Efficient High Speed Compression Trees on Xilinx FPGAs

Martin Kumm
University of Kassel
Kassel
kumm@uni-kassel.de

Peter Zipf
University of Kassel
Kassel
zipf@uni-kassel.de

Abstract

Compressor trees are efficient circuits to realize multi-operand addition with fast carry-save arithmetic. They can be found in various arithmetic applications like multiplication, squaring and the evaluation of polynomials with application to function approximation. Finding good elementary compressing elements on FPGAs is a non-trivial task as an efficient mapping to look-up tables and carry-chain logic has to be found. It was shown recently that common ternary adders on modern FPGAs outperform previous compressors in terms of area-efficiency, i. e., the number of compressed bits per logic unit. While Altera FPGAs allow a fast and compact implementation of ternary adders we observed that ternary adders on Xilinx FPGAs only achieve about half of the speed of a two-input adder. This work proposes novel compressing elements including different generalized parallel counters and a 4:2 compressor which is based on a modified ternary adder. They provide a better area-efficiency and/or a lower delay than previously proposed compressing elements.

1. Introduction

Various fundamental arithmetic operations involve the addition of multiple numbers. These include multipliers (real/complex), polynomials (incl. squarer) and linear transforms. On FPGAs, efficient implementations for soft-core multipliers [GAKC09, PAI11], large size multipliers using multiple DSP blocks and compressor trees [GAKCL12, BdDI+13], complex multipliers and multivariate polynomials for sine approximation using its Taylor series [BdDI+13] are reported in the literature. The implementation of multi-operand adders on application specific integrated circuits (ASICs) is most of the times best realized by using compression trees. Compared to binary adder trees using common carry propagate adders (CPAs), compression trees typically provide a much higher speed due to the avoidance of long carry chains with nearly the same area requirements. For FPGAs, the situation looks much different. Here, on the one hand, specific carry logic provides the implementation of fast ripple carry adders (RCAs) and, on the other hand, the delay of a simple wire in the routing fabric is much higher. However, only a fraction of a look-up table (LUT) is used when implementing two-input RCAs. Therefore, the main idea for good basic compression elements on FPGAs is to find a way to use the LUTs for additional computations.
A 4:2 compressor for Xilinx' four-input LUT FPGAs with carry logic (Virtex 2/4 and Spartan 2/3) was proposed by Ortiz et al. [OQH+09]. They mapped two cascaded full adders into a single slice

to compress 4 inputs (plus carry-in) to two outputs (plus carry-out). With the advent of modern FPGAs providing up to six-input LUTs, the possibilities for mapping additional compression elements in the LUT are much larger. Fundamental work on this was done by Parandeh-Afshar et al. [PANBI11, PA12] which also offers a good introduction to state-of-the-art compression trees on FPGAs. They used LUTs but also short carry chain paths (2 to 3 full adders in series) to construct generalized parallel counters (GPCs, see Section 2.1 for its definition) as efficient compression elements. They can be used as library elements for the automatic synthesis of compression trees. The problem here is to find a good assignment of the compression elements to the inputs which is always a tradeoff between area and delay. The heuristic algorithm in [PANBI11] focuses on a good utilization of the used compression elements, i. e., the compression element which inputs fit best to the remaining compressor tree inputs is chosen.
An alternative mapping of efficient compressor trees on FPGAs was proposed recently by Hormigo et al. They arranged common carry-save adders (CSA) and ternary adders in a diagonal way such that short paths of the FPGA carry-chain are utilized [HVZ13]. While the previous discussed compressing elements are also very effective for pipelining, the diagonal arrangement may lead to a lot of additional pipeline registers to balance the pipeline.
Another compression tree optimization heuristic was proposed recently by de Dinechin et al. [BdDI+13]. They proposed to represent the problem (i. e., all input bits that have to be added) as a data structure which they call a bit heap. It is a data representation of dot diagrams [EL04] which is a great abstraction of compression trees and is also very beneficial from a software engineering point of view. Their algorithms are available in the open-source arithmetic core generation framework FloPoCo [dDP12, flo13]. They analyzed compression elements for their efficiency and use the most efficient ones in their heuristic instead of forcing a good utilization. Another conclusion from their efficiency analysis was that the ternary adder is the best compression element in terms of efficiency. However, as it was shown by our group, ternary adders are slow compared to common two-input adders on Xilinx FPGAs [KHW+13]. While the performance penalty for Altera FPGAs is 5 to 10% up to 32 bit, it is close to 50% for Xilinx FPGAs which makes the ternary adder unattractive for high speed applications.
The main contribution of this work is the introduction of four new compression elements for Xilinx FPGAs with improved efficiency and less delay. The most efficient and flexible one with highest speed is a 4:2 compressor which is based on a modified ternary adder. It avoids the LUT and routing delays that occur in the ternary adder and provides the same efficiency. Furthermore, it can be pipelined without overhead and can be well integrated into pipelined compressor trees. It was shown by our group that pipelining plays a crucial role in adder-based arithmetic for multiple constant multiplication (MCM) which is similar to the situation in compressor trees [KZ11]. Average speedups of 300% were achieved by spending 18% more slice resources. However, the optimization of pipelined compressor trees was not treated in the literature so far. We show exemplarily that using pipelined compression elements results in fast and compact compressor trees.

2. Previous Work on Compressing Elements

2.1. Classification

The inputs of an n input multi-operand adder where each input i has a word size of b_i bits can be represented as a $n \times b$ bit-array (where b is the maximum of the input word sizes b_i). If a_{ij} denotes

the input bit of row i (input operand) and column j (bit weight) of the bit-array, the corresponding output value v of the multi-operand addition is [EL04]:

$$v = \sum_{i=0}^{n-1} \sum_{j=0}^{b_j-1} a_{ij} 2^j \tag{1}$$

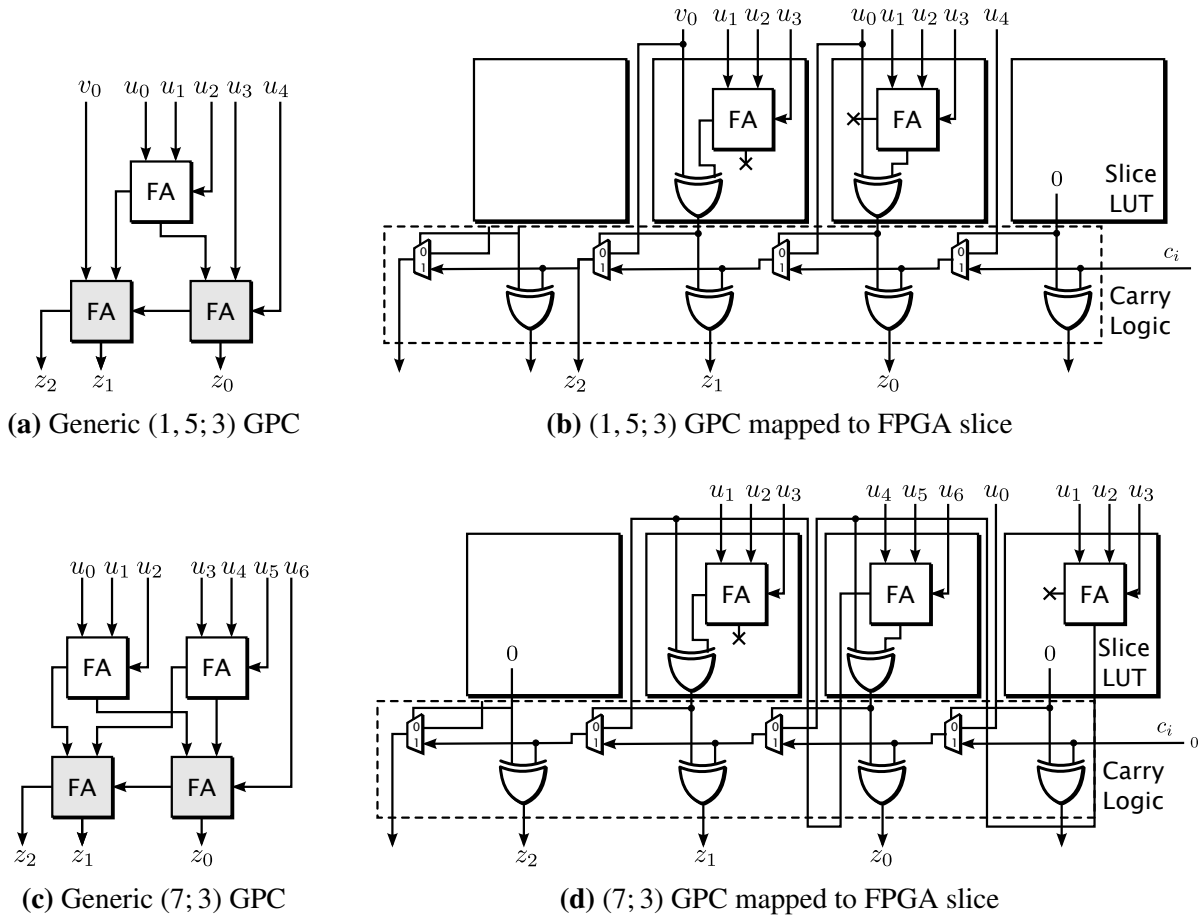
The basic operation in a compressor tree, which we call compressing element, takes a bit-array and produces a smaller bit-array. Compressing elements can be classified by their reduction of rows or columns (or both). A *counter* (or parallel counter) is a circuit that counts the number of input bits of a column which are one. Half-adder and full-adders are examples of 2:2 and 3:2 parallel counters, respectively. Here, $p : q$ denotes number of inputs (p) and outputs (q). *Generalized parallel counters* (GPCs) [PANBI11] which are also called multicolumn counters [EL04] allow that input bits may have different weights (and are thus in different columns). A GPC is commonly denoted as tuple $(p_{k-1}, p_{k-2}, \ldots, p_0; q)$, where p_j represents the number of input bits of weight 2^j and q is the number of output bits. A (3,5;4) GPC, for example, computes the sum of 3 input bits of weight two plus 5 input bits of weight one. The result is a number in the range $0 \ldots 2 \cdot 3 + 5 = 11$ and is represented by a single bit vector with 4 bit. A $p : q$ *compressor* is a circuit that reduces p input bits of the same weight to q output bits with possibly different weights [PA12]. They also provide explicit carry-in and carry-out signals which are *not* counted as input or output bits. To avoid long carry chains, the carry-out typically does not depend on the carry-in. Cascading b $p : q$ compressors reduces p bit vectors of word size b to q bit vectors in a redundant representation. Note that the notation of a compressor is not consistent in the literature as, e. g., full-adders or carry-save adders are sometimes also referred to as 3:2 compressors.

2.2. Generalized Parallel Counters on FPGAs

A set of GPC mappings to FPGAs which utilize LUTs and fast carry chains was proposed by Parandeh-Afshar [PANBI11]. This set is used as basic elements for heuristically synthesizing compressor trees. An example of a $(1, 5; 3)$ GPC is shown in Figure 1a. The main idea is to utilize the FPGA carry chains (shaded boxes in Figure 1a) and to map additional full-adders to the LUTs. The FPGA mapping [PANBI11] for Xilinx FPGAs which provide six-input LUTs that can be configured as two five-input LUTs with shared inputs, namely the Virtex 5-7, Spartan 6, Kintex 7 and Artix 7 families, is shown in Figure 1b. In addition to the carry-chain, one XOR gate has to be realized in the LUT to build a two-input adder. To improve the efficiency, additional full adders are realized in the same LUT. Another example of a $(7; 3)$ GPC (which is also a 7:3 counter) is shown in Figure 1c and its FPGA mapping is shown in Figure 1d. Some other simple LUT based GPCs were also proposed in [BdDI+13]. Here, a $3 : 2$ counter was mapped to a single five-input LUT with two outputs and a $6 : 3$ counter as well as a $(1, 5; 3)$ GPC was mapped to three six-input LUTs.

2.3. Ternary Adders on FPGAs

A ternary adder computes the sum of three input bit vectors $s = x + y + z$. Instead of using two stages of ripple-carry adders, a carry-save adder is used in the first stage which compresses the three input vectors to two vectors which are then merged in a second stage using a common RCA,



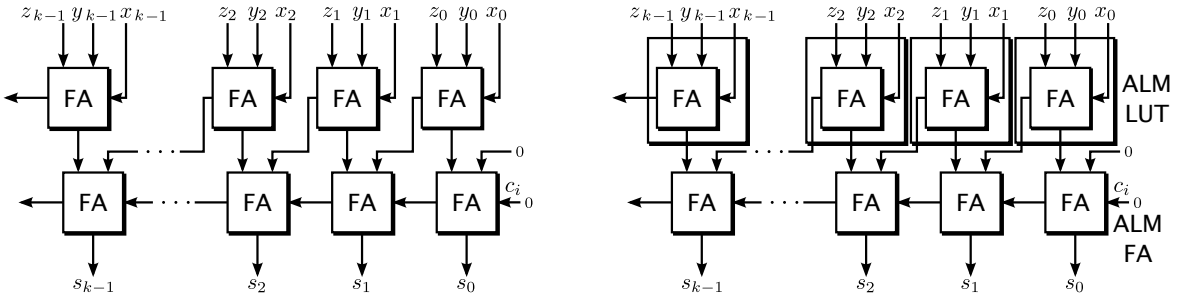
(a) Generic $(1, 5; 3)$ GPC **(b)** $(1, 5; 3)$ GPC mapped to FPGA slice

(c) Generic $(7; 3)$ GPC **(d)** $(7; 3)$ GPC mapped to FPGA slice

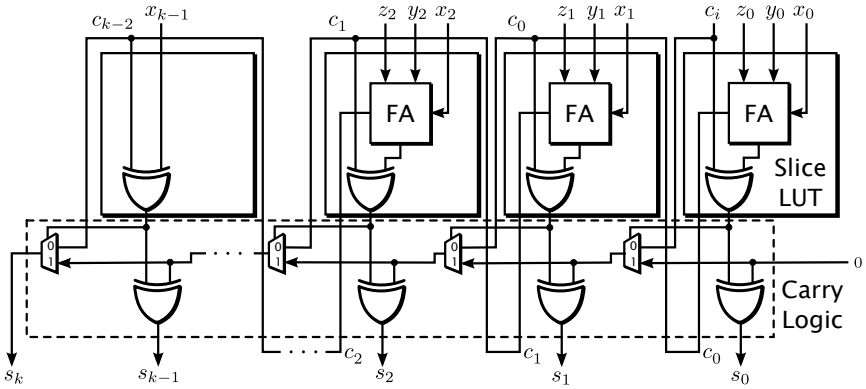
Figure 1: Compression elements from literature [PANBI11]

which is shown in Figure 2a. In principle, on modern FPGAs this uses the same number of full-adders but allows the mapping of the first stage of full-adders to the same LUT used for the RCA of the second stage. Hence, the ternary adder consumes the same resources as the two-input adder. Altera Arria I,II,V and Stratix II-V FPGAs directly support ternary adders. Their adaptive logic modules (ALMs) support a *shared arithmetic mode* in which a LUT output can be directly connected to the full-adder input of the next higher bit [BLSY09]. This allows a direct mapping of the full-adders of the first stage to the LUTs and the second stage of full-adders can be realized by the ALM full-adders as shown in Figure 2b.

Xilinx FPGAs do not provide something comparable to the shared arithmetic mode. Thus, the routing fabric of the FPGA has to be used to connect the carry-out of the first full-adders to the full-adder inputs of the next stage. In addition to that, parts of the full-adders in the second stage have to be realized in the LUT (an XOR gate) which requires an additional input. Thus, four inputs and two outputs of the LUT are needed. This mapping is possible on all modern Xilinx FPGAs and is shown in Figure 2c [SP06]. Note that an additional carry input (c_i) is possible in the first stage while not all inputs can be used in the last stage as the corresponding slice output for an additional full-adder is occupied with the carry-chain output.



(a) Generic architecture **(b)** Altera Stratix ALM mapping [BLSY09]



(c) Xilinx slice mapping [SP06]

Figure 2: Realization of ternary adders

2.4. Evaluation of Compressing Elements

Different performance metrics were proposed to evaluate compressing elements [BdDI+13]. Compressing elements can be characterized by their number of input bits b_i and number of output bits b_o. From these, the most important performance measure is the number of reduced bits $\delta = b_\mathrm{i} - b_\mathrm{o}$ in relation to the hardware resources, which is called the efficiency [BdDI+13]:

$$E = \frac{\delta}{k} \tag{2}$$

Here, k stands for the number of basic logic elements (BLEs) which is defined as the smallest common logic element in the FPGA. For the Xilinx FPGAs considered, it is represented as a quarter of a slice consisting of a six-input LUT which can be configured as two five-input LUTs with shared inputs, the carry-chain logic and two flip-flops. Hence, a common two-input adder consumes one BLE per bit.

Another important measure is the delay of the compressing element. It consists of LUT delays τ_L, routing delays τ_R and carry-chain delays τ_CC. A detailed timing analysis with the tool `trace` revealed that for a Virtex 6 the LUT delay is $\tau_\mathrm{L} \approx 0.3$ ns, a local routing takes about $\tau_\mathrm{R} \approx 0.3 \ldots 0.5$ ns and carry propagation of a carry chain with 4 bits takes 0.06 ns, leading to a single carry delay of $\tau_\mathrm{CC} \approx 0.015$ ns. Thus, the carry-propagation is much less than a LUT delay or a local routing delay which are in the same order of magnitude ($\tau_\mathrm{L} \approx \tau_\mathrm{R} \approx \tau$). This is close to a previous approximation of $\tau_\mathrm{L} + \tau_\mathrm{R} \approx 30\tau_\mathrm{CC}$ [BdDI+13].

An evaluation of compression elements is listed in Table 1. The upper half of the table was published earlier [BdDI+13] and is reproduced here for the sake of comparison. It contains previous compression elements including two-input adders and ternary adders. It can be observed that the best compressing element in terms of efficiency is the ternary adder which approaches $E = 2$ for a large number of BLEs (k) while having a delay between about 3τ for low word sizes ($k < 10$ bit) and about 6τ for large word sizes of $k = 64$ bit. Note that none of the compressing elements except the ternary adder has a better efficiency than $E = 1$.

Table 1: Comparison of different compression elements

GPC / Compressor	b_i	b_o	$\delta = b_\mathrm{i} - b_\mathrm{o}$	BLEs (k)	Efficiency ($E = \delta/k$)	delay
Naive LUT based GPCs from [BdDI+13][1]:						
(3;2) GPC	3	2	1	1	1	$\tau_\mathrm{L} \approx \tau$
(6;3) GPC	6	3	3	3	1	$\tau_\mathrm{L} \approx \tau$
(1,5;3) GPC	6	3	3	3	1	$\tau_\mathrm{L} \approx \tau$
LUT and carry-chain GPCs from [PANBI11]:						
(6;3) GPC	6	3	3	4	0.75	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
(1,5;3) GPC	6	3	3	3	1	$\tau_\mathrm{L} + 3\tau_\mathrm{CC} \approx \tau$
(2,3;3) GPC	5	3	2	3	0.67	$\tau_\mathrm{L} + 3\tau_\mathrm{CC} \approx \tau$
(7;3) GPC	7	3	4	4	1	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
(1,6;4) GPC	7	4	3	4	0.75	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
(3,5;4) GPC	8	4	4	4	1	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
(4,4;4) GPC	8	4	4	4	1	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
(5,3;4) GPC	8	4	4	4	1	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
(6,2;4) GPC	8	4	4	4	1	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
Adder with k BLE:						
2-input adder	$2k + 1$	$k + 1$	k	k	1	$\tau_\mathrm{L} + k\tau_\mathrm{CC}$
3-input adder	$3k - 1$	$k + 1$	$2k - 1$	k	$2 - \frac{2}{k}$	$2\tau_\mathrm{L} + \tau_\mathrm{R} + k\tau_\mathrm{CC} \approx 3\tau + k\tau_\mathrm{CC}$
Proposed improved mappings for GPCs, originally introduced in [PANBI11]:						
(6;3) GPC	6	3	3	3	1	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 3\tau_\mathrm{CC} \approx 3\tau$
(1,5;3) GPC	6	3	3	2	1.5	$\tau_\mathrm{L} + 2\tau_\mathrm{CC} \approx \tau$
(2,3;3) GPC	5	3	2	2	1	$\tau_\mathrm{L} + 2\tau_\mathrm{CC} \approx \tau$
(7;3) GPC	7	3	4	3	1.33	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 3\tau_\mathrm{CC} \approx 3\tau$
(5,3;4) GPC	8	4	4	3	1.33	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 3\tau_\mathrm{CC} \approx 3\tau$
(6,2;4) GPC	8	4	4	3	1.33	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 3\tau_\mathrm{CC} \approx 3\tau$
Proposed GPCs:						
(5,0,6;5) GPC	11	5	6	4	1.5	$\tau_\mathrm{L} + 4\tau_\mathrm{CC} \approx \tau$
(1,4,1,5;5) GPC	11	5	6	4	1.5	$\tau_\mathrm{L} + 4\tau_\mathrm{CC} \approx \tau$
(1,4,0,6;5) GPC	11	5	6	4	1.5	$\tau_\mathrm{L} + 4\tau_\mathrm{CC} \approx \tau$
(2,0,4,5;5) GPC	11	5	6	4	1.5	$2\tau_\mathrm{L} + \tau_\mathrm{R} + 4\tau_\mathrm{CC} \approx 3\tau$
Proposed 4:2 compressor chain with k BLE:						
4:2 compressor	$4k$	$2k + 1$	$2k$	k	$2 - \frac{2}{k}$	$\tau_\mathrm{L} + k\tau_\mathrm{CC}$

[1] (6;3) and (1,5;3) are also included in [PANBI11]

3. Proposed Improved Compression Elements

3.1. General Considerations

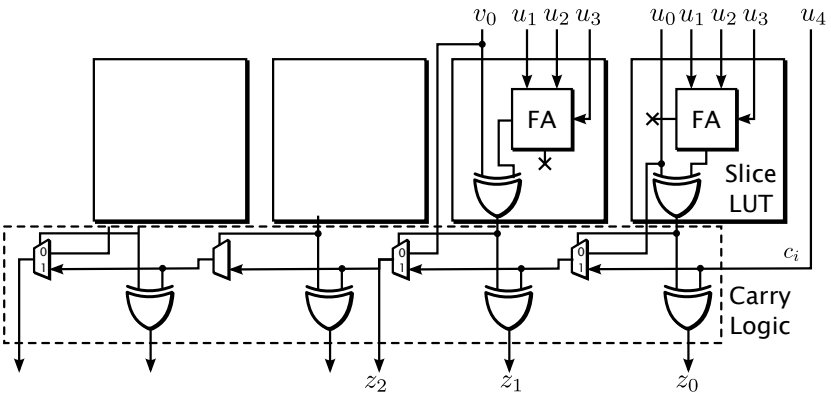
From the efficiency metric of Section 2.4 is becomes clear that 1) the number of input bits per BLE has to be maximized and 2) the carry-chain resources have to be used in an efficient manner. The first issue is not the case for most of the GPCs presented in [PANBI11] as can be seen from the examples in Figure 1b and Figure 1d in which parts of the LUTs are unused. The second issue is obviously not the case for the LUT based GPCs of [BdDI+13] as they don't use any carry-chain resource.
Additional restrictions are given by some limitations of the slice structure. The first limitation concerns the carry-in. There is only one slice input ({A/B/C/D}X [Xil12]) that can be either routed to the carry-in of the first stage or to the 0-input of the carry-chain multiplexer. Alternatively, this carry-chain multiplexer can be fed from the LUT. As one LUT output is used for the second full-adder stage, there are two choices for the least significant stage: either using the second LUT output for additional computations without using the carry-in (this has to be set to zero) or using the second LUT output as routing resource to the carry-chain multiplexer to use the carry-in without doing additional computations. The second limitation concerns the carry-out of the last stage. There are only two outputs per quarter slice which are able to route out the carry-out or the sum result (via {A/B/C/D}MUX output) and the O5 output of the LUT or the sum result (via {A/B/C/D}Q output) [Xil12]. Hence, the O5 LUT output of the last stage can not be routed to a previous stage.

3.2. Improved GPC mappings

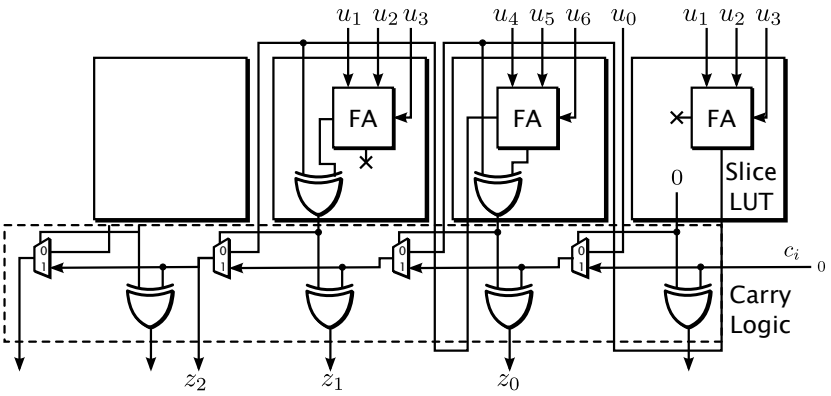
The main limitation of the GPCs proposed in [PANBI11] is that the considerations mentioned above are not fully used, i. e., BLEs are wasted for routing the carry-in or carry-out signals. Examples for improved mappings of the $(1, 5; 3)$ and $(7; 3)$ GPCs of Figure 1 are given in Figure 3. The GPCs for which improved mappings were found are listed in the lower half of Table 1. It can be seen that most of the GPCs can be optimized leading to higher efficiencies.

3.3. Proposed GPCs

The compressing elements considered so far use only a low amount of BLEs. If the number of BLEs is not a multiple of four, the unused parts of the carry chain can not be used for further arithmetic operations – only the LUTs and flip-flops (FFs) can be used. Hence, new GPCs are proposed that try to utilize the full slice as much as possible. The resulting GPC FPGA mappings are shown in Figure 4. Their properties are also listed in the lower part of Table 1.
All GPCs achieve an efficiency of 1.5 while the GPCs (5,0,6;5), (1,4,1,5;5) and (1,4,0,6;5) are designed such that no routing from one LUT to another occurs and, thus, the delay is minimal. If delay or speed is not that critical, the (2,0,4,5;5) GPC may be used in addition to better fit other input patterns.
Their irregular shape, i. e., the large differences of the input numbers for different weights, can be compensated by combining the compressing elements in a shifted manner. Combining two $(5, 0, 6; 5)$ GPCs where one GPC is left shifted by one bit can cover the input pattern $(5, 5, 6, 6)$, which is much more regular. However, such cases should be also found by automated synthesis tools [PANBI11, BdDI+13].



(a) (1,5;3) GPC



(b) (7;3) GPC

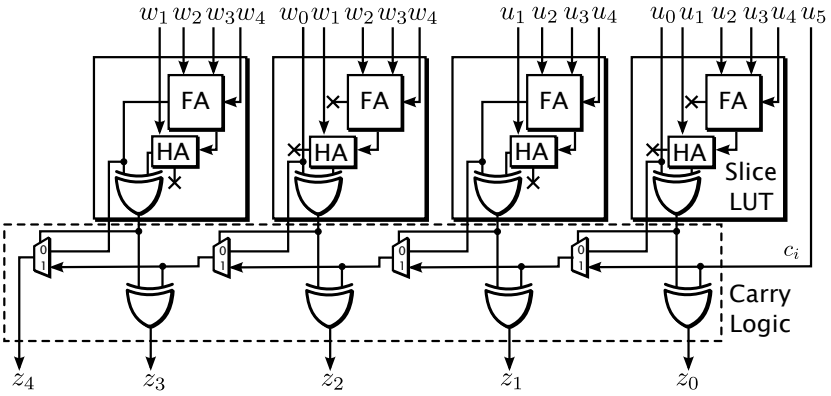
Figure 3: Improved GPC mappings

3.4. Proposed 4:2 Compressor

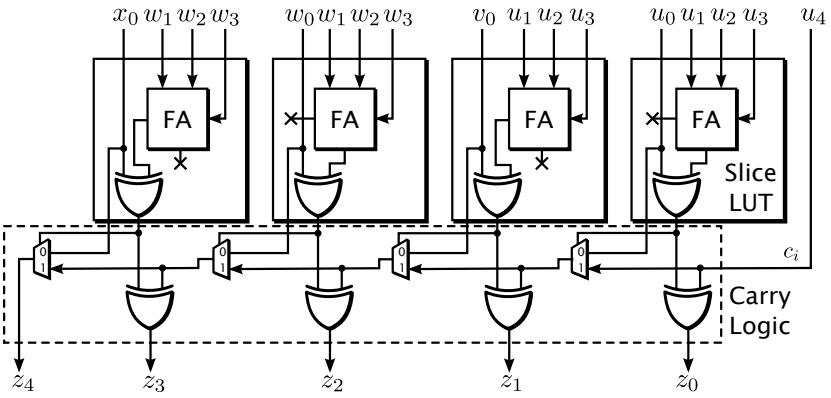
As it was shown before, the ternary adder has the best efficiency for a large number of BLEs (k) or large word sizes but has a poor delay. But its basic configuration can be used as a 4:2 compressor by simply eliminating the carry propagation to the second full-adder stage, leading to one additional input and one additional output per BLE, resulting in the structure shown in Figure 5. Thus, the combinatorial delay is reduced to a single LUT delay while the efficiency remains the same as for the ternary adder. For a full slice configuration with $k = 4$, the efficiency is identical to the low-delay GPCs from the last section (E=1.5) but is further increasing for larger k. Note that the last stage has to be reduced to two inputs due to the output restrictions.

4. Results

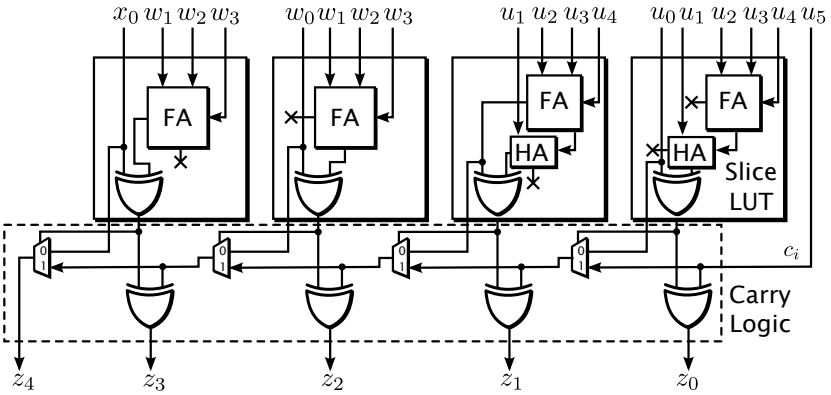
To evaluate the performance of the proposed compressing elements, we designed and synthesized pipelined compression trees with eight 10-bit inputs using different techniques. Even for this relatively low word size, the ternary adder and the 4:2 compressor lead to the best efficiency of $E_k = 1.8$ using $k = 10$ BLEs. The ternary adder tree requires two stages with four ternary adders in total while the compressor tree with 4:2 compressors requires three stages with three 4:2 compressors in total plus one common two-input adder to merge the result.



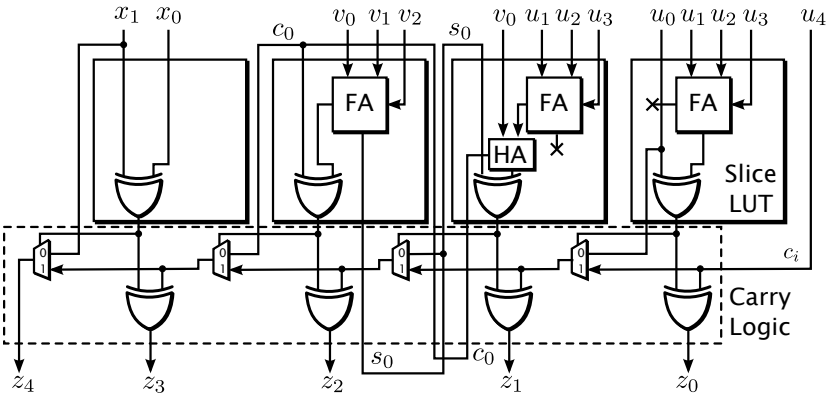
(a) $(5, 0, 6; 5)$ GPC



(b) $(1, 4, 1, 5; 5)$ GPC

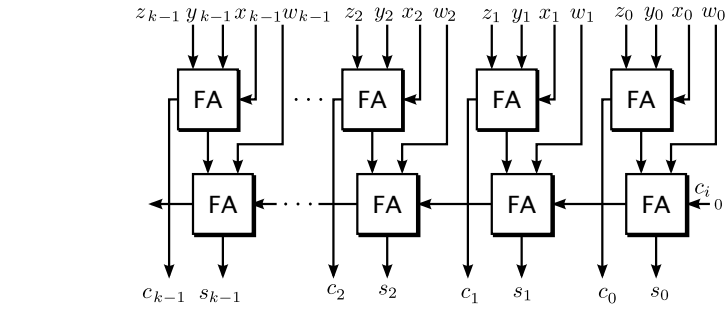


(c) $(1, 4, 0, 6; 5)$ GPC

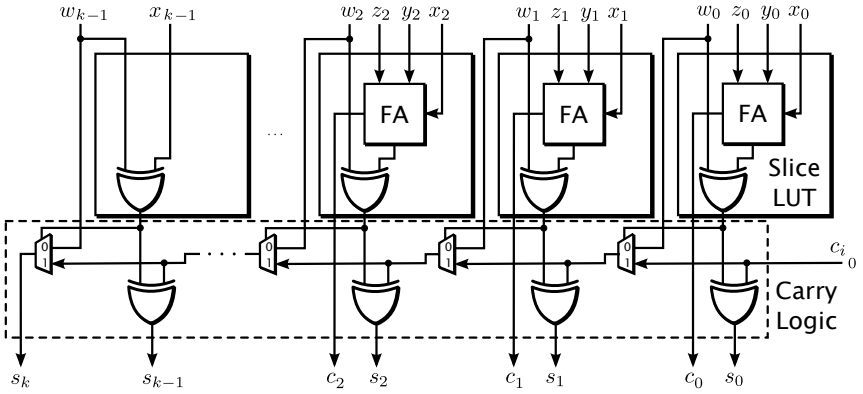


(d) $(2, 0, 4, 5; 5)$ GPC

Figure 4: Proposed GPC slice mappings



(a) General Structure



(b) Slice Mapping

Figure 5: Proposed 4:2 compressor

We compared the results with the bit heap compression tree optimization of the FloPoCo framework [BdDI+13] which uses LUT-based compressing elements for the Xilinx target. It offers an automated pipelining mechanism where pipeline registers are placed according to a timing model of the FPGA and a user-constrained target frequency f_t. The FloPoCo target frequency has be set to f_t = 400 MHz and f_t = 700 MHz. The synthesis was performed for a Xilinx Virtex 6 FPGA (XC6VLX75T-FF784) with speed grade 2. The results are listed in Table 2. The timing results were obtained by registering inputs and outputs to avoid that the critical path is dominated by the routing from the FPGA pin to the input of the circuit. The resource usage results do not include these registers.

As expected, the least slice resources are obtained by using ternary adders or 4:2 compressors. In terms of speed, the 4:2 compressor tree offers a 33% higher speed compared to the ternary adder while using the same amount of slices. The speed can be further increased by pipelining a LUT-based compressor design as shown by the FloPoCo design with f_t =700 MHz, resulting in a circuit with more than 900 MHz. However, it will be hard to keep this speed when the compressor is embedded within a more complex application. Furthermore, this speed has to be paid for by a huge amount of extra resources which is a factor of seven compared to the 4:2 compressor design.

5. Conclusion and Outlook

It was shown in this work that better compressing elements can be found by evaluating the low-level structure of the FPGA. Novel compressing elements for modern Xilinx devices were proposed including different GPCs and a 4:2 compressor based on a ternary adder. All of them provide a

Table 2: Synthesis results for an 8-input 10 bit pipelined adder tree using different methods

Method	Slices	FF	LUT	f_{max} [MHz]	T_{min} [ns]
FloPoCo [BdDI+13] (f_t = 400 MHz)	35	21	70	435.73	2.30
FloPoCo [BdDI+13] (f_t = 700 MHz)	91	145	145	929.37	1.08
ternary adder tree	13	49	49	491.88	2.03
proposed 4:2 compressor	13	65	47	657.46	1.52

better efficiency, a lower delay or both compared to previous compressing elements. They can be pipelined without overhead using the otherwise unused flip-flops in the device. A design example of a pipelined compressor tree showed the effectiveness of the 4:2 compressor.
Further work has to be done in the automatic synthesis of pipelined compressor trees. Previous work only focused on non-pipelined compressor trees although this quite limits the speed of the design. However, the strategy for pipelining is different as each input bit has to be covered by at least a single flip-flop for each compression stage to get a balanced pipeline. Another issue is the selection of the best compressing element. While the used efficiency metric [BdDI+13] assumes that all compressing element inputs can be covered by inputs of the compressor tree, this may not always be the case. Here, one has to find the element leading to the best resulting efficiency. While this was partly solved for the non-pipelined case [PANBI11] it is still open for efficient pipelining.

References

[BdDI+13] Brunie, Nicolas, Florent de Dinechin, Matei Istoan, Guillaume Sergent, Kinga Illyes, and Bogdan Popa: *Arithmetic Core Generation Using Bit Heaps*. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–8, 2013.

[BLSY09] Baeckler, Gregg, Martin Langhammer, James Schleicher, and Richard Yuan: *Logic Cell Supporting Addition of Three Binary Words*. US Patent No 7565388, Altera Coop., 2009.

[dDP12] Dinechin, Florent de and Bogdan Pasca: *Designing Custom Arithmetic Data Paths with FloPoCo*. IEEE Design & Test of Computers, 28(4):18–27, 2012.

[EL04] Ercegovac, Miloš D and Tomás Lang: *Digital Arithmetic*. Elsevier, 2004.

[flo13] *FloPoCo Project Website*, 2013. `http://flopoco.gforge.inria.fr`.

[GAKC09] Gao, Shuli, Dhamin Al-Khalili, and Noureddine Chabini: *Implementation of Large Size Multipliers Using Ternary Adders and Higher Order Compressors*. In *International Conference on Microelectronics (ICM)*, pages 118–121. IEEE, 2009.

[GAKCL12] Gao, S, D Al-Khalili, N Chabini, and P Langlois: *Asymmetric Large Size Multipliers with Optimised FPGA Resource Utilisation*. Computers & Digital Techniques, IET, 6(6):372–383, 2012.

[HVZ13] Hormigo, J, J Villalba, and E L Zapata: *Multioperand Redundant Adders on FPGAs*. IEEE Transactions of Computers, 62(10):2013–2025, 2013.

[KHW+13] Kumm, Martin, Martin Hardieck, Jens Willkomm, Peter Zipf, and Uwe Meyer-Baese: *Multiple Constant Multiplication with Ternary Adders*. In *IEEE International Conference on Field Programmable Logic and Application (FPL)*, pages 1–8, 2013.

[KZ11] Kumm, Martin and Peter Zipf: *High Speed Low Complexity FPGA-Based FIR Filters Using Pipelined Adder Graphs*. In *IEEE International Conference on Field-Programmable Technology (FPT)*, pages 1–4, 2011.

[OQH+09] Ortiz, M, F Quiles, J Hormigo, F J Jaime, J Villalba, and E L Zapata: *Efficient Implementation of Carry-Save Adders in FPGAs*. In *IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP)*, pages 207–210, 2009.

[PA12] Parandeh-Afshar, Hadi: *Closing the Gap Between FPGA and ASIC: Balancing Flexibility and Efficiency*. PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2012.

[PAI11] Parandeh-Afshar, Hadi and Paolo Ienne: *Measuring and Reducing the Performance Gap between Embedded and Soft Multipliers on FPGAs*. In *International Conference on Field Programmable Logic and Applications (FPL)*, pages 225–231. IEEE, 2011.

[PANBI11] Parandeh-Afshar, H., A. Neogy, P. Brisk, and P. Ienne: *Compressor Tree Synthesis on Commercial High-Performance FPGAs*. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 4(4):1–19, 2011.

[SP06] Simkins, James M and Brian D Philofsky: *Structures and Methods for Implementing Ternary Adders/Subtractors in Programmable Logic Devices*. US Patent No 7274211, Xilinx Inc., March 2006.

[Xil12] Xilinx, Inc.: *Virtex-6 FPGA Configurable Logic Block User Guide (UG364 v1.2)*, February 2012.