EXPLOITING FAST CARRY-CHAINS OF FPGAS FOR DESIGNING COMPRESSOR TREES

Hadi Parandeh-Afshar, Philip Brisk and Paolo Ienne

School of Computer and Communication Sciences
Ecole Polytechnique Fédéral de Lausanne
CH-1015 Lausanne, Switzerland
email: {first_name.last_name}@epfl.ch

ABSTRACT

Fast carry chains featuring dedicated adder circuitry is a distinctive feature of modern FPGAs. The carry chains bypass the general routing network and are embedded in the logic blocks of FPGAs for fast addition. Conventional intuition is that such carry chains can be used only for implementing carry-propagate addition; state-of-the-art FPGA synthesizers can only exploit the carry chains for these specific circuits. This paper demonstrates that the carry chains can be used to build compressor trees, i.e., multi-input addition circuits used for parallel accumulation and partial product reduction for parallel multipliers implemented in FPGA logic. The key to our technique is to program the lookup tables (LUTs) in the logic blocks to stop the propagation of carry bits along the carry chain at appropriate points. This approach improves the area of compressor trees significantly compared to previous methods that synthesized compressor trees solely on LUTs, without compromising the performance gain over trees built from ternary carry-propagate adders.

1. INTRODUCTION

FPGAs are widely used to implement signal processing and multimedia applications, whose performance is dictated by the efficiency of the implementation of their arithmetic kernels. Dedicated arithmetic circuitry, such as multipliers DSP blocks, and carry chains, has been embedded in modern FPGAs to improve performance and logic density for these industrially relevant circuits. The carry chains connect adjacent logic cells, bypassing the programmable routing network, which significantly reduces delay. The carry chains have been augmented with additional circuitry to facilitate fast carry-propagate addition. *Altera*' carry chains include *full adders (FAs)* which form a *ripple-carry adder* [1]; *Xilinx* added *xor* gates and programmable multiplexors to their carry chains to facilitate efficient parallel-prefix addition. The DSP blocks are distinct from the logic cells, and offer efficient multiplication and multiply-accumulate for a variety of bitwidths.

Although embedding dedicated arithmetic circuitry in FPGAs improves the implementation of certain circuits, there is a concern about the range of applications that can use these resources. Therefore, it is important to explore new techniques to utilize these resources as generally as possible. This can be accomplished either through the introduction of more generally useful and programmable dedicated blocks, or sophisticated new mapping algorithms; this paper explores the latter.

Conventional intuition has held that carry chains can only implement *carry-propagate adders (CPAs)* and related circuits such as subtractors. This paper shows that the carry chains can implement compressor trees as well. Compressor trees are circuits that perform multi-input addition and partial product reduction for parallel multiplication using carry-save arithmetic, which specifically avoids long chains of carry-propagate addition. Multi-input addition occurs in many multimedia and DSP applications such as motion estimation [3] and FIR [4] filters; moreover, systematic transformations can also optimize circuits by merging distinct carry-propagate adders with one another and with the partial product reduction trees of parallel multipliers, creating large multi-input adders [5]. Accelerating these multi-input adders can significantly improve the overall performance of the application. In ASIC technology, these multi-input adders are realized as compressor trees, such as Wallace [6] or Dadda [7] trees. Compressor tree synthesis techniques for ASIC design flows, however, produce circuits that do not map well onto FPGA logic cells, with or without carry chains due to their irregular wiring structure.

Considering the fact that the fast carry-chains in FPGAs bypass the routing network, conventional wisdom has held that multi-operand addition should be implemented as trees of CPAs called adder trees. Parandeh-Afshar et al. [8] proposed a novel method to synthesize compressor trees on FPGAs using LUTs. This approach outperformed adder trees significantly, but did not utilize the carry chains. Their method uses *Generalized Parallel Counters (GPCs)*, which are introduced in Section 2.2, as the building blocks for the compressor trees. Compared to arithmetic components that

are used to synthesize compressor trees in ASIC design flows, such as carry-save adders and single-column parallel counters, GPCs offer more flexibility and improve logic utilization when mapped onto LUTs. Although this compressor tree synthesis method significantly reduced the critical path delay compared to synthesizing ternary adder trees [9], the technique used considerably more logic cells. The goal of this paper is to find a technique for compressor tree synthesis that retains the advantages of Parandeh-Afshar et al.'s technique, but without the area overhead.

Our proposal is a hybrid top-down/bottom-up heuristic for compressor tree synthesis that utilizes carry chains in order to improve area utilization. Compressor trees are built from GPCs, similar to [9], but by using a combination of LUTs and carry chains, rather than LUTs alone.

The bottom-up aspect of our approach uses *atom-level* modeling [10], in which primitive components are FPGA logic cells that are directly configured to implement GPCs. In this level of design, each GPC is built by a combination of LUTs, dedicated adders and carry-chains in a very compact form. In contrast, when each GPC is modeled in a higher level than atom-level, the synthesis tools are not able to utilize the logic cell resources efficiently.

The top-down aspect of our heuristic then synthesizes a compressor tree from a network of GPCs. In Parandeh-Afshar et al.'s [8] technique, the output bits of each GPC have the same logic delay: that of one 6-input LUT. As our new approach employs a carry chain, the low-order output bits of each GPC have smaller logic delays than the high-order bits. Our top-down synthesis method is made aware of these variations and makes judicious decisions based on this information to find the best interconnect network of GPCs in order to minimize logic delay. Compared to Parandeh-Afshar et al.'s technique, our new method reduced area by 20% on average with a negligible increase in critical path delay; compared to synthesis on ternary adder trees, critical path delay improved by 23% using our method, with a tolerable area overhead of just 11%.

The structure of the paper is as follows. Section 2 explains the *Altera Stratix-II Adaptive Logic Module (ALM)* which we have used for our design. The new compressor tree synthesis method, which is the main contribution of the paper, is discussed in Section 3. Experiments are presented in Section 4. Section 5 summarizes related work on FPGA arithmetic and synthesis, and Section 6 concludes the paper.

2. THE ALTERA STRATIX-II/III/V ADAPTIVE LOGIC MODULE (ALM)

The *Adaptive Logic Module (ALM)* is the logic cell used by *Altera* in the *Stratix II-IV* series of FPGA. Each ALM contains a variety of *look-up table (LUT)*-based resources that can be divided between two *adaptive LUTs (ALUTs)*. An ALM can operate in one of the following modes: *Normal*, *Extended LUT*, *Arithmetic* and *Shared-Arithmetic*.

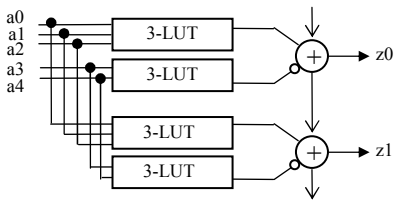

Fig. 1. ALM in arithmetic mode.

In normal mode, one ALM can implement various combinations of two logic functions of up to five inputs; the two ALUTs share some inputs, so some combinations of two logic functions may not be wholly independent of one another. The two ALUTs can be programmed to implement the same 6-input logic function; in this configuration, four inputs are shared between them. The ALM can also implement a restricted set of 7-input logic functions.

As an example example, a 6-input, 3-outputs circuit, where none of the outputs require the same logic function, requires 3 ALMs, but will only use one ALUT per ALM.

Each ALM has two dedicated full adders and a carry chain which are used in arithmetic and shared-arithmetic mode. Arithmetic mode provides two pairs of 3-LUTs which can implement a small logic function whose outputs are fed to a ripple-carry adder; the ALM inputs are shared between the LUTs and there are some constraints on the number of inputs shared between the LUTs [1]. Fig. 1 shows a legal configuration of an ALM in arithmetic mode.

Shared arithmetic mode is similar to arithmetic mode, but the interconnection pattern between the 3-LUTs and ripple-carry adder is different. The 3-LUTs are configured as a carry-save adder, whose outputs drive the ripple carry adder; this provides an efficient implementation of a 3-input (ternary) adder. Altera's literature [1, 9] indicates that one decisive factor in this design decision was that, ternary adder trees may have fewer levels than trees built from 2-input adders; however, they did not consider the possibility of synthesizing compressor trees.

3. HYBRID COMPRESSOR DESIGN APPROACH

This section presents the main contributions of the paper. The objective is to synthesize compressor trees onto ALMs as efficiently as possible both in terms of area and delay. We build compressor trees out of 6-input, 3-output GPCs; in actuality, only three such GPCs are required. Our mapping method is a hybrid top-down, bottom-up strategy.

The bottom-up aspect of our approach is to pre-compute mappings of each of the three GPCs onto two ALMs using atom-level modeling. Each GPC is built from LUTs, FAs, and carry chains. Although a carry chain physically spans all of the ALMs in a *Logic Array Block (LAB*—a cluster of several ALMs with sparse crossbar-based routing between them), two independent GPCs can be synthesized on consecutive pairs of ALMs: each GPC is designed such that

all carry-in and carry-out bits are always 0. In other words, we logically break the carry chain at the boundary of each GPC; this prevents carry propagation between adjacent GPCs, despite the existence of a physical carry chain between them. Fig. 2 shows a conceptual illustration; implementation details are presented in Section 3.2.

Once the GPCs are mapped onto ALMs, the top-down portion of our heuristic builds the compressor tree as a network of GPCs. This step is an extension of Parandeh-Afshar et al.'s [8] compressor tree synthesis technique; in that paper, each GPC was implemented by 6-input LUTs using the ALM in normal mode; consequently, each output has the same logic delay. When arithmetic mode is used, in contrast, the carry chain causes the output bits to have different logic delays. Consequently, the top-down compressor tree synthesis method presented here must account for these varying delays when constructing the network of GPC; details are provided in Section 3.3.

3.1. Generalized Parallel Counters (GPCs)

An *m:n* counter is a circuit that takes *m* inputs, counts the number that are set to 1, and outputs the result as an unsigned binary integer in the range *[0, m]*, which requires $n = \log_2(m+1)$ output bits. Compressor trees can be built using networks of counters; when doing so, the input bits to each counter have the same bit position, called the *rank*.

A *generalized parallel counter (GPC)* supports input bits of varying rank. A GPC is a tuple $P = (K_{N-2}, K_{N-1}, \ldots, K_1, K_0; W)$, where K_j is the number of input bits of rank j to sum, and W is the number of output bits; all of the input bits of each rank are assumed to be independent. For example, a *(3, 5; 4)* GPC can count up to three bits of rank *1* and five of rank *0*; the maximum output value is $5\times2^0 + 3\times2^1 = 11$, so $W = 4$ output bits are required. For the purpose of discussion, we fix the number of input and output bits to be positive constants, M and W. Given M and W, there exists a family of GPCs that satisfy these I/O constraints. The number of input bits cannot exceed M, and the maximum allowable output cannot exceed $2^W - 1$:

$$\sum_{j=0}^{N-2} K_j \le M . \qquad 0 \le \sum_{j=0}^{N-2} K_j 2^j \le 2^W - 1 \qquad (1)$$

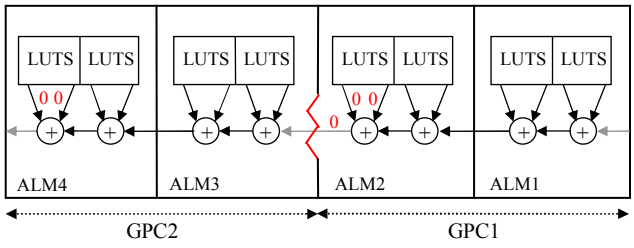

Fig. 2. Back-to-back GPC placement in a LAB.

For example, let $M = 8$ and $W = 4$. The *(3, 5; 4)* GPC belongs to this family, as it has 8 inputs and its maximum output value is *11*; on the other hand, a *(2, 2, 4; 5)* GPC has a maximum output value of *16*, which requires 5 output bits, and does not belong to this family.

The *compression ratio* of a GPC with M inputs and W outputs is M/W. It is favorable to use GPCs with large compression ratios, as they produce fewer output bits than those with smaller compression ratios; this, in turn, tends to reduce the number of logic levels in the compressor tree.

Each ALM has 6 inputs in arithmetic and shared-arithmetic mode; for this reason, we are interested in GPCs with 6 inputs, so that each GPC can be realized in one layer of logic in the FPGA. With 6 inputs, the highest compression ratios can be realized with 3 outputs, although 4- and 5-output GPCs are also possible.

Given M and W, a *primitive* GPC is one that satisfies the I/O constraints. GPC P_1 *covers* GPC P_2 if P_1 can implement the functionality of P_2 by setting some of its inputs to 0. For example, a *(2, 3; 3)* GPC covers a *(2, 2; 3)* GPC by setting one of the rank-0 inputs to 0. Given a family of GPCs, a *covering* GPC is one that is not covered by any other GPCs in the family.

Any GPC having $K_0 = 0$ or *1* is unreasonable; all others are *reasonable* [8]. For example, a *(3, 0; 3)* GPC takes no rank-*0* input bits, so the rank-*0* output bit is *0*; the *3* rank-*1* bits could be compressed using a *3:2* counter. Similarly, the rank-*0* output bit of a *(3, 1; 3)* GPC is equal to the rank-*0* input bit; if the input bit needs to propagate to a lower level of the compressor tree, it suffices to bypass the GPC and connect it directly to the lower level.

For $M = 6$ and $W = 3$, there are three GPCs that are reasonable and covering: *(0, 6; 3)*, *(1, 5; 3)* and *(2, 3; 3)*; there are a total of twelve reasonable GPCs, nine of which are covered by others. The bottom-up phase of our compressor tree synthesis heuristic is concerned with efficiently mapping these three GPCs onto ALMs.

3.2. Bottom-Up Approach: GPC Fitting

Here, each 6-input, 3-output, reasonable, covering GPC is synthesized on the *Stratix-II* ALM using atom-level modeling. One approach is to configure the ALM in normal mode, where three 6-LUTs can implement each GPC. As the logic functions for each GPC output are different, three ALMs per GPC are required. This is the GPC structure that *Altera*'s synthesis tool, *Quartus II*, will infer for a large compressor tree built from GPCs modeled normally using an HDL. The fitter is unable to infer the use of arithmetic mode, and cannot logically break the physical carry chain to pack one GPC into two ALMs. However, if the circuit synthesized is a single GPC, the fitter can infer the carry chains in some cases. So we need to model each GPC in atom-level which is discussed here.

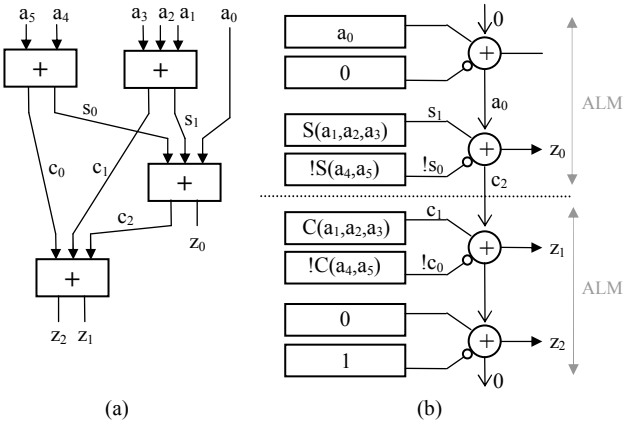

Fig. 3. (0,6;3) GPC: (a) circuit level (b) atom level.

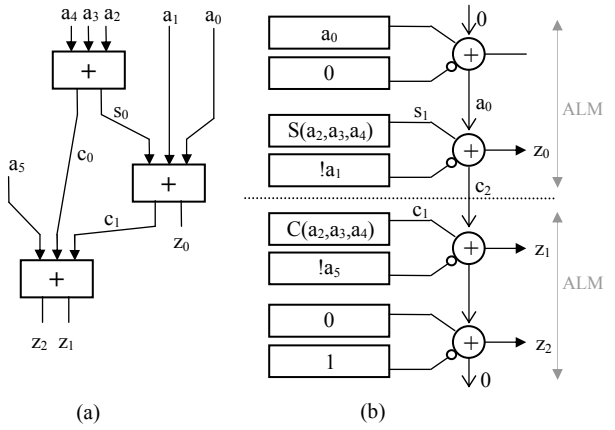

Fig. 4. (1,5;3) GPC: (a) circuit level (b) atom level.

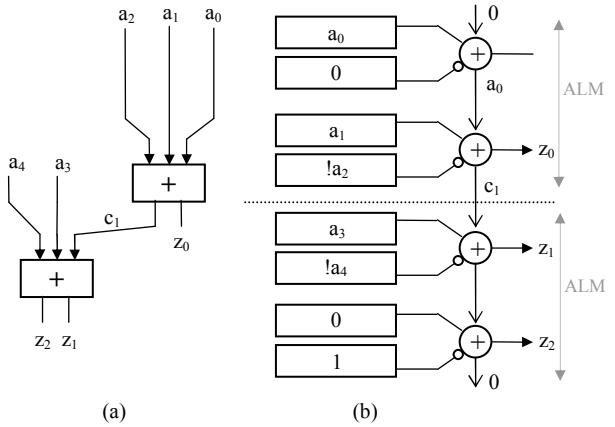

Fig. 5. (2,3; 3) GPC: (a) circuit level (b) atom level.

Figs. 3-5(a) show the circuit-level view of the three GPCs; Figs. 3-5(b) show the atom-level view of each GPC mapped onto the ALMs using arithmetic mode. The second input to each FA in the carry chain has an inverter, so the LUT that drives the input computes the inverse of the logic function that one would naturally expect to use.

An important point in the design of these three GPCs is that the carry-in bit of the first ALM and the carry-out bit of the second ALM are both 0. Thus, two GPCs can be placed on four consecutive ALMs within a LAB. To ensure that the second ALMs carry-out bit is zero, we must ensure that two of the three inputs to the fourth FA in the chain are 0. The timing analyzer must be able to recognize that the carry chain has effectively been "cut" by this mapping strategy, and that the carry-output of one GPC does not propagate to the carry-input of the next; otherwise, the timing analyzer would over-estimate the delay along an entire carry chain spanning a whole LAB.

As each GPC produces three outputs, the first output of the first ALM is never used; additionally, the delay of the three outputs differs based on their respective positions along the carry chain. The variation in output delay creates a new challenge when building a compressor tree out of GPCs. A proper connection of GPCs will lead to a more balanced GPC network in terms of delay; this will be fully detailed in the following section.

3.3. Top-Down Approach: GPC Mapping

The preceding section described how to map each 6-input, 3-output reasonable covering GPC onto ALMs at the atom-level. This section describes a new method to synthesize a compressor tree using these GPCs as components. This method is an extension of Parandeh-Afshar et al.'s [8] heuristic, in which GPCs were previously mapped onto ALMs using normal mode, rather than arithmetic mode.

The primary objective of the mapping heuristic is to minimize the height of the compressor tree; minimizing the total number of GPCs used is a secondary objective. To this end, we prioritize the three GPCs by their compression ratios. The *(0, 6; 3)* and *(1, 5; 3)* GPCs both have compression ratios of *3*, while the *(2, 3; 3)* GPC has a compression ratio of *2.67*. Given a choice of using the three GPCs at any point, the first two are preferable to the third due to their higher compression ratios.

Fig. 6 shows the pseudo-code of the mapping heuristic. The inputs are the integers M and W, which characterize the family of GPCs we will use (for the *Altera Stratix II-IV* FPGA families $M = 6$ and $W = 3$), along with a set of bits to sum. A *column* is defined to be a set of bits having the same rank; the third input is an array of integers, where the i^{th} integer represents the number of input bits in the i^{th} column. The heuristic has three main steps:

Step 1. The first step, given M and W, builds a library of primitive and covering GPCs; the primitive GPCs are sorted twice: first by decreasing order of compression ratio; then, those having equal compression ratios are sorted again by increasing order of critical path delay.

```
Mapping_algorithm(Integer : M, Integer : W,
                  Array of Integers : columns )
{
Step1:   Build_GPC_library( );

         Repeat
         {
Step2:       While (col_indx ≤ max_col_indx)
             {
                if(columns[col_indx] ≥ 6)
                      Map_by_GPC();
                else
                      col_indx++;
             }
             lsb_to_msb_covering();

Step3:       Connect_GPCs_IOs( );
             Propagate_comb_delay( );
             Generate_next_stage_dots( );

         } Until three rows of dots remains;
}
```

Fig. 6. GPC mapping heuritic.

The first step of the algorithm can be performed once for pair *M* and *W*, and then stored in a database offline. Steps 2 and 3, however, must be repeated each time a new compressor tree is synthesized. Steps 2 and 3 iterate until at most three bits per column remain; at this point, a ternary CPA, built using a layer of ALMs in shared arithmetic mode, sums the remaining bits.

Step 2. In the second step, the current inputs to be added are mapped to the primitive GPCs. The highest priority GPC is *(0, 6; 3)*; it is used whenever a column is found having at least 6 bits. This process repeats until all columns have fewer than 6 bits remaining. Afterwards, the remaining bits are covered using Parandeh-Afshar et al.'s heuristic [8], which makes a single pass from the least significant column to the most significant column.

Step 3. Step 2 covers all of the bits at one level of the compressor tree with GPCs. The outputs from the GPCs generated by the previous iteration of the algorithm (or the compressor tree inputs) must be connected to the layer of GPCs introduced by Step 2. Each bit of rank *r* produced by a GPC (or compressor tree input) at layer *i-1* must be connected to a GPC input of rank *r* at layer *i*. For a given bit, there will be many GPCs having inputs of the same rank, and the choice of connection may affect the critical path of the compressor tree. Our strategy is to connect the delay-critical outputs of GPCs at stage *i-1* to the GPC inputs of the same rank at stage *i* whose delays are less.

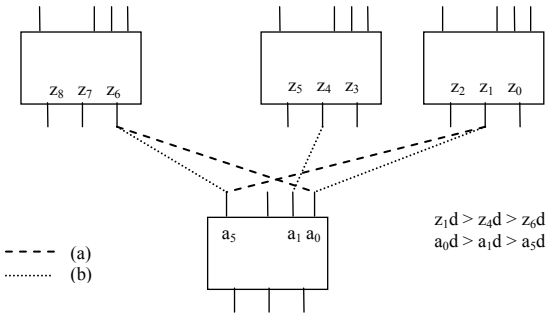

Fig. 7. Two equivalent circuits with different delays.

For example, in Fig. 4, for the *(1, 5; 3)* GPC, input a_5 goes through one LUT and one FA, and has a lower delay than the other inputs; inputs a_2, a_3, and a_4 go through one LUT and two FAs; and inputs a_0 and a_1 go through one LUT and three FAs. Given a choice, the output bit from layer *i-1* whose delay is largest should be connected to a_5.

Fig. 7 illustrates how two different interconnection schemes can affect the critical delay of a circuit. Two options exist to connect the outputs of three GPCs in layer *i-1* to the inputs of one GPC in layer *i*. The relation between the delays is shown as well. The critical paths of the two interconnection schemes, respectively, are:

$$CP_{(a)} = max(z_1d+a_5d,\ z_4d+a_1d,\ z_6d+a_0d), \text{ and} \quad (3)$$

$$CP_{(b)} = z_1d+a_0d. \quad (4)$$

Based on the delays shown in Fig. 7, it is clear that the delay of circuit (a) is less than the delay of circuit (b).

After the outputs of the GPCs in layer *i-1* are connected to the inputs of the GPCs in layer *i*, the delays of the GPC outputs of layer *i* are computed; these delays are used to drive the interconnection task at the following level.

Lastly, we must generate the columns of bits to be summed at layer *i+1*. These include the GPC output bits from layer *i*, and any leftover bits from layer *i-1* that were not mapped onto a GPC at layer *i*.

4. EXPERIMENTAL EVALUATION

This section evaluates the new compressor tree synthesis heuristic and compares it to synthesis using ternary adder trees (shared arithmetic mode) and Parandeh-Afshar et al.'s compressor tree synthesis heuristic (normal mode) [8]. The target FPGA is an *Altera Stratix-II*.

4.1. Experiment Setup

To model the GPCs at the atom-level, we used the *Verilog Quartus Module (VQM)* format provided by the *Quartus-II University Interface Program (QUIP)* [10]. VQM is a

restricted form of the Verilog language standard in which the basic components are logic cells with parameters such as LUT masks and configuration mode, which are defined by the user. The three GPCs in Figs. 3-5 were modeled using VQM; these models were then used as components from which larger compressor trees were constructed. The delay profile of each GPC was extracted by synthesizing it on the *Stratix-II*. The mapping heuristic in Fig. 6 was implemented in C++, and the delay profiles of each GPC were provided. The output is a structural VHDL implementation for each compressor tree, which is itself a netlist of GPCs, followed by a ternary adder. *Altera*'s *Quartus II* CAD tool was used for our experiments.

We were surprised to discover that *Quartus II* does not pack GPCs adjacent to one another in a LAB. Each LAB contains eight ALMs, and a carry chain can be initiated from either the first or fourth ALM. *Quartus II*'s fitter places each GPC either in the first/second or fourth/fifth ALMs, and does not use the others. To solve this problem, we grouped every pair of GPCs together, in mapping order, and established a virtual carry chain between them. The timing analyzer recognized that the virtual carry is not a real carry and did not overestimate the delay of the circuit.

4.2. Benchmarks

The benchmarks we used in this study are summarized in Table 1. These benchmarks are compressor trees taken from arithmetic, DSP, and video processing applications. Among the benchmarks, DCT, H.264 ME, FIR3, FIR6, SM9x9, and SM18x18 naturally contain multi-input addition operations which are best implemented as compressor trees. HPoly, G.721, and Video Mixer are larger circuits that are transformed as described by Verma and Ienne [5] to expose large compressor trees. Most of these benchmarks are publicly available with a few exceptions. The FIR filters were built using randomly generated constants; and Video Mixer is provided by *Synopsys Corporation* as an example to illustrate their *Behavioral Optimization of Arithmetic (BOA)* tool.

Table 1. Benchmark summary.

Benchmark	Description
DCT	Multiplierless DCT
HPoly	Horner polynomial Eval.
H.264 ME	H.264 motion estimation
G.721	G.721 encoder
FIR3, FIR6	3- and 6-tap FIR filters
SM9x9, SM18x18	Parallel signed multipliers
vm.add2I, vm.add2Q	Video mixer

4.3. Results

We compare four different methods to synthesize each adder/compressor tree; in all cases, the compressor tree, including final adder, was synthesized as a purely combinational circuit.

Ternary: Ternary adder trees [9].

Orig-LUT: Parandeh-Afshar et al.'s [8] mapping heuristic with GPCs mapped onto LUTs (normal mode).

Orig-Arith: Parandeh-Afshar et al.'s [8] mapping heuristic with GPCs mapped onto ALMs as described in Section 3.2 (arithmetic mode).

New-Arith: The mapping heuristic shown in Fig. 6 with GPCs mapped onto ALMs as described in Section 3.2 (arithmetic mode).

Fig. 8(a) shows the critical path of the compressor trees. In all cases, Ternary has the largest critical path delay, which is to be expected, given the results of Parandeh-Afshar et al. [8]. Orig-LUT and New-Arith have comparable critical path delays for most benchmarks, however, there exist a few benchmarks for which one or the other retains a slight advantage. Compared to the other two compressor tree synthesis methods, Orig-Arith fares quite poorly; the reason is that Orig-Arith does not account for variations in the delays of the output bits due to the use of carry chains. If we begin with the premise that the use of arithmetic mode is preferable due to area utilization, it is quite clear that New-Arith is superior to Orig-Arith in terms of reducing critical path delay.

Fig. 8(b) reports the number of ALMs required for each compressor tree synthesis technique. Ternary achieves the smallest design, which is to be expected, because each ALM in shared arithmetic mode compresses six input bits down to two (ignoring the carries); the drawback in this case is the critical path delay due to the use of carry chains at each stage of the compressor tree. Each GPC, in contrast, compresses five or six input bits down to three. Orig-LUT requires three ALMs per GPC, while Orig-Arith and New-Arith require two. This gives Orig-Arith and New-Arith significant advantages in terms of area utilization compared to Orig-LUT.

On average, New-Arith improves the critical path delay by 8% compared to Orig-Arith. Furthermore, the delay gap between New-Arith and Orig-LUT is only 2%. On average, New-Arith improves area utilization by 20% compared to Orig-LUT. As Ternary and Orig-Arith is not competitive in terms of critical path delay, and Orig-LUT is not competitive in terms of area, we find that New-Arith is the most effective of the three compressor tree synthesis methods evaluated here.

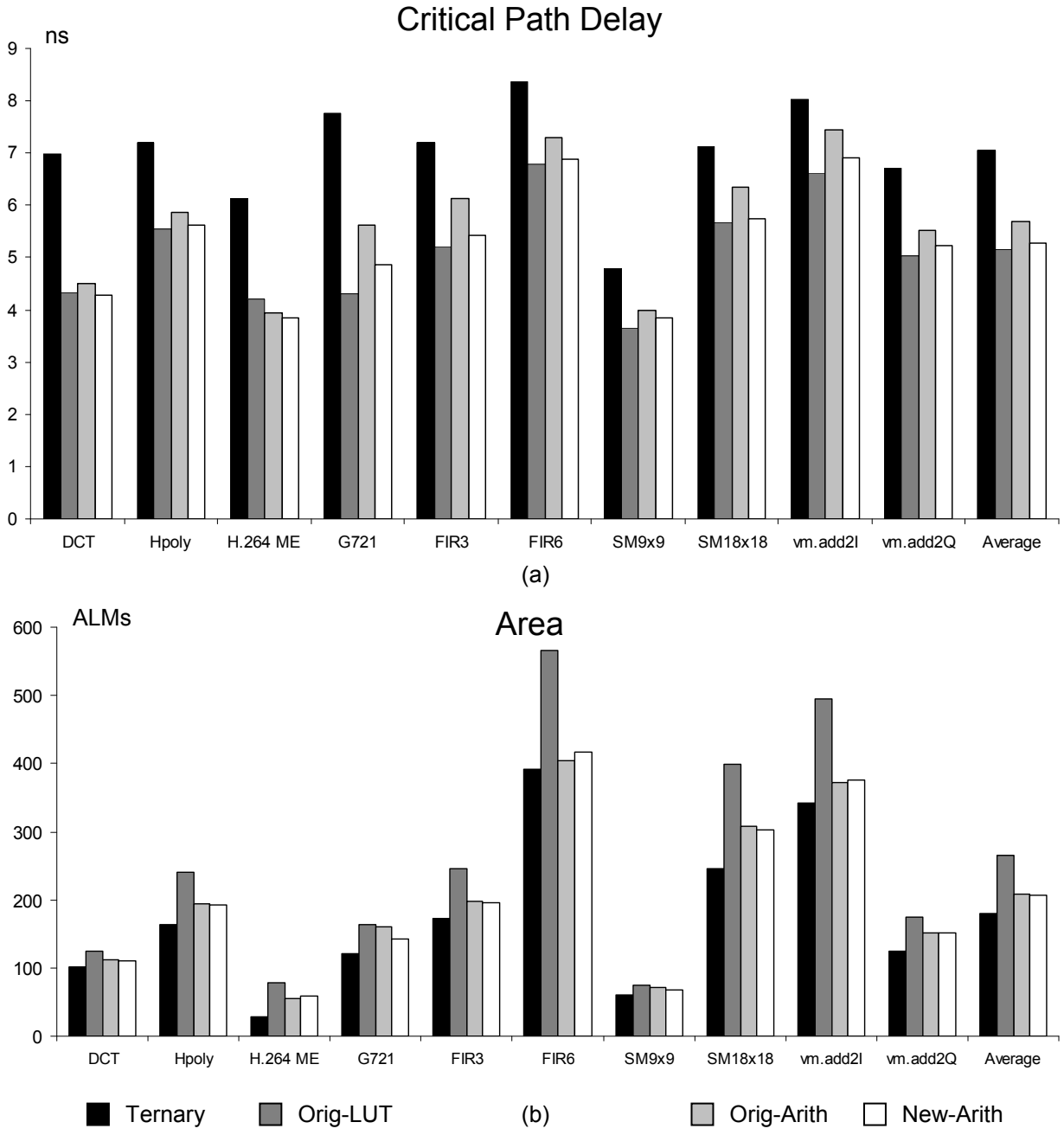

Fig. 8. Delay (a) and area (b) obtained by synthesizing multi-input addition on a Stratix-II FPGA.

5. RELATED WORK

Wallace [6] and Dadda [7] introduced compressor trees built using carry save adders. Their work was in the context of parallel multipliers, in which the number of input bits per column varies. Within the compressor tree, bits having the same rank may differ significantly in terms of delay. To exploit this fact, Stelling et al. [11] described an optimal algorithm for compressor tree construction called *3-greedy (3GD)*.

Um and Kim [12] proposed a layout-aware synthesis method for compressor trees that tries to minimize wirelength between CSAs. This method, along with the 3GD algorithm and Wallace and Dadda trees build compressor trees from *2:2* and *3:2* counters, and are appropriate for ASIC design flows, but not for FPGAs.

Poldre and Tammemae [13] constructed a compressor tree for parallel multipliers from *4:2 compressors* and synthesized them on *Xilinx Virtex* FPGAs, exploiting the

carry-chains present in those devices. They reported delays that were 1.5x faster and used 1.28x less area than standard adder trees. FPGA architectures, however, have been enhanced since this work, and it is not clear whether these older techniques still suffice. In particular, the LUT size has been increased from 4 to 6 inputs, and both *Altera* (*Stratix-II-IV*) and *Xilinx* (*Virtex-5*) developed methods to use LUTs in conjunction with the carry-chains to perform ternary addition; the advantage of integrating hardware support for ternary addition within FPGA logic blocks was noted and evaluated in an *Altera* white paper [9].

Parandeh-Afshar et al. [8] proposed a compressor tree synthesis for FPGAs, that this paper has extended. Although the delay was improved, they incurred significant area overheads. In a subsequent paper [14], they formulated the problem as an integer linear programming (ILP) and were able to reduce the area requirement. In principle, this ILP could be modified to account for the varying delays of each GPC output as well, and could therefore be adapted in place of the mapping heuristic proposed here in Fig. 6.

Frederick and Somani [15] mapped general logic operations, beyond arithmetic operations, onto the carry-select chains used in *Altera*'s *Stratix* and *Cyclone* FPGAs. *Altera* has used a different carry chain in the *Stratix II-IV* FPGAs. The authors noted that their method is specific to the older carry-select chain, and cannot be in conjunction with the newer *Stratix II-IV* carry chains or the carry chains employed in *Xilinx* FPGAs. Our work is similarly specific to the carry chains in the *Stratix II-IV* FPGAs, which have remained in place over several generations of devices in the *Stratix* family.

6. CONCLUSION

This paper has proposed a new technique to synthesize compressor trees on FPGAs using LUTs in conjunction with carry chains. 6-input, 3-output GPC were designed at the atom-level, and the compressor trees were constructed as a network of GPCs. Building GPCs using LUTs in conjunction with carry chains reduced the number of ALMs required to realize each GPC from three to two, compared to synthesizing them using LUTs alone. This altered the mapping problem from the perspective of delay optimization, however, as the delay of each bit produced by a GPC depends on its position along the carry chain. To compensate, a new mapping technique was proposed that was aware of these delays. The proposed technique retains the benefits of prior compressor tree synthesis methods in terms of critical path delay, while significantly reducing the area utilization. A previous compressor tree synthesis method that does not account for the variation in delay of the outputs of each GPC performed poorly in comparison.

7. REFERENCES

[1] Altera, Corp. "The Stratix II device handbook, vol. 1 and 2," available online from http://www.altera.com/

[2] Xilinx Corp. "Virtex-4 user guide," available online from http://www.xilinx.com/

[3] C-Y. Chen, S-Y. Chien, Y-W. Huang, T-C. Chen, T-C. Wang, and L-G. Chen, "Analysis and architecture design of variable block-size motion estimation for H.264/AVC," *IEEE Trans. Circuits and Systems-I*, Vol. 53, No. 2, pp. 578-593, Feb., 2006.

[4] S. Mirzaei, A. Hosangadi, and R. Kastner, "High speed FIR filter implementation using add and shift method," *Int. Conf. Computer Design*, San Jose, CA, USA, Oct. 1-4, 2006.

[5] A. K. Verma, P. Brisk, and P. Ienne, "Data-flow transformations to maximise the use of carry-save representation in arithmetic circuits," *IEEE Transactions on Computer-Aided Design,* Vol. 27, pp. 1761-1774, 2008.

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

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

[8] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in Proceedings of the *Asia-South Pacific Design Automation Conf.*, Seoul, Korea, January, 2008, pp. 138-143.

[9] Altera, Corp. "Stratix II vs. Virtex-4 performance comparison," available online from http://www.altera.com/

[10] Altera, Corp. " Quartus-II University Interface Program," http://www.altera.com/education/univ/research/unv-quip.html

[11] P. F. Stelling, C. U. Martel, V. J. Oklobdzija, and R. Ravi, "Optimal circuits for parallel multipliers," *IEEE Trans. Computers*, Vol. 47, No. 3, pp. 273-285, March, 1998.

[12] J. Um, and T. Kim, "Layout-aware synthesis of arithmetic circuits," *Design Automation Conf.,* pp. 207-212, June 10-14, 2002.

[13] J. Poldre and K. Tammemae, "Reconfigurable multiplier for Virtex FPGA family," *Int. Workshop on Field-Programmable Logic and Applications*, Glasgow, UK, pp. 359-364, Aug. 30 – Sept. 1, 1999.

[14] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving synthesis of compressor trees on FPGAs via integer linear programming," in Proceedings of the *International Conference on Design Automation and Test in Europe*, Munich, Germany, March, 2008.A. K.

[15] M. T. Frederick and A. K. Somani, "Beyond the arithmetic constraint: depth-optimal mapping of logic chains in LUT-based FPGAs," in Proceedings of the *ACM/SIGDA International Symposium on FPGAs*, Monterey, CA, USA, 2008, pp. 37-46.