A Novel FPGA Logic Block for Improved Arithmetic Performance

Hadi Parandeh-Afshar Philip Brisk Paolo Ienne
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland
{hadi.parandehafshar, philip.brisk, paolo.ienne}@epfl.ch

ABSTRACT
To improve FPGA performance for arithmetic circuits, this paper proposes a new architecture for FPGA logic cells that includes a 6:2 compressor. The new cell features additional fast carry-chains that concatenate adjacent compressors and can be routed locally without the global routing network. Unlike previous carry-chains for binary and ternary addition, the carry chain used by the new cell only spans 2 logic blocks, which significantly improves the delay of multi-input addition operations mapped onto the FPGA. The delay and area overhead that arises from augmenting a traditional FPGA logic cell with the new compressor structure is minimal. Using this new cell, we observed an average speedup in combinational delay of 1.41× compared to adder trees synthesized using ternary adders.

Categories and Subject Descriptors
B.6.1 [Logic Design]: Design Styles–FPGAs; B.2.4 [Arithmetic and Logic Structures]: High-Speed Arithmetic–cost/performance

General Terms: Design, Performance.

Keywords
FPGA, Compressor Tree, 6:2 Compressor, Multi-operand Addition, Carry-chain, Arithmetic Circuits

1. INTRODUCTION
The performance gap between FPGAs and ASICs is generally exacerbated for arithmetic circuits, compared to state machines and control-dominated circuits, despite numerous architectural improvements over the past 20 years. Kuon and Rose [19] have recently shown that hard, hand-optimized IP cores—namely DSP and MAC blocks—do not offer tangible performance advantages due to the high cost of routing data to and from these blocks, and mismatches in bitwidth. Previous enhancements to FPGA logic blocks, such as support for binary and ternary addition [1, 3, 40, 41] and carry-chains [8, 12, 14, 18, 21], have improved FPGA performance for arithmetic operations. Nonetheless, the performance gap between ASICs and FPGAs remains.

FPGA'08, February 24–26, 2008, Monterey, California, USA.

Multi-input addition is a very important arithmetic operation. It occurs in applications such as FIR filters [22], 3G wireless base station channel cards [2, 28], and motion estimation in video coding applications such as H.264/AVC [7]; partial product compression in parallel multiplication [10, 38] is also a form of multi-input addition. Somewhat more general in scope, Verma and Ienne [36] recently proposed a set of circuit transformations that can merge disparate addition operations into one large multi-input addition operation. Their technique separates each multiplication operation into two parts: partial product generation and partial product compression. The transformations can merge partial product compressors arising from multiplication with other addition operations and other compressors as well. Due to the generality of these transformations, we believe that FPGA architectures can significantly benefit from modifications designed to enhance the performance of multi-input addition.

In ASIC design, it is well-known that the best implementation for multi-input addition is to build *compressor trees* using *carry-save adders (CSAs)* [10, 38], *parallel counters* [32, 33], and other components such as *4:2* and *5:2* compressors [20, 39]. These circuits will be introduced in Section 2.5.

In particular, *adder trees* constructed from *carry-propagate adders (CPAs)* such as *ripple-carry* or *carry-select* do not perform well at all. The reason is that the critical path delay of a CPA, regardless of its specific implementation, is from the carry-in bit to the carry-out. Compressor trees are built using a more efficient structure, in such a manner that a CPA is only needed to perform a final addition, after all of the bits to be summed have been compressed down to 2 or 3 rows.

Due to the structure of FPGAs—and particularly, the dedicated adder circuitry and fast carry-chains—the conventional wisdom has been that multi-input addition is best implemented using adder trees rather than compressor trees. The reason is twofold. First and foremost, it was thought the compressor trees are not efficiently synthesized onto LUTs. Secondly, the reduction in delay due to the fact that the fast carry-chain does not pass through the routing network was thought to offset the superior arithmetic structure of a compressor tree.

A significant performance improvement for multi-input addition was realized by the logic architecture of the Altera Stratix II [1]. Prior to this, the dedicated addition circuitry in high-end FPGAs supported binary (2-input) addition [40]. The Stratix II architecture allowed the *look-up tables (LUTs)* to be configured as 3:2 compressors, which were then fed into a binary adder; together, this structure yielded ternary (3-input) addition. If n input operands are to be summed, $log_2 n$ logic layers are needed if binary adders are used, and $log_3 n$ layers are needed in the case of ternary adders.

A performance comparison between the Stratix II and Virtex-4 showed that the novel ternary adder structure yielded a significant performance improvement [2]. Altera has kept this basic structure in their Stratix III devices [3], while Xilinx has added support for ternary addition to the Virtex-5 [41].

Parandeh-Afshar et al. [24] recently discovered a method to successfully map compressor trees onto FPGAs (without using dedicated adders or fast carry-chains, except for the final addition) using a component called a *generalized parallel counter (GPC)* [31], which is introduced in Section 2.4. Our first attempt at GPC mapping used a relatively straightforward greedy heuristic, and achieved a reduction in critical path delay of 27%, on average, compared to an adder tree built from ternary adders on the Stratix II. Unfortunately, the compressor trees required 11% more *adaptive logic modules (ALMs)* than the adder trees.

A second attempt formulated GPC mapping as an *integer linear program (ILP)*, which was solved using a commercial tool [25]. Using the ILP, the compressor trees required 3% fewer ALMs, on average, than the ternary adder tree. Due to the reduced ALM count, we were able to achieve a tighter placement, which reduced the average wire length. This, in turn, yielded a 32% reduction in critical path delay compared to the ternary adder tree.

The GPC mapping techniques described above synthesized 6-input GPCs on the 6-input LUTs of modern high-performance FPGAs. This paper, in contrast, introduces architectural modifications to a standard FPGA logic cell to include support for a circuit called a 6:2 compressor. The 6:2 compressors have fast carry-chains and are constructed from dedicated adders, similar to those used for ternary addition in modern FPGAs. Unlike prior carry chains, however, the carry chain in a 6:2 compressor does not propagate beyond 2 logic blocks. Most importantly, the GPC mapping heuristics, which are already successful, can be extended to exploit the 6:2 compressor in the proposed logic cell to further reduce both delay and area of multi-input addition operations. In our experiments, a speedup of approximately 1.41× was observed compared to ternary adder trees, with minimal area overhead.

2. ARITHMETIC PRELIMINARIES

Here, we introduce concepts from digital arithmetic that are relevant to this work. We begin with some nomenclature regarding unsigned binary numbers (Section 2.1). We then introduce compressor trees (Section 2.2), and the components used to build them: single-column parallel counters (Section 2.3), generalized parallel counters (GPCs) (Section 2.4), and compressors (Section 2.5).

2.1 Nomenclature

Let $B = (b_{n-1}, b_{n-2}, \ldots, b_0)$ be an n-bit binary number, where each b_j, $0 \le j \le n-1$ is a bit. The *rank* of a bit is defined by its subscript j; specifically, a bit of rank j contributes the quantity $b_j 2^j$ to the value of B.

In multi-input addition, we are given a set of k *n-bit* unsigned binary numbers, $B_0, \ldots, B_{k-1}$, and the goal is to compute their sum. The i^{th} number is: $B_i = (b_{i,n-1}, b_{i,n-2}, \ldots, b_{i,0})$.

A *column* is a set of input bits having the same rank. For example, the j^{th} column, denoted C_j is the set: $C_j = \{b_{0,j}, b_{1,j}, \ldots, b_{k-1,j}\}$. When constructing a compressor tree, the input is often represented as a set of n columns, $C_0, C_1, \ldots, C_{n-1}$ rather than a set of binary numbers.

2.2 Compressor Trees

Compressor trees are a general class of circuits that perform multi-input addition much more efficiently than adder trees. Techniques for compressor tree construction will be discussed in Section 3.1 in the context of related work. In short, a compressor tree takes $B_0, B_1, \ldots, B_{k-1}$ as inputs, and produces two outputs, *sum (S)* and *carry (C)*, such that:

$$S + C = \sum_{0 \le i \le k-1} B_i \,. \tag{1}$$

A CPA is then used to perform the final addition, $S+C$.

The logic cells in the Altera Stratix III and Xilinx Virtex-5, support ternary addition. For such a system, a slightly smaller compressor tree could be generated that produces three outputs, O_1, O_2, and O_3, rather than two outputs. The ternary adder then produces the final sum: $O_1 + O_2 + O_3$. This is the approach that we have taken in our prior work for mapping compressor trees onto FPGAs [24, 25], and we take the same approach here.

2.3 Single-Column Parallel Counters

A *single column parallel counter* [33] is a circuit that takes m input bits, each of the same rank (hence the term "single-column"), counts the number of input bits that are set to 1, and produces the output as an unsigned binary number. The number of output bits, n, required to represent a value in the range $[0, m]$ is:

$$n = \left\lceil log_2 (m+1) \right\rceil \tag{2}$$

Henceforth, we will refer to a single-column parallel counter as a *m:n* counter, where m and n are the number of input and output bits respectively.

In arithmetic design, *3:2* and *2:2* counters are known as *full* and *half adders* respectively. In compressor tree generation, a *3:2* counter can also be called a *carry-save adder (CSA)*. Fig. 1(a) shows an example of a compressor tree (without the final adder) built from 2 CSAs.

2.4 Generalized Parallel Counters

An *m:n* counter described in the previous section can only count bits of the same rank. A *generalized parallel counter (GPC)* [32] is an extension of the same concept, except its inputs are bits from a multitude of columns of different ranks.

Formally, a GPC is defined as follows: $(m_{k-1}, m_{k-2}, \ldots, m_0; n)$. The inputs to the GPC are m_0 bits of rank 0, m_1 bits of rank 1, etc. n is the output of GPC which is a value in the range $[0, M]$, where

$$M = \sum_{0 \le i \le k-1} m_i 2^i \tag{3}$$

The number of output bits, can be computed by substituting M for m in Eq. (2). Fig. 1(b) shows an example of a (3, 4; 4) GPC.

2.5 Compressors

Compressors (not to be confused with compressor trees) are similar to *m:n* counters and GPCs, but they also feature carry-in/carry-out bits, and some of the output and carry-out bits may have the same rank. Many compressor trees in the past have been constructed from a combination of compressors and *m:n* counters.

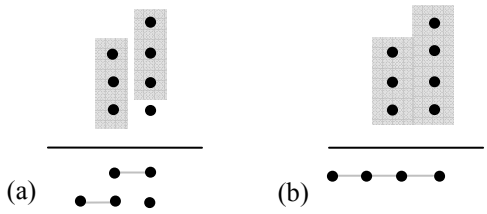

Figure 1.

Illustration of column compression using CSAs (a) and a (3, 4; 4) GPC (b)

One example is the *4:2 compressor* [39], which takes 4 inputs bits of rank-0 and one carry-in bit, and produces two output bits (of ranks 0 and 1) and one carry-out bit of rank 1 (the name is rather confusing because it actually takes 5 input bits and produces 3 output bits). Other compressors that have been proposed in the past for multiplier design include *5:2*, *5:3*, [20] and *9:2* [28]. In this paper, we propose to augment an FPGA logic cell with functionality to implement a *6:2* compressor. We will describe the *6:2* compressor in detail and show how to interconnect them to form larger compressors in Section 4.1; Fig. 3, which is also part of Section 4.1, illustrates these concepts.

3. RELATED WORK

Here, we summarize related work that is relevant to this paper. Section 3.1 describes prior techniques to synthesize compressor trees using the multitude of components introduced in Section 2. Sections 3.2 and 3.3 respectively summarize work in FPGA mapping algorithms and architecture, focusing on arithmetic circuits. Section 3.4 describes the Altera Stratix II ALM in detail; our implementation and experiments focus on this cell.

3.1 Compressor Tree Synthesis

Compressor trees for partial product accumulation were introduced in 1964 and 1965 by Wallace [38] and Dadda [10], who built them from CSAs; HAs were used at points where only 2 bits in the same column need to be compressed.

Fadavi-Arkedani [11] recognized that the bits produced by a compressor tree may arrive at different times at the final adder, and designed a specific adder for this purpose; however, this work assumed that all partial product bits arrive to the compressor tree at the same time. Stelling et al. [30, 31] relaxed this assumption, and developed appropriate techniques to build the compressor tree and design the final adder appropriately.

Due to the importance of wire delays in deep submicron technology, Um and Kim [34] proposed a two-phase layout-aware compressor tree synthesis technique that strives for a much more regular interconnect topology than the compressor trees produced by the 3-greedy algorithm.

Verma and Ienne [35] developed an integer linear program (ILP) that could optimally synthesize compressor trees from a library of *m:n* counters. To bound the runtime of the synthesis procedure, they limited *m* to the range *[1, 8]*. Previously, *m:n* counters, like compressor trees, were built from CSAs, or libraries of smaller *m:n* counters. Through efficient logic synthesis techniques for arithmetic circuits [37], Verma and Ienne found that better *m:n* counters could be constructed from basic gates, rather than smaller counters. The availability of a library of highly optimized counters was important to the success of their ILP formulation; another contributing factor was that the ILP could optimize for the delay profile of any final adder.

GPCs have also been used in the past to build efficient compressor trees for parallel multipliers [32]. Mora Mora et al. [23] described a multiplier generation approach for ASICs that implemented GPCs using ROMs, with the restriction that all input columns to the GPC have the same number of bits.

The *4:2* compressor was introduced by Weinberger [39], and subsequently used by Santoro and Horowitz in a *64×64* parallel multiplier [27]. Over the years, various researchers have proposed the use of larger compressors as well, including Kwon et al. (*5:2/5:3*) [20] and Song and De Micheli (*9:2*, *27:5*) [28].

3.2 FPGA Mapping for Arithmetic Circuits

All of the aforementioned work focused on multipliers designed for ASICs. Most of these techniques would not yield good circuits, if applied directly to FPGAs, because the relative delays of logic and general routing are completely different for FPGAs and ASICs, while the carry-chains are more efficient. Since the primary role of the carry-chains was to facilitate efficient carry-propagate addition, the conventional wisdom was that adder trees would yield better results than compressor trees.

Parandeh-Afshar et al. [24] showed that compressor trees could efficiently be synthesized onto FPGAs using GPCs. They limited the number of inputs to a GPC to 6, the number of LUT inputs of high-performance FPGAs. This ensured that the delay of each GPC is equivalent to one logic level of the FPGA (plus routing, which is harder to predict during mapping). On an Altera Stratix II, the delay of a compressor tree built from GPCs was 27% faster than that that of an adder tree. The GPC mapping, however, increased the ALM count by 11%, on average.

The mapping technique described above used a relatively straightforward greedy heuristic. In a subsequent work, they reformulated the mapping as an ILP [25]. Using the ILP, the delay was 32.0% less than ternary adder trees, and the ALM count was slightly smaller than an adder tree, as well. Part of the reduced delay came from the fact that fewer ALMs were used, which allowed for a tighter placement, which in turned reduced wirelength, and hence wire delays.

Poldre and Tammemae [26] synthesized *4:2* compressors onto the 4-input LUTs of the Xilinx Virtex FPGAs, exploiting the carry-chains to propagate the carry-in/carry-out bits. The *6:2* compressors that we would like to use require 2 carry-chains, so we have chosen to redesign the logic cell structure to accommodate this feature.

3.3 FPGA Architecture

The new logic cell proposed in this work features a new type of carry-chain intended to allow a logic cell to be configured as a *6:2* compressor. The Altera Stratix II/III ALM employs a ripple-carry chain [1, 3], and the Xilinx Virtex-4/5 chain includes programmable multiplexers and *xor* gates to send propagate and generate signals to adjacent CLBs [40, 41].

Hauck et al. [14] proposed more complicated carry-chains that can implement *Brent-Kung*, *carry-select*, and *carry-lookahead* addition. Different logical constructs were needed for different cells in the chain, making them non-uniform. This creates integration challenges because it is difficult to layout a regular fabric consisting of irregular cells. This would require a large manual effort to design each individual cell at the transistor level, and would complicate the layout process for the entire chip.

Frederick and Somani [12] proposed a uniform logic block with carry-chains that could efficiently implement a carry-skip adder; a similar bi-directional carry-skip chain was earlier proposed by Cherepacha and Lewis [8, Fig. 6]. Kaviani et al. [18] and Leijten-Nowak and van Meerbergen [21] developed ALU-like blocks that support arithmetic functions such as addition, subtraction and (partial) multiplication. For multi-operand addition, our GPC mapping techniques [24, 25] would not use these structures. GPCs, which do not use these carry chains, have fewer logic layers than adder trees that do use them. The carry-chains described here are designed specifically to be useful to GPC mapping.

Distributed Arithmetic (DA) [22] is a paradigm for implementing effective hardware for DSP systems that uses LUTs instead of multipliers. Grover et al. [13] developed a special DA-oriented LUT structure (*DALUT*) specifically for MAC operations. In addition to two 4-input LUTs, their DALUT cell included arrays of *xor* gates, bit-level adders and shift accumulators, shift registers, and a CPA to add partial summations and carries. Brisk et al. [6] reported that DSP/MAC blocks are not good candidates for implementing multi-operand addition. The logic cell described here is intended to address this shortcoming.

Most FPGAs are now hybrid-reconfigurable devices, in the sense that they include pre-placed ASIC-like hard IP blocks, such as multipliers, DSP/MAC blocks, and standard I/O interfaces [42]. Kastner et al. [17], for example, developed techniques by which a compiler could identify a set of applications to identify good candidates for IP cores; their analysis, however, was limited to 2-operation combinations of addition and multiplication, and they did not consider the use of compressor trees for multi-operand addition.

A *K-input macro gate* [9] is similar to a LUT, but it cannot implement all 2^K logic functions, and therefore has reduced delay and area. Hu et al. [16] suggested that FPGA cells could benefit from the inclusion of both LUTs and macro gates. Similar to Kastner et al., they developed an automated method to profile a set of applications to find good macro-gate candidates. They did not, however, consider arithmetic-dominated functions [15], or fast carry-chains between macro gates.

The *Field Programmable Counter Array (FPCA)* [6] is a programmable IP used to accelerate multi-input addition in FPGAs. The FPCA is similar to an FPGA, but replaces LUTs with *m:n* counters instead. In a hybrid FPGA/FPCA, a compressor tree is mapped onto the FPCA, while all other operations are mapped onto the FPGA. As suggested by Kuon and Rose [19], the cost of routing data to and from the FPCA may limit its performance benefit. The new FPGA cell proposed here is much less ambitious, and exploits carry-chains rather than logical structures for effective local routing.

3.4 The Altera Stratix II ALM

In prevoius work on FPGA mapping [24, 25], GPCs were synthesized solely on LUTs, and carry-chains were not used, except for the final CPA. We tried several times, to develop mapping techniques that could exploit these carry-chains; however, we were unsuccessful. This frustration motivated us to design carry-chains that we could effectively exploit. We have augmented the Altera Stratix II ALM with these new carry-chains. This section reviews the Stratix II ALM in detail; the new carry chains are presented in Section 4.

This section describes the logic architecture of the Altera Stratix II FPGA (the same basic architecture was kept for the subsequent Stratix III as well). In particular, the existing carry-chains propagate through all 8 cells at a time; the carry chain in our proposed logic block, in contrast, propagates only through 2 at a time, thereby increasing flexibility and reducing delay.

Fig. 2 shows the *Adaptive Logic Module (ALM)* of the Stratix II/III in *shared arithmetic mode* [1, 3], in which the ALM is configured to perform ternary addition. It is important to note that the 6-input LUTs in the ALM are decomposed into smaller LUTs of 3- and 4-inputs. Only the smaller LUTs are shown in Fig. 2.

In Fig. 2, the ALM contains two pairs of 3-input LUTs; both LUTs in each pair share the same three inputs. Each pair of LUTs is configured to be a *3:2* compressor, producing the sum/carry bits *(S_r, C_r)* and *(S_{r+1}, C_{r+1})* respectively. All 3 inputs to the same pair of LUTs have the same rank, *r*; the sum bit also has rank *r*, and the carry-bit produced has rank *r+1*.

Two *full adders (FAs)*, connected in ripple carry fashion are shown in Fig. 2 as well. From the LUTs, the rank *r* sum bit is connected to an input of the rank *r* FA, while the rank *r+1* carry bit is connected to an input of the rank *r+1* FA. The carry-chain directly connects two bits (both of rank *r+2*) to the adjacent ALM, without going through the general routing network of the FPGA: the carry-bit of the rank *r+1 3:2* compressor, and the carry-bit of the rank *r+1* FA.

The sum outputs of the FAs are routed to the ALM outputs; however, the carry-outs are not. Each carry-out bit is connected to the carry-in of the next FA in the chain (except at the very end of the chain). When building a compressor tree, however, one needs the flexibility to connect the carry-out bit of a CSA to the appropriate CSA, not *just* the next CSA in a ripple-carry adder.

It would certainly be possible to allow the carry-out bit to be routed to one of the outputs of an ALM. The main penalties for doing so would be increase in the size of the multiplexer that selects which signal would be routed to the output (most of which are not shown in Fig. 2), increased fanout of the carry-out of each FA, and possibly extra configuration bits.

We have not, however, decided to implement such an ALM. Instead, we have made appropriate modifications to the carry-chain to allow it to allow the ALM to implement the functionality of a *6:2* compressor. The new carry-chain could be integrated in either an Altera or Xilinx FPGA; due to space limitations, our work describes the appropriate modifications to an Altera ALM.

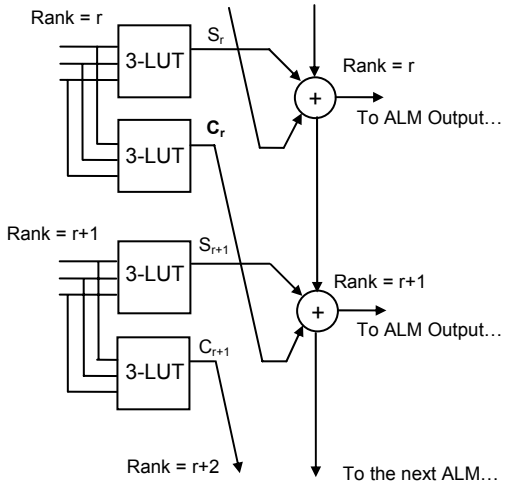

Figure 2. The Stratix II ALM in Shared Arithmetic Mode.

The carry-chain shown in Fig. 2 is used in the Altera ALMs. In the Xilinx *Configurable Logic Block (CLB)*, a different carry-chain is used [40, 41]. An extra *xor* gate and multiplexers are added to facilitate the propagation of *propagate* and *generate* signals, in the style of a *parallel-prefix adder*, rather than a ripple-carry adder.

4. NEW FPGA LOGIC CELL

In this section, we propose a new carry-chain that can be integrated into any FPGA logic cell. The new logic block allows the cell to be configured to implement the functionality of a *6:2* compressor. In Section 5, which follows, a new GPC-based mapping heuristic is proposed to exploit the new cell.

4.1 6:2 Compressor

In this section, we expand on the discussion of compressors in Section 2.5 and describe a *6:2* compressor in detail. The *6:2* compressor has 6-input bits: i_5, i_4, i_3, i_2, i_1, and i_0, and two carry-in bits, $c_{in,1}$ and $c_{in,0}$ all of rank-0 (the same rank that other inputs have); it produces two output bits out_1 and out_0 of ranks *1* and *0* respectively, and two carry-out bits, $c_{out,1}$ and $c_{out,0}$ of ranks *2* and *1* respectively. In the proposed logic cell, the two output bits out_1 and out_0 are routed to the two outputs of the logic cell, while $c_{out,1}$ and $c_{out,0}$ are routed to the next logic cell via the carry-chain.

Fig. 3(a) illustrates the basic I/O structure of a *6:2* compressor. A *6:2* compressor can be built from three *3:2* counters and a *2:2* counter, as illustrated in Fig. 3(b); this implementation of the *6:2* compressor is adopted for our logic cell. Fig. 3(c) illustrates the interconnect structure among four *6:2* compressors that reduces columns of ranks *j*, *j+1*, *j+2*, and *j+3* to two bits per column. The $c_{out,0}$ carry-out of the counter in column *j*, connects to the $c_{in,0}$ carry-in of the compressor in column *j+1*; meanwhile, $c_{out,1}$ connects to the $c_{in,1}$ carry-in of the compressor in column *j+2*.

In Fig. 3(c) the interconnect structure looks similar in principle to a ripple-carry chain: the c_{out} bits are always connected to c_{in} bits of subsequent counters. In Fig. 3(b), however, the c_{in} bits are inputs to the *3:2* counter, and go directly to the output. Therefore, the carry-chain goes through at most two *6:2* compressors before the "ripple effect" ends. This also holds true for the carry-chain in the proposed FPGA logic cell.

Initially, we considered the possibility of integrating support for a *6:3* counter (or 6-input GPC), rather than a *6:2* compressor, into a logic cell; we discarded this idea for three reasons. First, this would require adding a third output to the cell. Second, doing this would increase the number of connections from the cell to the general routing network and add extra complexity to the local routing network in each *Logic Array Block (LAB)*, which contains 8 ALMs. Third, a compressor tree built from *6:2* compressors would have fewer logic levels than a tree built from *6:3* counters. Due to the significant delays observed in FPGA routing networks, the *6:2* compressor seemed like a much better choice.

4.2 Logic Cell Design

Here, we describe the methodology we used to design the new logic cell. A *6:2* compressor must reduce 8 input bits of rank 0 (including the two carry-in bits) into 4 output bits of ranks 0, 1, 1, and 2 respectively (the latter two being the carry-out bits). Fig. 4 illustrates the process. As shown in Fig. 2, a total of 6 inputs can be connected to the two pairs of 3-input LUTs. Therefore, we assume that two bits (the carry-in bits) are provided via the carry-chain. The *6:2* compressor of Fig. 3(b) is constructed by a mixture of LUTs and adder circuits.

Like shared arithmetic mode, both pairs of 3-input LUTs are configured as *3:2* counters. In Fig. 4(a), this reduces six bits to four and yields two sum bits, S_1 and S_0, of rank 0, and two carry bits, C_1 and C_0, of rank 1. At this point, we must compress four bits of rank 0 and two bits of rank 1.

The second step, shown in Fig. 4(b), adds the bits S_1 and S_0 using a half adder (HA). This HA is new circuitry that is added to the ALM; however, an HA is just an *xor* gate and an *and* gate, so the overhead is minimal. The HA does not compress the two bits; it replaces two rank-0 bits (S_1 and S_0) with a rank-0 sum (labeled S_2) and a rank-1 carry (labeled C_2).

The last step, in Fig. 4(c), is to compress the two 3-bit columns in parallel with *3:2* compressors. The result of compressing the rank-0 bits (S_2, $c_{in,0}$, and $c_{in,1}$) are the output bits, out_0 and out_1, of the *6:2* compressor; the result of compressing the rank-1 bits (C_0, C_1, and C_2) are the carry-out bits, $c_{out,0}$ and $c_{out,1}$.

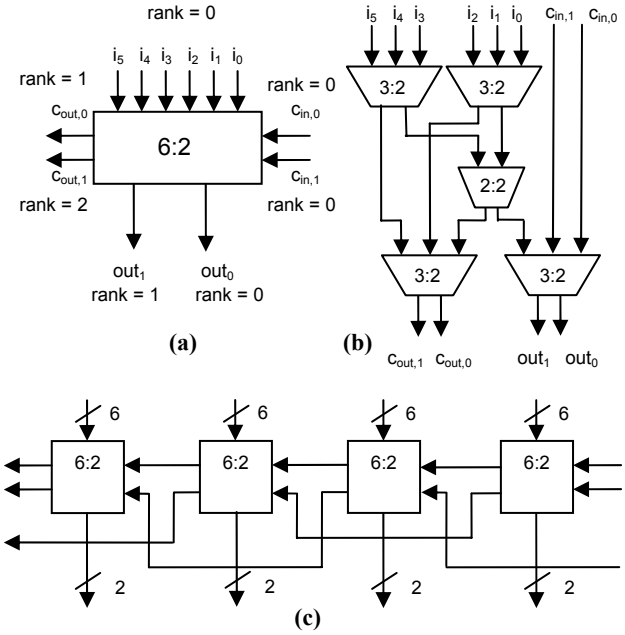

Figure 3.

I/O diagram of a 6:2 compressor (a); a 6:2 compressor constructed from 3:2 and 2:2 counters (b); a chain of 6:2 compressors performs column compression (c).

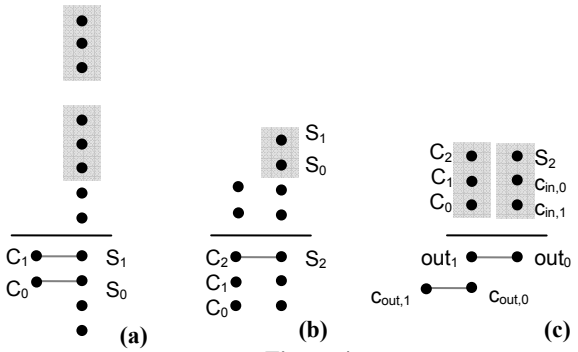

Figure 4.

Three steps to construct a 6:2 compressor for our logic cell.

4.3 Logic Cell Architecture

Here, we describe the architecture of the new logic cell. Two versions of the architecture are described. The first, in Fig. 5(a), is specific to the Altera Stratix II/III ALM because a single carry-chain implements both ternary addition and *6:2* compression. The second, in Fig. 5(b), employs two carry-chains, one for ternary addition and the second for *6:2* compression. The latter of the two carry-chains could be replicated and placed into any FPGA logic cell that offers the appropriate LUT structure.

The architecture in Fig. 5(a) augments the Altera ALM with four extra multiplexers so that the carry-chain can implement both desired functions. The extra multiplexers increase the delay through the carry-chain, which is a major drawback. This increases the delay of carry-propagate addition compared to Altera's ALM. We consider this extra delay to be unacceptable. We chose not to use this ALM in our experiments.

The architecture in Fig. 5(b) avoids the extra multiplexers by having two distinct carry-chains, the standard one for ternary addition, and a new one dedicated to the *6:2* compressor. The area of the second carry-chain is comparable to that of the four extra multiplexers in Fig. 5(a). Fig. 5(b) requires five direct connections to adjacent logic blocks; Fig. 5(a) requires just three.

The replicated carry-chain in Fig. 5(b) is portable, in the sense that it could simply be copied and integrated into any other logic cell that provides a similar LUT structure to the Stratix II.

The multiplexers on the right-hand-side of Fig. 5(b) do not change the overall delay of ternary addition or *6:2* compressors, because they do not affect the carry chain, which is the critical path through the circuit. The extra multiplexers are not on the path from the LUT outputs to the ALM output, so they do not affect delay when the carry chains are not used.

The dashed lines in Fig. 5 indicate the rank-2 carry-out of the preceding *6:2* compressor. As shown in Fig. 3(c), these wires skip the current compressor and connect to the carry-input of the next compressor. In Fig. 5(a), they must be connected to a multiplexer in the current compressor; in Fig. 5(b), they bypass the current compressor.

The inputs of each bit in the carry-chain at the top of the new cell are labeled *X*, *Y*, and *Z* in Fig. 5. At the bottom of the cell, the carry-out bits are labeled with the same letters, which illustrates the specific interconnection structure to the next cell in the chain. The specific input and output bits are also labeled, using the same variable names as in Section 4.1.

5. MAPPING HEURISTIC

Here, we describe a heuristic for synthesizing a compressor tree onto the FPGA logic cell shown in Fig. 5(b). This heuristic is an extension of our previous mapping heuristic that uses GPCs [24]. Each cell can either be configured as a 6-input GPC, using LUTs, or as a *6:2* compressor, using the proposed carry chain. The heuristic favors the *6:2* compressors, because they produce fewer output bits than *6*-input GPCs. This is illustrated in Fig. 6; Fig. 6(a) shows compression using *6:3* counters; there are 3 output bits per column; when *6:2* compressors are used, there are 2 output bits per column; the other output bits are propagated down the carry chain.

Fig. 6 illustrates the concept of a *compression ratio (CR)*; for any counter or compressor, CR is defined as follows:

$$CR = \frac{\#\,inputs}{\#\,outputs} \tag{4}$$

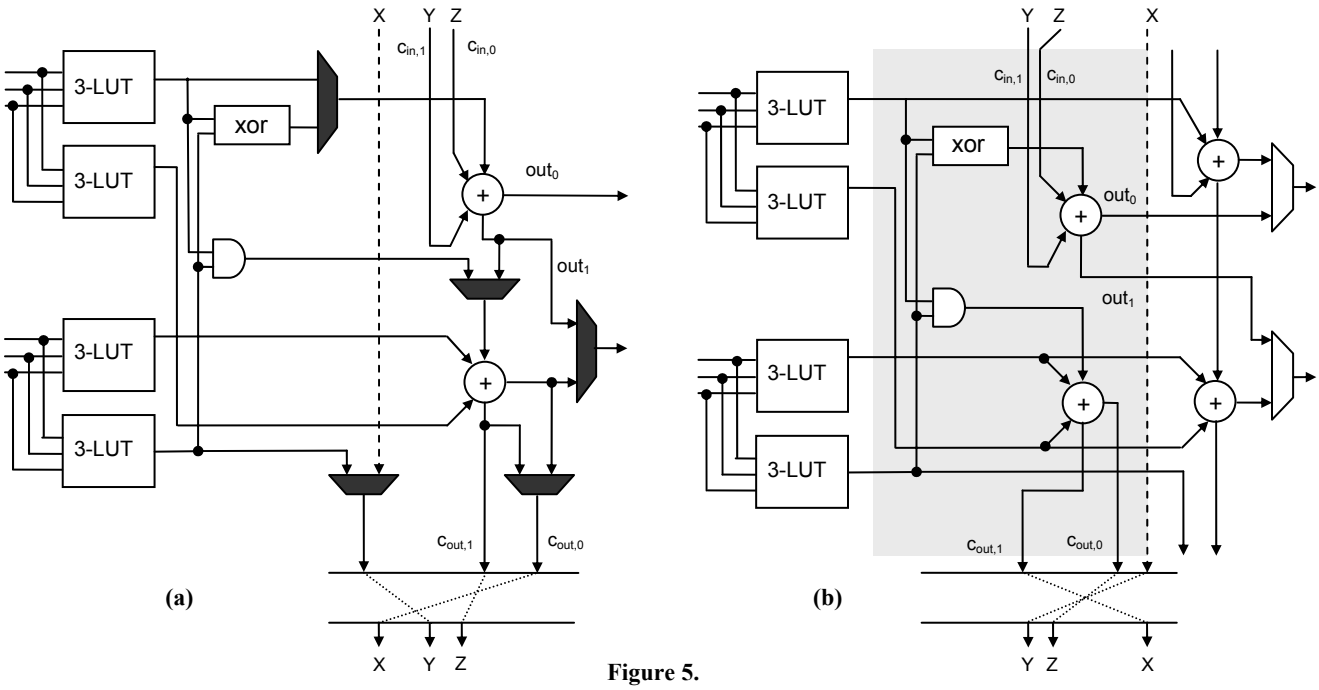

Figure 5.

Two architectures for the proposed FPGA cell. The dashed line is the rank-2 carry-out from the previous cell, which bypasses the current cell. X, Y, and Z, and the dotted lines, indicate the carry-chain connections between adjacent cells. In (a), the extra multiplexers that are required are shown in gray. In (b), the replicated carry-chain and additional extra logic are highlighted.

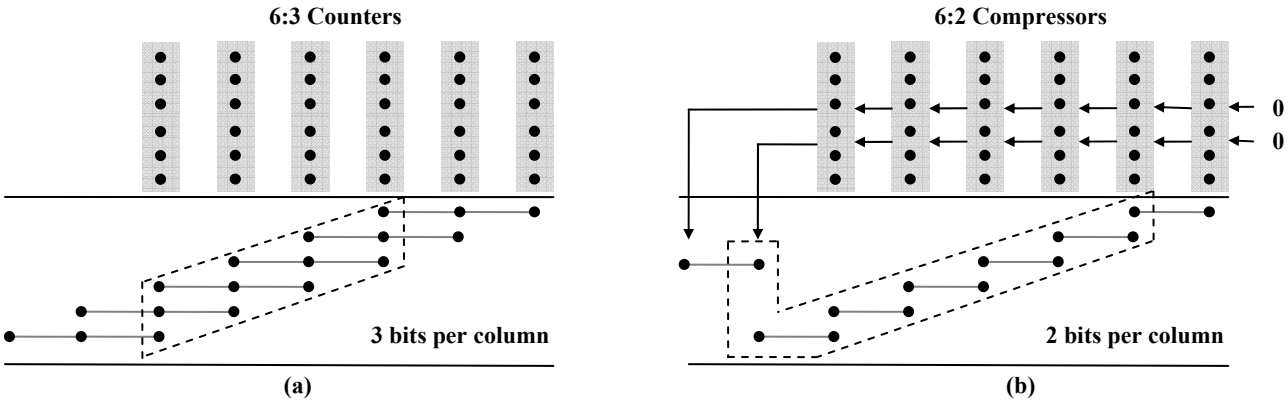

Figure 6.

Covering a set of columns with 6:3 counters yields 3 bits per column in the output (a); using 6:2 compressors reduces the number of bits per column to 2. Contiguous columns covered with 6:3 counters can be converted to 6:2 compressors.

For example, a 6-input, 3-output GPC has CR = 6/3 = 2; a 6:2 compressor, on the other hand, has CR = 6/2 = 3. In general, aggressive use of *6:2* counters in place of 6:3 counters and 6-input, 3-output GPCs, will reduce the number of bits in each column, as illustrated in Fig. 6; this, in turn, reduces the number of logic levels in a compressor tree.

Fig. 7 shows a flowchart of the heuristic for GPC mapping. The input is a set of columns of bits to be added and a library of GPCs (including single-column counters) that can be supported by the FPGA. The main steps are described as follows.

Step 1: First, the algorithm covers all of the columns using GPC mapping based on a greedy heuristic described previously [24]. A few small changes are made to the heuristic to favor the use of *6:2* compressors; however, the compressors themselves are not introduced until Step 2. As illustrated in Fig. 6, a contiguous set of *6:3* counters can be replaced with 6:2 compressors without changing the covering. Thus, the mapping heuristic should favor the use of single-column counters over GPCs when possible. This heurstic is optimistic in the sense that it employs single column counters in the hope that they can be converted to *6:2*

There are many different ways to cover the set of input bits with counters and compressors. The basic strategy of the original heuristic [24] is to try to use GPCs that maximize the compression ratio at each step. When multiple such GPCs are available, we choose a single-column counter when possible.

Step 2: To use *6:2* compressors, it is necessary to find contiguous columns whose bits are covered by single-column counters. The length of a carry-chain in the Stratix II/III is 8: the number of ALMs in a LAB. Thus, if we find a set of 8 (or fewer) contiguous columns, each of which has bits covered by single-column parallel counters, then those counters can be converted to compressors and mapped onto a LAB.

Steps 3-5: Step 3 maps the GPCs and *6:2* compressors found in steps 1 and 2 onto the LABs which contain the new cell. Step 4 determines the number of bits in each output column that results from the compression during this iteration of the heuristic.

If all columns have 3 or fewer bits, the remaining bits are summed using ternary adders in Step 5.

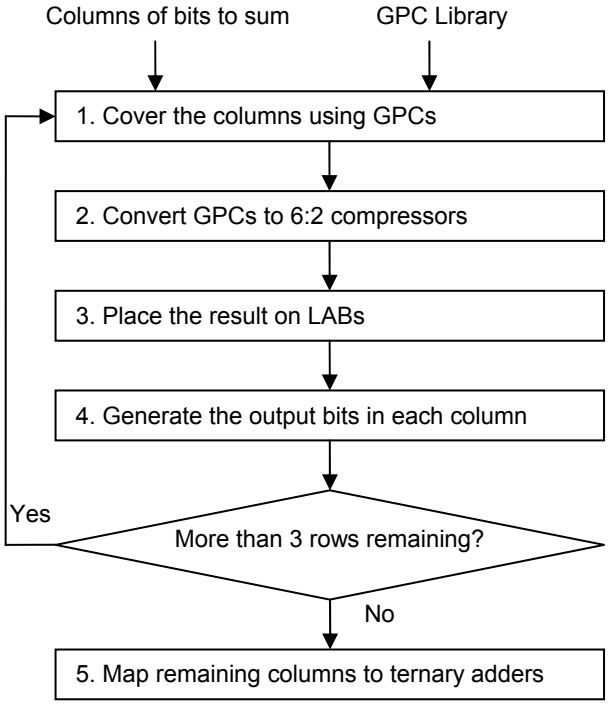

Figure 7.

GPC Mapping Algorithm

Otherwise, the process repeats for the remaining columns.

Although not shown in Fig. 7, the heuristic must also produce the connections between outputs of the previous layer of compression and the inputs of the current one, so that the proper routing between ALMs and LABs can be maintained.

6. EXPERIMENTAL RESULTS

We modeled an FPGA similar to the Altera Stratix II/III using *VPR* [4, 5]. Unlike the Stratix II/III, our LABs contained 4 instances of our logic cell (the version in Fig. 5(b)), rather than 8;

this was due to the complications involved with modeling carry chains in *VPR*, a time consuming and tedious process.

The connection between cells inside each LAB are local and do not go through the global routing network. The inputs and outputs of each LAB are connected to the global routing network. Each LAB has 5 inputs to its carry chain: 2 for ternary addition (like the Stratix II/III) and 3 for the *6:2* compressor. 2 of the 3 compressor inputs are connected to the first cell, and the remaining one is connected to the second cell.

We also modeled a LAB with ALMs based on the current cells used by Altera; the new cell, described above, was designed as a straightforward extension of this original cell. The LAB, ALM, and enhanced ALM cell (Fig. 5(b)) were modeled in VHDL and synthesized using *Synopsys Design Compiler* with 90nm TSMC standard cells. The designs were then placed and routed using *Cadence Silicon Encounter*. The delays, were extracted after placement and routing and then copied into the architecture configuration file that is used for modeling logic cells in VPR.

To model routing delays, VPR requires information such as the per-unit resistance and capacitance of wires. These quantities vary depending on both technology and metal layers. To the best of our knowledge, FPGA vendors do not state which metal layers are used for routing. In our experiments, we used the electrical characteristics of our foundry for their 90nm technology.

If viewed as a combinational circuit, an FPGA contains many false paths and false loops; these false paths and loops will never actually be active due to the way that the FPGA is programmed. Nonetheless, we must account for them during modeling and synthesis. For example, if our cell is configured to be a *6:2* compressor, then the combinational delay from LAB inputs to outputs through the ripple-carry chain for ternary addition should be ignored by VPR, and vice-versa. To coax VPR into ignoring the false paths, we use one output for each case (ternary adder, *6:2* compressor, and LUT). A multiplexer is inserted to select between the different outputs. The delay of the multiplexers is included in our VPR model for each output.

We selected a set of benchmarkss circuits from DSP and video processing applications. When applicable, we applied the transformations described by Verma and Ienne [36] to expose multi-operand addition operations. We then synthesized each multi-operation addition 4 times:

3-ADD: Synthesis using ternary adder trees using the standard Altera LAB (with 4 ALMs per LAB).

GPC: Synthesis using compressor trees using our GPC mapping heuristic [24] on the standard Altera LAB.

6:2: Synthesis using compressor trees on the modified ALM (Fig. 5(b)) using Fig. 7, with *only 6:2* compressors; the GPC library was limited to single column counters.

6:2 + GPC: Synthesis using compressor trees on the modified ALM (Fig. 5(b)) using Fig. 7 with the complete GPC library.

Figs. 8-10 present the results of our experiments. Fig. 8 shows the speedup obtained using each heuristic; Fig. 9 shows the routing delay for each synthesis method; Fig. 10 shows the area.

Fig. 8 shows that the 6:2 + GPC yielded the minimal delay in all cases. Compared to 3-ADD, the speedup obtained by GPC, 6:2, and 6:2 + GPC were 1.15×, 1.13×, and 1.41× respectively. It is interesting to note that GPC and 6:2 had similar delays, on average, but combining the two significantly reduced the overall delay. Thus, it is clear that during mapping, separate situations arise that favor GPCs and *6:2* compressors.

Since *6:2* compressors cannot be effectively synthesized and interconnected without carry-chains, we think that this justifies the inclusion of the extra carry-chain in the ALM. The amount of extra logic required for the separate carry chain, shown in Fig. 5(b), is small, compared to the cost of the complete ALM (only a fraction of which is shown in Fig. 2), especially when the area of the SRAM cells that hold the configuration bits is considered.

Fig. 9 shows that GPC synthesis has the greatest routing delay among the four approaches. The reason for this is that GPC synthesis tends to produce more outputs from the ALMs than either 3-ADD trees or synthesis with *6:2* compressors. In GPC synthesis, each GPC output (three outputs per 6-input GPC) is an ALM output, and must be routed to an ALM input in the next level of the tree. In the case of 3-ADD, only the sum bits of each adder are ALM outputs; the carry output is propagated along the carry-chain to the next ALM in the LAB. Likewise, when a *6:2* compressor is used, the two output bits are ALM outputs; the two carry bits produced by the compressor are propagated into the next cell in the LAB by the carry-chain. Thus, 3-ADD, 6:2, and 6:2 + GPC tend to produce fewer ALM outputs than GPC, and there is less routing delay as a result.

Logic delay is reduced by reducing the number of logic levels in the tree—this is the advantage of ternary adders over binary adders, as well as compressor trees over adder trees. Likewise, the routing delay is reduced due to the use of local carry chains replacing ALM outputs.

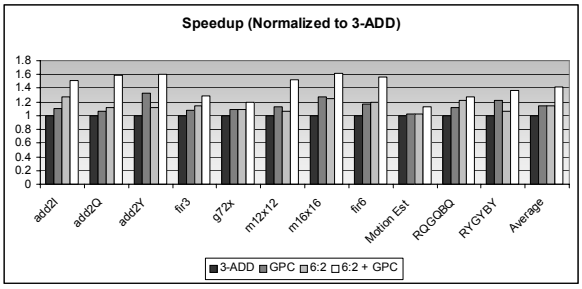

Figure 8.

Speedup observed for different compressor tree synthesis methods (normalized to 3-ADD).

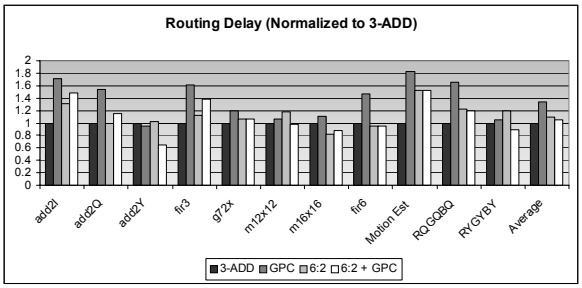

Figure 9.

Routing delay for different compressor tree synthesis methods (normalized to 3-ADD).

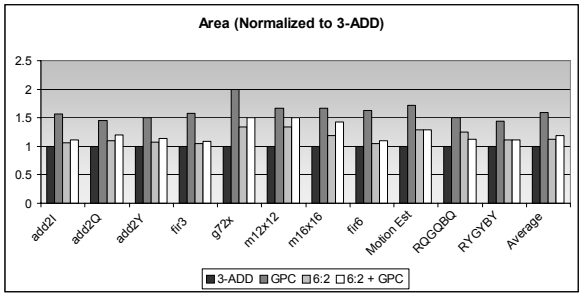

Figure 10.

Area (normalized to 3-ADD) of adder and compressor trees synthesized in four different ways.

From Fig. 10, we can see that 3-ADD requires less area than mapping with GPCs and/or *6:2* compressors. GPC requires significantly more area than the other mapping techniques. The area required for 6:2 and 6:2 + GPC are comparable. This can be attributed to the fact that each 6:2 compressor requires one cell, while each GPC requires two cells. Furthermore, the area overhead of 6:2 and 6:2 + GPC, compared to 3-ADD is minimal, compared to the overhead of GPC.

Altogether, 6:2 + GPC provides the best delay compared to the other mapping techniques. The superiority of GPC over 3-ADD has already been established [24, 25]. The delay of 6:2 is actually greater than that of GPC; however, the best overall result combines both of these techniques. In terms of area, GPC is significantly worse than the others, but the respective area over of 6:2 and 6:2 + GPC is quite small compared to 3-ADD.

We did not compare these results to our ILP formulation for GPC mapping [25]. First and foremost, the time required to solve an ILP optimally is several orders of magnitude larger than the time required to produce heuristic solutions, and this is just for GPC mapping. If we included the possibility of using *6:2* compressors in addition, the search space would become even larger and the ILP would become even slower. By adding some constraints and sacrificing global optimality, we might be able to get the ILP to converge much more rapidly; however, doing so is beyond the scope of this paper, and is left open for future work.

7. CONCLUSION

We have developed a new type of FPGA logic cell that allows it to be configured as a 6:2 compressor without sacrificing its current functionality as a 6-input LUT or a ternary adder without changing the input/output structure of the cell. Transformations proposed by Verma and Ienne [37] have shown that multi-operand addition can be exposed for a wide variety of real-world applications, buttressing our decision to accelerate this particular operation with custom hardware. The new logic cell does not incur much additional area overhead compared to the ALM of the Stratix II/III or CLB of the Virtex-4/5. Our experiments show that the best overall delay can be achieved by augmenting our prior work on GPC mapping to use *6:2 compressors* as well.

In the future, we intend to develop better mapping techniques for *6:2* compressors and GPCs. We may extend the ILP formulation of Parandeh-Afshar et al. [25] to account for *6:2* compressors, however, we are concerned about the extra runtime due to the enlarged search space. We do intend to develop more aggressive heuristics that produce better results than the greedy heuristic of Parandeh-Afshar et al. [24], but run in polynomial time, unlike their ILP formulation [25]. For now, these improvements are left open for future work.

REFERENCES

[1] Altera Corporation, *Stratix II Device Handbook, vol. 1* and *2*, available online: http://www.altera.com/

[2] Altera Corporation, *Stratix II vs. Virtex-4 Performance Comparison*, available online: http://www.altera.com/

[3] Altera Corporation, *Stratix III Device Handbook, vol. 1* and *2*, available online: http://www.altera.com/

[4] Betz, V., and Rose, J. VPR: a new packing, placement and routing tool for FPGA research, *7th Int. Workshop on Field-Prog. Logic and Applications (FPL '97)* (London, UK, September 1-3, 1997) 213-222.

[5] Betz, V., Rose, J., and Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAs*, Springer, 1999.

[6] Brisk, P., Verma, A. K., Ienne, P., and Parandeh-Afshar, H. Enhancing FPGA performance for arithmetic circuit, *Design Automation Conf. (DAC '07)* (San Diego, CA, USA, June 4-8, 2007) 334-337.

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

[8] Cherepacha, D., and Lewis, D. DP-FPGA: an FPGA architecture optimized for datapaths. *VLSI Design, vol. 4, no. 4*, 1996, 329-343.

[9] Cong, J., and Huang, H. Technology mapping and architecture evaluation for k/m-macrocell-based FPGAs. *ACM Trans. Design Automation of Electronic Systems, vol. 10, no. 1,* January, 2005, 3-23.

[10] Dadda, L., Some schemes for parallel multipliers, *Alta Frequenza, vol. 34*, May, 1965, 349-356.

[11] Fadavi-Ardekani, J. M × N Booth encoded multiplier generator using optimized Wallace trees. *IEEE Trans. VLSI Systems, vol. 1., no. 2*, June, 1993, 120-125.

[12] Frederick, M. T., and Somani, A. K. Multi-bit carry chains for high-performance reconfigurable fabrics. *Int. Conf. Field Prog. Logic and Applications (FPL '06)* (Madrid, Spain, August 28-30, 2006) 1-6.

[13] Grover, R. S., Shang, W., and Li, Q. A faster distributed arithmetic architecture for FPGAs. *Int. Symp. FPGAs (FPGA '02)* (Monterey, CA, USA, February 24-26, 2002) 31-39.

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

[15] Hu, Y., and He, L. Private communication. June 8, 2007.

[16] Hu, Y., Das, S., and He, L. Design, synthesis, and evaluation of heterogeneous FPGA with mixed LUTs and macro-gates. *Int. Workshop on Logic and Synthesis (IWLS '07)* (San Diego, CA, USA, May 30- June 1, 2007) – extended version to appear at ICCAD, November, 2007.

[17] Kastner, R., Kaplan, A., Ogrenci-Memik, S., and Bozorgzadeh, E. Instruction generation for hybrid reconfigurable systems. *ACM Trans. Design Automation of Electronic Systems, vol. 7, no. 4,* October, 2002, 605-627.

[18] Kaviani, A., Vranseic, D., and Brown, S. Computational field programmable architecture, *IEEE Custom Integrated Circuits, Conf. (CICC '98)* (Santa Clara, CA, USA, May 11-14, 1998) 261-264.

[19] Kuon, I., and Rose, J. Measuring the gap between FPGAs and ASICs. *IEEE Trans. Computer-Aided Design, vol. 26, no. 2*, February, 2007, 203-215.

[20] Kwon, O., Nowka, K., and Swartzlander Jr., E. E., A 16-bit by 16-bit MAC design using fast 5:3 compressor cells, *Journal of VLSI Signal Procsesing*, Vol. 31, No. 2, pp. 77-89, June, 2002.

[21] Leijten-Nowak, K., and van Meerbergen, J. L., An FPGA architecture with enhanced datapath functionality, *Int. Symp. FPGAs (FPGA '03)* (Monterey, CA, USA, February 23-25, 2003) 195-204.

[22] Mirzaei, S., Hosangadi, A., and Kastner, R. High speed FIR filter implementation using add and shift method, *Int. Conf. Computer Design (ICCD '06)* (San Jose, CA, USA, October 1-4, 2006).

[23] Mora Mora, H., Mora Pascual, J., Sánchez Romero, J. L., and Pujol López, F. Partial production reduction based on look-up tables, *Int. Conf. VLSI Design (VLSI Design '06)* (Hyderabad, India, January 3-7, 2006) 399-404.

[24] Parandeh-Afshar, H., Brisk, P., and Ienne, P. Efficient Synthesis of Compressor Trees on FPGAs. *Asia and South Pacific Design Automation Conference (ASPDAC '08)* (Seoul, Korea, January 21-24, 2008).

[25] Parandeh-Afshar, H., Brisk, P., and Ienne, P. Improving Synthesis of Compressor Trees on FPGAs via Integer Linear Programming, to appear: *Design, Automation and Test in Europe Conference and Exhibition (DATE '08)* (Munich, Germany, March 10-14, 2008).

[26] Poldre, J., Tammemae, K. Reconfigurable multiplier for Virtex FPGA family, *Int. Workshop on Field-Programmable Logic and Applications (FPL '99)* (Glasgow, UK, August 30 – September 1, 1999) 359-364.

[27] Santoro, M., and Horowitz, M. A pipelined 64x64b iterative array multiplier, *IEEE Int. Solid-State Circuits Conf. (ISSCC '88)* (February 17-19, 1988) 36-37, 290.

[28] Song, P., and De Micheli, G. Circuit and architecture trade-offs for high speed multiplication, *IEEE Journal of Solid-State Circuits, vol. 26, no. 9*, September, 1991, 1184-1198.

[29] Sriram, S., Brown, K., Defosseux, R., Moerman, F., Paviot, O., Sundararajan, V., and Gatherer, A. A 64 channel programmable receiver chip for 3G wireless infrastructure, *IEEE Custom Integrated Circuits Conf. (CICC '05)* (San Jose, CA, USA, September 18-21, 2005) 59-62.

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

[31] Stelling, P. F., and Oklobdzija, V. J., Design strategies for optimal hybrid final adders in a parallel multiplier, *Journal of VLSI Signal Processing, vol. 14, no. 3*, December, 1996, 321-331.

[32] Stenzel, W. J., Kubitz, W. J., and Garcia, G. H. A compact high-speed parallel multiplication scheme, *IEEE Trans. Computers, vol. C-26, no. 10,* October, 1977 948-957.

[33] Swartzlander Jr., E. E. Parallel counters. *IEEE Trans. Computers*, *vol. C-22, no. 11*, November, 1973, 1021-1024.

[34] Um, J., and Kim, T. Layout-aware synthesis of arithmetic circuits, *Design Automation Conf. (DAC '02)* (New Orleans, LA, USA, June 10-14, 2002) 207-212.

[35] Verma, A. K., and Ienne, P. Automatic synthesis of compressor trees: reevaluating large counters, *Design Automation and Test in Europe (DATE '07)* (Nice, France, April 16-20, 2007) 443-448.

[36] Verma, A. K., and Ienne, P. Improved use of the carry-save representation for the synthesis of complex arithmetic circuits, *Int. Conf. Computer-Aided Design (ICCAD '04)* (San Jose, CA, USA, November 7-11, 2004) 791-798.

[37] Verma, A. K., and Ienne, P. Improving XOR-dominated circuits by exploiting dependencies between operands, *Asia-Pacific Design Automation Conf. (ASP-DAC '07)* (Yokohama, Japan, January 23-26, 2007) 601-608.

[38] Wallace, C. S. A suggestion for a fast multiplier, *IEEE Trans. Elec. Computers, vol. 13*, February, 1964, 14-17.

[39] Weinberger, A. 4:2 carry-save adder module, *IBM Technical Disclosure Bulletin, vol. 23*, Jan. 1981.

[40] Xilinx Corporation, *Virtex-4 User Guide*, available online: http://www.xilinx.com/

[41] Xilinx Corporation, *Virtex-5 User Guide*, available online: http://www.xilinx.com/

[42] Zuchowski, P. S., Reynolds, C. B., Grupp, R. J., Davis, S. G., Cremen, B., and Troxel, B. A hybrid ASIC and FPGA architecture, *Int. Conf. Computer-Aided Design (ICCAD '02)* (San Jose, CA, USA, November 10-14, 2002) 187-194.