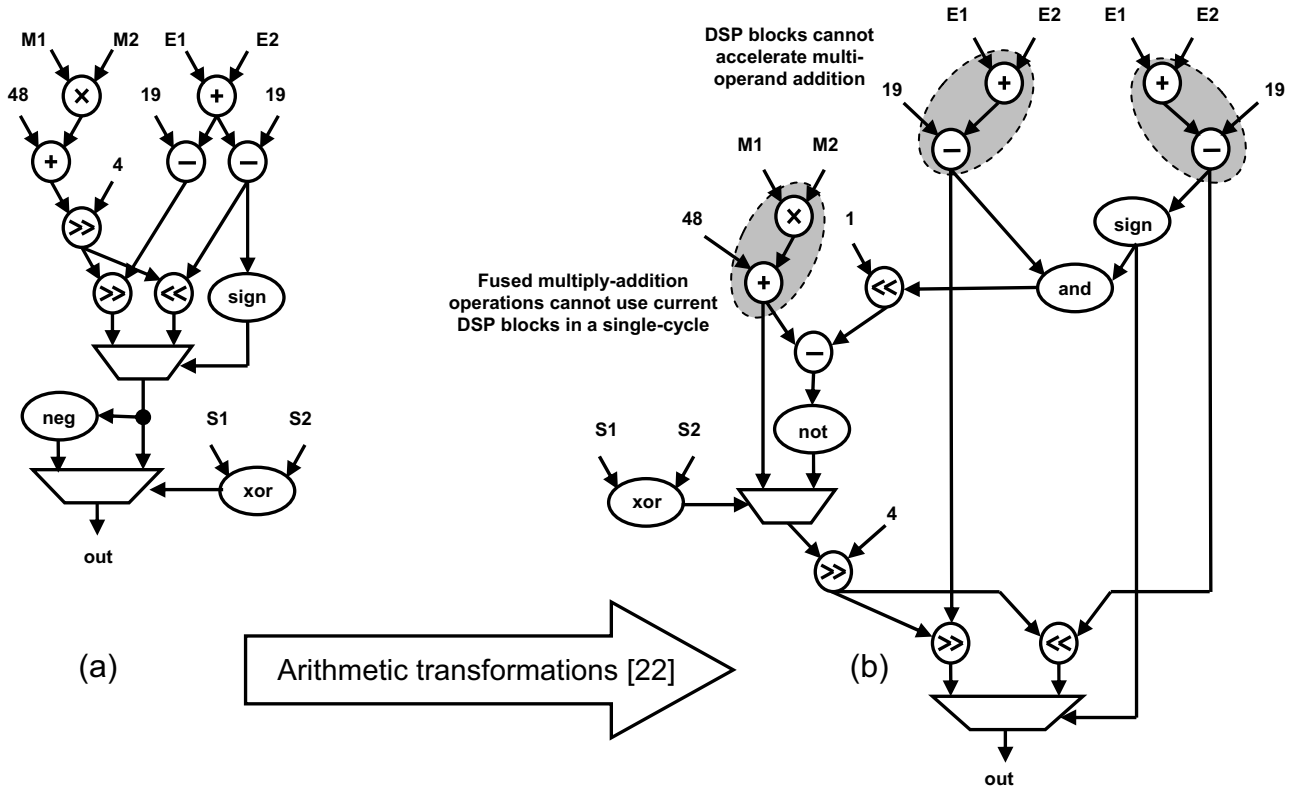

Fig. 1. Motivating example: a data flow graph derived from a language implementation of the *fmult* function in the *g721* benchmark shown before (a) and after (b) applying a set of systematic transformations to form compressor trees [22]. The fused multiply-add and addition-subtraction operations are best realized in compressor trees in ASIC technology, but cannot utilize the DSP blocks in modern high-performance FPGAs, motivating the need for the flexible alternative DSP block introduced in this paper.

Fixed-function ASIC intellectual property cores, such as the aforementioned DSP block [24], have been integrated into FPGAs as well. Our work seeks to enhance the functionality of these blocks so that they can perform multi-operand addition as efficiently as they currently perform multiplication and MAC. Other recent academic proposals suggest the integration of floating-point units, shifters, multiplexors, and crossbars [2, 6, 11-13].

Fig. 2 depicts the half-DSP block used in *Altera*'s *Stratix III* and *IV*-series FPGAs. The half-DSP block contains four *9x9*-bit multipliers, which can be combined with a small adder tree to form an *18x18*-bit multiplier. One limitation of this design is that the compressor trees within the multipliers are inaccessible for multi-input addition in its most general form, i.e., there is no mechanism to bypass the PPG.

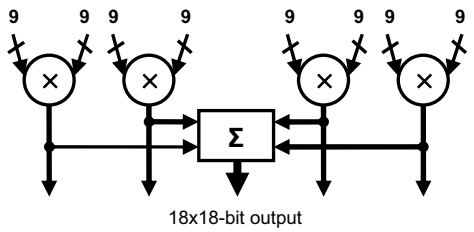

Fig. 2. Conceptual illustration of the four 9x9-bit multipliers in an *Altera Stratix III-IV* half-DSP block, along with a small adder tree that lets them combine to form an 18x18-bit multiplier.

Wallace [23] and Dadda [7] introduced compressor trees in the 1960s; however, their methods built them as networks of full adders, which map poorly onto FPGAs. Poldre and Tammemae [21] described a mapping algorithm to synthesize compressor trees on *Xilinx Virtex* FPGAs; however, their method does not generalize to more recent architectures. Parandeh-Afshar et al. [18, 19] demonstrated a more general technique to synthesize compressor trees on to FPGAs with 6-input LUTs, and later specialized it to exploit the carry chains and arithmetic modes present in the *Stratix II-IV* and *Virtex 5-6* family FPGAs [20]. Providing dedicated hardware support for compressor trees, in contrast, permits a much more performance and area-efficient realization of compressor trees compared to realize them using LUTs and routing resources.

Cevrero et al. [3, 4, 17] introduced the *field programmable compressor tree (FPCT)* as a flexible alternative to DP blocks. The FPCT is user-configurable and can adapt to many multi-input addition operations, including partial product reduction; however, the PPG must be synthesized on LUTs, which is wasteful of LUT resources and stresses the routing network. An $m \times n$-bit multiplier generates $O(mn)$ partial product bits (2-input AND gates), which are synthesized on LUTs and routed to the FPCT inputs; in contrast, just $m + n$ inputs are routed to a dedicated multiplier. Fig. 3 shows an FPCT that has been configured to implement a *9x9*-bit Baugh-Wooley (signed) multiplier [1] with a LUT-based PPG.

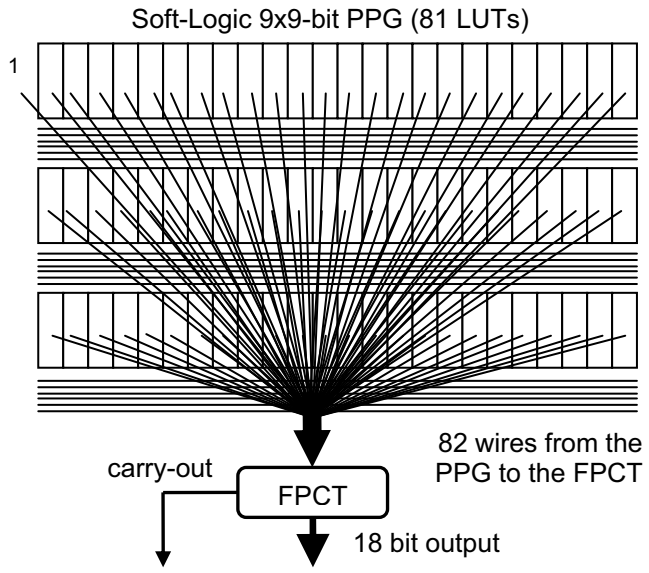

Fig. 3. A 9x9-bit signed multiplier with PPG synthesized on LUTs and using an FPCT for partial product reduction.

C. Contribution and Limitations

The DSP block proposed here integrates a bypassable PPG into an FPCT. The DSP block can be configured to perform multiplication, using the FPCT for partial product reduction, or to bypass the PPG to use the FPCT for multi-operand addition. Integrating the PPG and the FPCT in the same block eliminates the costs arising from a LUT-based PPG, which were illustrated in Fig. 3. To improve logic utilization within the DSP block, some fixed-function compression logic is included along with the FPCT. A *9x9*-bit PPG was chosen to match the multipliers provided by the DSP blocks in *Altera*'s *Stratix III* and *IV* FPGAs. To efficiently build larger-bitwidth multipliers from several of our DSP blocks, one of the PPGs is modified to perform several reduced-bitwidth multiplication operations. More details will be provided in Section IV.D.

Cevrero et al. [4] estimated that the area of a half-DSP block exceeds that of an FPCT; however, they did not quantify the difference in size. The proposed DSP block can implement two *9x9*-bit multiplication operations, while *Altera*'s half-DSP block can realize four. In terms of area, the FPCT-based design includes several sources of overhead: the FPCT is larger than a comparable fixed-function compressor tree, as it is a form of configurable logic; a layer of eighty-two *2:1* multiplexors per *9x9*-bit PPG is required to make the PPG bypassable; the aforementioned modifications to the PPG require some additional multiplexing; lastly, the existence of fixed-function compression logic also adds to the size.

Although it is larger than current DSP blocks, the block proposed here is considerably more flexible, as it can perform multi-input addition in addition to multiplication and MAC. The existence of transformations to automatically expose large compressor trees [22] is fundamental to the utility of the new DSP block; without these transformations, it is unlikely that compressor trees would be found in sufficient quantity to justify the inclusion of dedicated hardware in FPGAs.

II. Arithmetic Preliminaries

Here, we briefly summarize concepts and terminology from computer arithmetic that will be used throughout the paper.

Let $B = b_{k-1}b_{k-2}...b_1b_0$ be a k-bit integer. The subscript r of a bit b_r is called the *rank* of b_r. Each bit of rank r contributes a total value of b_r2^r to the value of B. The value of B is therefore $b_02^0 + b_12^1 + ... + b_{k-2}2^{k-2} + b_{k-1}2^{k-1}$. Given a set of integers to sum, a *column* is a set of bits having the same rank. The input to a multi-input adder, including the partial product reduction tree of a multiplier, can be viewed as a set of columns of varying height, rather than a set of *rows* (i.e., integers).

An *m:n counter* is a circuit that takes m input bits, counts the number of input bits that are set to *1*, and outputs the result as an n-bit unsigned binary integer, whose value ranges from *0* to m; given m the number of output bits required is $n = \lceil log_2(m+1) \rceil$. An *m:n* counter compresses m input bits of rank r down to n input bits of ranks r, $r+1$, …, $r+n-1$.

A *generalized parallel counter (GPC)* is a type of counter that can count bits having different ranks. A GPC is expressed as a tuple $P = (K_{N-2}, K_{N-1}, ..., K_1, K_0; W)$, where K_r is the number of input bits of rank r to sum, and W is the number of output bits. The maximum output value is denoted M, and is computed as $M = K_02^0 + K_12^1 + ... + K_{N-2}2^{N-2} + K_{N-1}2^{N-1}$; to represent a value in the range *0* to M, the GPC produces $W = \lceil log_2(M+1) \rceil$ output bits. As an example, a *(3, 3; 4)* GPC compresses three bits of rank r, and three bits of rank $r+1$ down to four bits of ranks r, $r+1$, $r+2$, and $r+3$.

An *M:W* counter can implement the functionality of a GPC; it suffices to connect each GPC input bit of rank r to 2^r inputs of the *M:W* counter. Continuing the example, a *9:4* counter could be used to implement the *(3, 3; 4)* GPC.

A *carry propagate adder (CPA)* is a circuit that adds two binary integers. A wide variety of different CPAs exist, such as ripple-carry, carry-select, etc.; we use the term "CPA" to discuss addition in an architecture-agnostic fashion.

Multi-input addition refers to the problem of adding $k > 2$ integers (or, sets of columns where at least one column contains more than two bits). Let A_0, A_1, …, A_{n-1} be a set of integers. A *compressor tree* is a circuit that produces outputs, *Sum (S)* and *Carry (C)*, such that $S+C = A_0 + A_1 + ... + A_{n-1}$ using carry-save addition (e.g., counters or GPCs). The output of a compressor tree is a redundant number system which is called the *carry-save representation*. A CPA is then used to compute $S+C$ to produce the final result.

III. FPCT Overview and Extensions

The FPCT [4] is a user-configurable multi-input adder. As shown in Fig. 4, the FPCT is an array of 8 components, called Compressor Slices (CSlices). Each CSlice, shown in Fig. 4(a), takes as input 16 bits to sum along with 15 carry-in bits, and produces up to 6 output bits, depending on its configuration, along with 15 carry-out bits. The 8-CSlice FPCT, shown in Fig. 4(b), takes 128 input bits (16 per CSlice) along with 15 carry-in bits for the lowest order (rightmost) CSlice; it produces up to 48 output bits (6 per CSlice) and the highest order CSlice (leftmost) produces 15 carry-out bits. The FPCT itself is left unmodified in this paper.

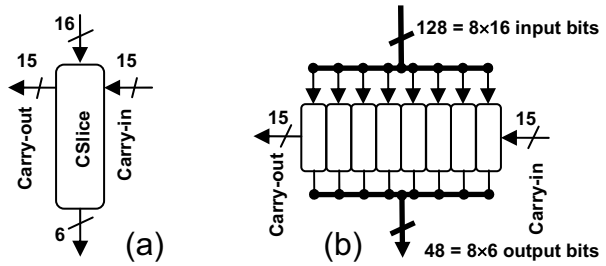

Fig. 4. I/O interface to a CSlice (a) and an 8-CSlice FPCT (b).

A. Architectural Overview

To conserve space, we omit a detailed discussion of the CSlice architecture. Compression is performed by a network of GPCs of varying size. The CSlice includes a bypassable CPA: it can produce 1, 2, or 3 output when using the CPA, or 2, 4, or 6 output bits in carry-save form, bypassing it; the latter is used when building large, multi-FPCT compressor trees.

At the top level, the CSlice contains a configurable *31:5* counter that can implement a variety of GPCs having the form $(K_2, K_1, K_0; 5)$, satisfying two constraints: $K_0 + K_1 + K_2 \leq 16$, and $K_0 2^0 + K_1 2^1 + K_2 2^2 \leq 31$; for example, it can implement a $(5, 4, 3; 5)$ GPC, but not a $(2, 2, 2, 3; 5)$ GPC, because the maximum difference in the rank of the input bits is at most 2.

The 15-carry-out bits connect to subsequent CSlices using a limited carry chain; depending on the configuration of the CSlice, some of these bits may be zero; likewise, the rank of each carry-out bit depends on the configuration.

B. Configuration for 9x9-bit Signed Multiplication

A Baugh-Wooley PPG [1] for an $m \times n$-bit signed multiplier produces $mn + 1$ partial product bits, some of which are computed using a NAND gate rather than an AND gate, and the most significant output bit of the multiplier is inverted. One of the partial product bits is set to the constant value '1.'

Fig. 5 shows that seven CSlices can realize a partial product reduction for a *9x9*-bit Baugh-Wooley PPG using an FPCT. A detailed analysis shows that this approach does not use FPCT resources efficiently. For a given set $S = (K_2, K_1, K_0)$ of input bits of varying rank, the *input utilization* of a CSlice is defined to be the quantity $\rho_{input} = (K_0 + K_1 + K_2)/16$, and the *counter utilization* is the quantity $\rho_{counter} = (K_0 2^0 + K_1 2^1 + K_2 2^2)/31$. These metrics measure our usage of the input bandwidth of the CSlice (16 inputs) and the first-level counter (31 inputs). These metrics easily generalize to measure the input and counter utilization of a complete FPCT as well.

Table I shows the input and counter utilization for the mapping shown in Fig. 5 for each CSlice and for the FPCT. For the FPCT, $\rho_{input} = 64\%$ and $\rho_{counter} = 71\%$. In particular, counters C_0 and C_6 are underutilized because the 3 leftmost and rightmost columns have few partial product bits—this is simply the nature of parallel multipliers; and the last CSlice, C_7 is not used at all. Admittedly, the overall utilization would improve dramatically for an FPCT with *7*, rather than *8*, CSlice; however, Cevrero et al. have established that an *8*-CSlice FPCT is a good choice for integration into an FPGA in terms of bandwidth and area [4].

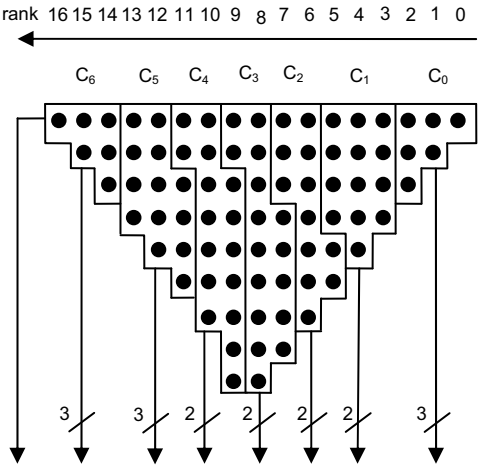

Fig. 5. Seven CSlices are required to compress the *82* partial product bits generated by a *9x9*-bit Baugh-Wooley partial product generator; however, two FPCTs are needed due to the carry-outputs.

TABLE I.
FPCT INPUT AND COUNTER UTILIZATION FOR THE MAPPING (9X9-BIT MULTIPLIER) SHOWN IN FIG. 5.

Resource	ρ_{input}		$\rho_{counter}$	
C_1	38%	6/16	55%	17/31
C_2	81%	13/16	97%	30/31
C_3	75%	12/16	90%	28/31
C_4	100%	16/16	100%	31/31
C_5	100%	16/16	94%	29/31
C_6	81%	13/16	97%	30/31
C_7	38%	6/16	36%	11/31
C_8	0%	0/16	0%	0/31
FPCT	64%	82/128	71%	176/248

C. Extensions to Improve Input and Counter Utilization

To increase ρ_{input} and ρ_{input}, we propose to use hard logic to compress the leftmost and rightmost columns of partial product bits. These two hard logic functions are referred to as *'A'* and *'B'* throughout the remainder of the paper. This scheme, shown in Fig. 6, reduces the number of CSlices required to implement a *9x9*-bit Baugh Wooley multiplier from *7* to *4*. As shown in Table II, ρ_{input} and $\rho_{counter}$ jump from *64%* to *94%* and *71%* to *98%* respectively; only *8* CSlice inputs and *6* counter inputs are unused. By design, the fixed-function compression logic achieves *100%* utilization.

Fig. 7 illustrates hard circuits *'A'* and *'B'* and their connections to a 4-CSlice "½-FPCT." In principle, hard circuits *A* and *B* could be augmented with the option to produce their outputs in carry-save form; this could be useful when using *9x9*-bit multipliers as building blocks for larger multipliers. We have opted not to include this option in order to simplify the design presented here.

IV. DSP BLOCK ARCHITECTURE AND SYNTHESIS

Fig. 8 depicts the proposed DSP block. An 8-CSlice FPCT is decomposed into two 4-CSlice c½-FPCTs, each of which is augmented with a bypassable 9x9-bit Baugh-Wooley PPG and hard circuits *A* and *B*.

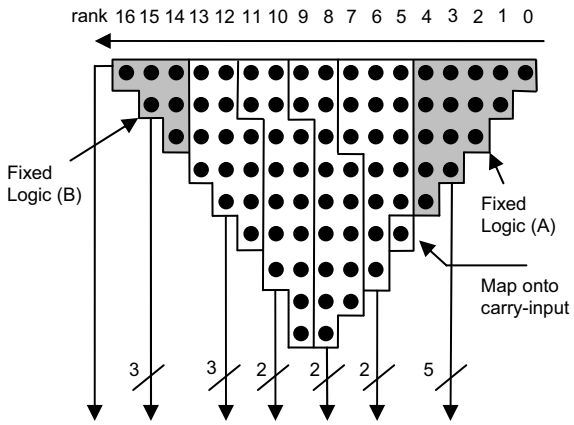

Fig. 6. By using some fixed logic to compress the five lowest and here thighest rank columns of partial product bits, the number of CSlices required is reduced from eight to four, permitting two *9x9*-bit unsigned Baugh-Wooley multipliers to be realized by a single *8*-CSlice FPCT. Note that one bit must connect to one of the carry-in bits of the lowest ranking CSlice in order to facilitate this mapping.

TABLE II.
FPCT INPUT AND COUNTER UTILIZATION FOR THE MAPPING (TWO 9X9-BIT MULTIPLIERS) SHOWN IN FIG. 6.

Resource	ρ_{input}		$\rho_{counter}$	
C_1	94%	15/16	100%	31/31
C_2	100%	16/16	100%	31/31
C_3	100%	16/16	94%	29/31
C_4	81%	13/16	97%	30/31
C_5	94%	15/16	100%	31/31
C_6	100%	16/16	100%	31/31
C_7	100%	16/16	94%	29/31
C_8	81%	13/16	97%	30/31
FPCT	94%	120/128	98%	242/248

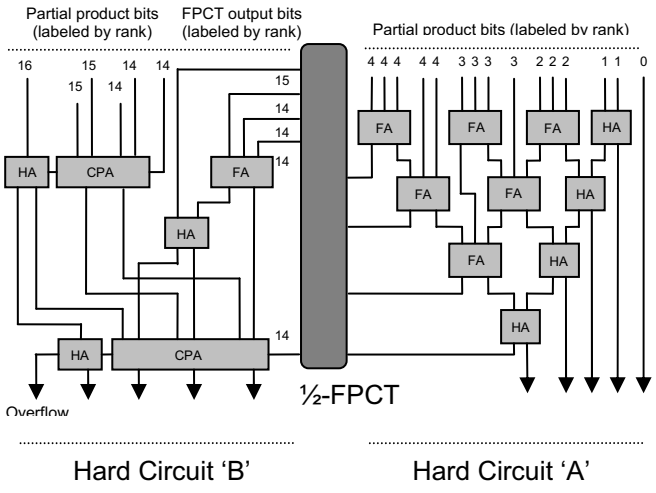

Fig. 7. Hard circuits 'A' and 'B' and their connection to a 4-CSlice FPCT, corresponding to Fig. 6.

The DSP block can be configured to implement two *9x9*-bit signed multipliers, as discussed in the preceding section; alternatively, it can be configured as an 8-CSlice FPCT (note that the FPCT itself has many different configurations itself in the latter mode, as discussed previously [3, 4]).

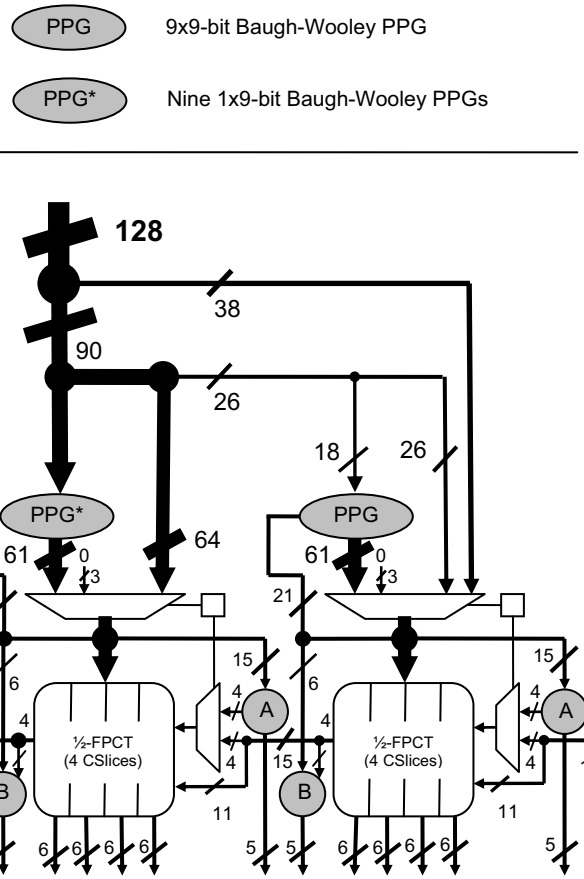

Fig. 8. The proposed DSP block, which contains two bypassable PPGs (one a 9x9-bit Baugh-Wooley PPG, the other contains nine 1x9-bit PPGs), , and a ½-FPCT which performs partial product reduction in conjunction with two hard logic compression circuits 'A' and 'B.' The FPCT can be used for multi-operand addition [3, 4] by bypassing the PPGs.

In actuality, the two PPGs on the left- and right-hand side of the DSP block in Fig. 8 are not identical. The PPG on the right-hand side is a straightforward *9x9*-bit Baugh-Wooley PPG. The PPG on the left-hand side, however, is more complex, as it has been designed to implement up to nine *1x9*-bit PPGs; using the maximum number of PPGs in this case is the same as a *9x9*-bit Baugh-Wooley PPG. The justification for this implementation decision is presented in Subsection B.

A. Multiplier Synthesis

The proposed DSP block has been designed to implement *9x9*-bit signed multiplication efficiently; here, we discuss how to synthesize larger signed multipliers from a combination of the *9x9*-bit signed multipliers in our DSP block.

We discuss our method using an example. Here, we perform *12x12*-bit multiplication. We denote the two inputs as *A[11...0]* and *B[11…0]* respectively. The most straightforward approach would be to perform four separate multiplications: *A[8..0]×B[8..0]*, *A[8..0]×B[11..9]*, *A[11..9]×B[8..0]*, and *A[11..9]×B[11..9]*, followed by a compressor tree to sum the results. This would require a two of our proposed DSP blocks to perform the four multiplications, and a third to sum the result (using just the FPCT).

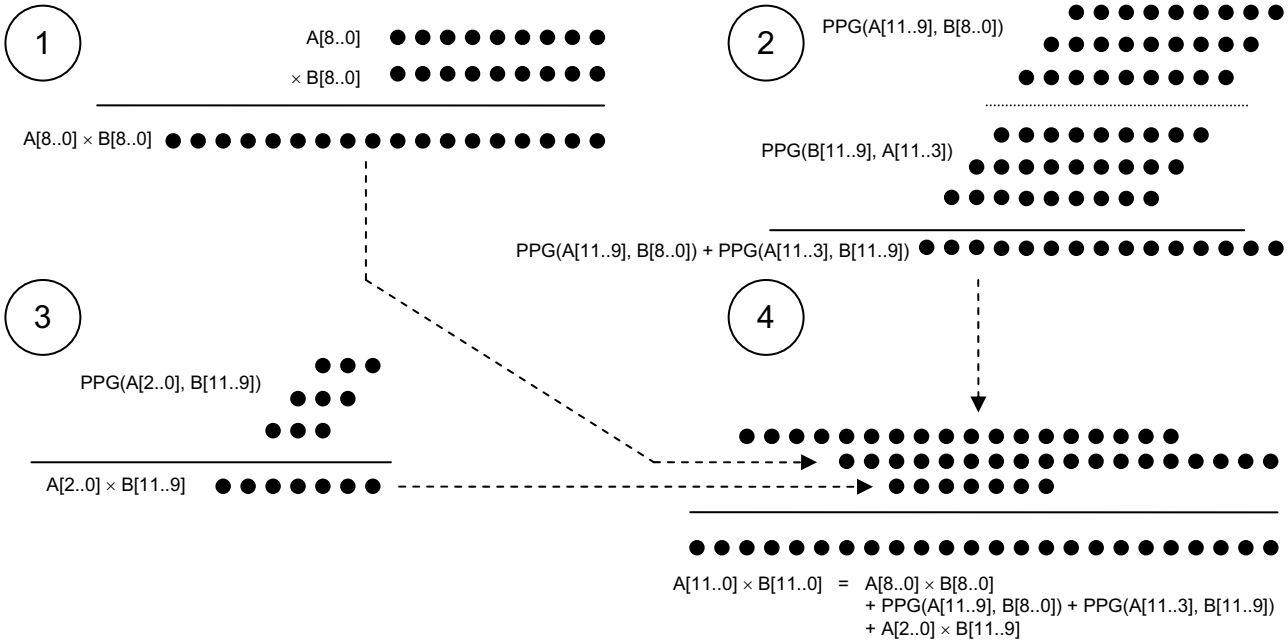

Fig. 9. Illustration of the decomposition of a *12x12*-bit multiplier into four smaller operations using several of the proposed DSP blocks. There are three multiplication steps (step 2 being irregular, in this case, requiring the modified PPG) and one three-input addition (step 4). Arguably, Steps 3 and 4 underutilize FPCT input and counter resources, and could be synthesized on FPGA soft logic instead. As a 3-input addition operation, Step 4 could be achieved using one layer of FPGA logic by exploiting carry chains; 3-input addition is supported in the carry chains of high-end FPGA families from both *Xilinx* and *Altera*.

Alternatively, we can use a different decomposition, shown in Fig. 9. As in the previous case, we compute *A[8..0]×B[8..0]*. A second multiplication is formed by generating two partial products, *PPG(A[11..9], B[8..0])* and *PPG(B[11..9], A[11..3])* and adding them; taken together, these partial products form the same bit pattern as one would get for a *6x9*-bit multiplication. Lastly, we multiply *A[2..0]×B[11..9]*. The results are then added together. This requires two of our modified DSP blocks to perform three multiplication operations and one multi-input addition. Alternatively, we can compute the *3x3*-bit *A[2..0]×B[11..9]* multiplication using FPGA soft logic, and use just one FPCT to sum the results; this reduces our resource usage to *1.5* of our proposed DSP blocks; the multiplier/FPCT of the second DSP block is not used, and is available to perform other computations.

B. PPG Input Interface Design

A normal *9x9*-bit multiplier structure cannot compute the sum of *PPG(A[11..9], B[8..0])* and *PPG(B[11..9], A[11..3])*, despite the fact that only *54* partial product bits are generated. We can view each row of bits in Fig. 9 as a vector. To compute a normal multiplication, the same vector is input to each row of the PPG; in Fig. 9, two vectors are required.

In principle, a situation could arise where we want to compute the sum of *9* independent *1x9*-bit PPGs. This would require a total of *81 + 9 = 90* input bits, *81* independent bits in each vector and *9* independent bits with which to multiply each respective vector.

With two *9x9*-bit PPGs, this would require the DSP block to have *180* inputs; however, the FPCT has *128* inputs, as shown in Fig. 2 (plus *15* carry-inputs). As discussed in a prior paper, the FPCT has comparable I/O bandwidth to the half-DSP blocks used in *Altera*'s *Stratix III* and *IV* FPGAs [4]. The I/O bandwidth of the half-DSP blocks has been designed precisely to reduce stress on the routing network [15]. Increasing the number of inputs to the proposed DSP block, therefore, would be less than ideal.

Our solution is to increase the flexibility of one of the two *9x9*-bit PPGs per DSP block in this fashion. The normal *9x9*-bit PPG requires *18* inputs (Fig. 8, right-hand side), while the modified one requires *90* (Fig. 8, left-hand side). The total number of inputs, *108*, is less than the FPCT's *128* inputs, and therefore does not increase the number of DSP block inputs beyond what was already required to facilitate the FPCT.

V. Experimental Methodology and Results

The proposed DSP block, including the FPCT, was modeled in structural VHDL. Correctness was verified using *Mentor Graphics Modelsim*; synthesis was performed with *Synopsys Design Compiler* and *Design Vision*; placement and routing was performed by *Cadence Design Systems' Silicon Encounter* tool. A *90nm Artisan* standard cell library based on a *TSMC* design kit was used. Floorplanning was performed with the goal of minimizing the routing delay. Based on our design flow, the proposed DSP block was *8%* larger than an 8-CSlice FPCT alone. *Altera's Stratix II* FPGA is also a 90nm device, so we evaluate the benefits of our proposed DSP block in the context of this FPGA.

Ho et al. [11] introduced the *Virtual Embedded Block (VEB)* methodology, which facilitates the use of vendor CAD tools to evaluate *intellectual property (IP) cores* under consideration for integration into FPGAs. A benchmark circuit is placed-and-routed on an FPGA; the subcircuit to be mapped onto the proposed IP core is synthesized on soft logic, i.e., it cannot use DSP blocks or blockRAMs. An ASIC design flow is used to determine the delay of the subcircuit that uses the proposed IP core. By hand, the designer must back-substitute the delays of the subcircuit using the proposed IP core its soft logic delays, and then compute the overall critical path delay of the circuit. Thus, the VEB methodology permits the evaluation of the IP core in the context of a circuit that has been placed and routed using vendor-supplied tools. Several other papers [3, 4, 6] have adopted this methodology to evaluate IP blocks, including the FPCT [3, 4].

Prior work has established the ability of the FPCT to accelerate multi-input addition operations [3, 4]; although a few layers of multiplexors have been added to the FPCT, we do not believe that they will significantly affect the benefits previously observed for multi-input addition operations. The primary drawback, from the perspective of area, was that all multiplication operations that were synthesized on the FPCT required a soft logic PPG, which significantly increased area utilization compared to existing DSP blocks. Our primary goal is to show that the DSP block proposed here can regain this area utilization by integrating a hard, but bypassable, PPG.

We consider multipliers of five bitwidths: *m9x9*, *m10x10*, *m12x12*, *m18x18*, and *m20x20*. We compare our proposed DSP block with FPGA (soft logic), DSP block, and FPCT implementation of the multipliers. When using the FPCT, each PPG is synthesized on soft logic; the partial product reduction tree and CPA are mapped onto one or more FPCTs.

Each *Stratix II* DSP block contains four *18x18*-bit multipliers; each can be configured to implement two *9x9*-bit multipliers, and all four can be combined to form a *36x36*-bit multiplier. The delay of each *9x9*-bit multiplier is close to the delay of the *18x18*-bit multiplier, and is significantly greater than the delay of a standalone *9x9*-bit multiplier.

m9x9 is mapped onto one of the eight *9x9*-bit multipliers in the *Stratix II* DSP block; *m10x10*, *m12x12*, and *m18x18* use *18x18*-bit multipliers, and *m20x20* uses a *36x36*-bit multiplier.

For our proposed DSP block, *m9x9* uses one of two *9x9*-bit multipliers; *m18x18* uses four *9x9*-bit multipliers plus another FPCT to sum the four results. *m10x10*, *m12x12*, and *m20x20* are decomposed as described in Section IV-C. For the purpose of comparison, we also synthesized each multiplier wholly on soft logic, using ternary (3-input) adder trees and GPC mapping [18] to form the partial product reduction trees.

Fig. 10 shows the critical path delay of each multiplier with each synthesis method. We did not expect the proposed DSP block to perform better than the *Stratix II* DSP block, because we are effectively replacing a mostly fixed multiplier structure with a reconfigurable one. In the case of *9x9*-bit multiplication, our proposed DSP block was faster because of the inefficient implementation of a *9x9*-bit multiplier in the Stratix II DSP block, as discussed above.

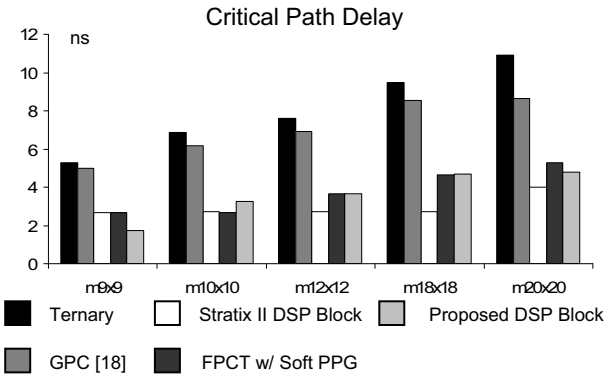

Fig. 10. Delay of the different multipliers.

For the larger bitwidth multiplication operations, the Stratix II DSP block retains an advantage because there are direct connections between adjacent blocks, so multipliers as large as *36x36*-bit multiplication can be formed without using the FPGA routing network; in contrast, in accordance with the VEB methodology, we connect several of our proposed DSP blocks and FPCTs together using the routing network. In principle, an FPGA that uses our proposed DSP block should provide fast connection to adjacent blocks, similar in principle to the way that current DSP blocks are integrated into FPGAs.

Comparable delays were observed for the proposed DSP block and the FPCT with soft-logic PPG for all benchmarks. For *m10x10* and *m18x18*, the FPCT achieved reduced the critical path delays compared to our proposed DSP block. The reason is that our proposed DSP block's usage of an integrated PPG forces the decomposition of multiplication into *9x9*-bit fragments, whereas, using a soft PPG allows one layer of FPCTs to perform partial product reduction directly.

The critical path delay of a multiplier alone is unlikely to be the critical path of a large industrial benchmark. Additional logic may be chained with the multiplier, or the critical path may occur in a different part of the circuit.

Table III shows the number of IP blocks that are required to build each multiplier. For the *Stratix-II* DSP block, it reports the number of *9x9* multipliers. For *FPCT* and the proposed DSP block, it reports the number of half-FPCT/DSP blocks. Table III also shows the number of ALMs used when synthesizing multipliers on the FPCT (soft PPG) and our proposed DSP block (low-bitwidth multipliers, e.g. *2x2*-bit or *3x3*-bit). The proposed DSP block requires considerably fewer resources than using the FPCT alone; moreover, it retains the ability of the FPCT to accelerate general multi-input addition operations, which the *Stratix II* DSP block cannot [3, 4].

To compare the total respective area of the data reported in Table III, we scaled the total area to that of an ALM. Cevrero et al. [4] reported that the area of each FPCT block is equal to the area of *8* ALMs; our estimates show that the area of each *Stratix II* DSP block is *9.6x* larger than the area of an FPCT, and that the area of our proposed DSP block is *8%* larger than that of an FPCT. Fig. 11 compares the overall area required for each benchmark, normalized to the area of the number of half-DSP blocks used in the *Stratix II*.

TABLE III.
NUMBER OF HARD LOGIC BLOCKS AND ALMS USED FOR EACH MULTIPLIER SYNTHESIS METHOD

	Logic Block Usage					ALM Usage				
Hard Logic Block	m9x9	m10x10	m12x12	m18x18	m20x20	m9x9	m10x10	m12x12	m18x18	m20x20
Stratix II DSP Block	1	2	2	2	8	0	0	0	0	0
FPCT w/Soft PPG	2	2	4	6	14	47	50	72	165	200
Proposed DSP Block	1	4	4	7	10	0	1	4	0	2

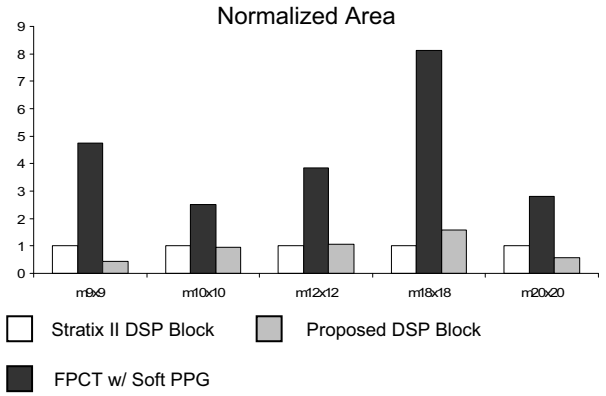

Fig. 11. Normalized area of the different multipliers.

The FPCT was not competitive in terms of area, due to the use soft logic PPG. For *m9x9* and *m20x20*, the proposed DSP block achieved better area utilization than the *Stratix II* DSP block, which used *18x18* and *36x36*-bit multipliers to realize these functions. The *Stratix II* DSP block, in contrast, achieved better area utilization for *m18x18*, as it contained a well-design *18x18*-bit ASIC multiplier. The area utilization was comparable for *m10x10* and *m12x12*.

VI. CONCLUSION

We have proposed a new DSP block for FPGAs that can perform multi-input addition as well as multiplication. Our proposal combines two bypassable *9x9*-bit PPGs with two 4-CSlice half-FPCTs and some fixed function partial product compression logic. Our experiments show that our DSP block remains competitive with *Altera*'s *Stratix II* DSP block in terms of critical path delay and area utilization; while retaining the FPCT's ability to accelerate multi-input addition operations, unlike existing DSP blocks. We strongly believe that the additional flexibility offered by our proposed DSP block is a significant advantage compared to the DSP blocks employed in existing FPGAs.

REFERENCES

[1] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Computers*, vol. C-22, 1973 pp. 1045-1047.

[2] M. J. Beauchamp, S. Hauck, K. D. Underwood, and K. S. Hemmert, "Architectural modifications to enhance the floating-point performance of FPGAs," *IEEE Trans. VLSI*, vol. 16, no. 2, pp. 177-187, Feb. 2002.

[3] A. Cevrero, et al., "Architectural improvements for field programmable counter arrays: enabling efficient synthesis of fast compressor trees on FPGAs," in *Proc. Int. Symp. FPGAs '08*, 2008, pp. 181-190.

[4] A. Cevrero, et al., "Field programmable compressor trees: acceleration of multi-input addition of FPGAs," *ACM Trans. Reconfigurable Technology and Systems*, vol. 2, no. 2, article no. 13, June, 2009.

[5] D. Cherepacha and D. Lewis, "DP-FPGA: an FPGA architecture optimized for datapaths," in *Proc. VLSI Design '96*, 1996, pp. 323-343.

[6] Y. J. Chong and S. Parameswaran, "Flexible multi-mode embedded floating-point unit for field programmable gate arrays," in *Proc. Int. Symp. FPGAs '09*, 2009, pp. 171-180.

[7] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, Mar. 1965, pp. 349-356.

[8] W. Feng and S. Kaptanoglu, "Designing efficient input interconnect blocks using counting and entropy," *ACM Trans. Reconfigurable Technology and Systems*, vol. 1, no. 1, article no. 6, Mar. 2008.

[9] M. T. Frederick and A. K. Somani, "Multi-bit carry chains for high performance reconfigurable fabrics," in *Proc. Int. Conf. on Field Programmable Logic and Applications '06*, 2006, pp. 1-6.

[10] S. Hauck, M. M. Hosler, and T. W. Fry, "High-performance carry chains for FPGAs," *IEEE Trans. VLSI*, vol. 8, no. 2, pp. 138-147, Apr., 2000.

[11] C. H. Ho, et al., "Virtual embedded blocks: a methodology for evaluating embedded elements in FPGAs," in *Proc. IEEE Symp. Field Programmable Custom Computing Machines '06*, 2006, pp. 35-44.

[12] P. Jamieson and J. Rose, "Architecting hard crossbars on FPGAs and increasing their area-efficiency with shadow clusters," in *IEEE Int. Conf. Field Programmable Technology '07*, 2007, pp. 57-64.

[13] P. Jamieson and J. Rose, "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters," in *IEEE Int. Conf. Field Programmable Technology '06*, 2006, pp. 1-8.

[14] A. Kaviani, D. Vranisec, and S. Brown, "Computational field programmable architecture," in *Proc. IEEE Custom Integrated Circuits Conf. '98*, 1998, pp. 261-264.

[15] M. Langhammer. Private communication. 2009.

[16] K. Leijten-Nowak and J. L. van Meerbergen, "An FPGA architecture with enhanced datapath functionality," in *Proc. Int. Symp. FPGAs '03*, 2003, pp. 195-204.

[17] H. S. Niaki, et al., "Design space exploration for field programmable compressor trees," in *Proc. Int. Conf. Compilers, Architecture, and Synthesis for Embedded Systems '08*, 2008, pp. 207-216.

[18] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in *Proc. Asia South Pacific Design Automation Conf. '08*, 2008, pp. 138-143.

[19] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving synthesis of compressor trees on FPGAs via integer linear programming," in *Proc. Design Automation and Test in Europe '08*, 2008, pp. 1256-1261.

[20] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *Proc. 19th Int. Conf. Field Programmable Logic and Applications '09*, 2009, pp. 242-249.

[21] J. Poldre and K. Tammemae, "Reconfigurable multiplier for Virtex FPGA family," in *Proc. Int. Wkshp. Field Programmable Logic and Applications '99*, 1999, pp. 359-364.

[22] A. K. Verma, P. Brisk, and P. Ienne, "Data-flow transformations to maximize the use of carry-save representation in arithmetic circuits," *IEEE Trans. CAD*, vol. 27, no. 10, Oct. 2008, pp. 1761-1774.

[23] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electronic Computers*, vol. 13, Feb. 1964, 14-17.

[24] P. S. Zuchowski, C. B. Reynolds, R. J. Grupp., S. G. Davis, B. Cremen, and B. Troxel, "A hybrid ASIC and FPGA architecture," in *Proc. Int. Conf. Computer-Aided Design '02*, 2002, pp. 187-194.