2011 20th IEEE Symposium on Computer Arithmetic

Accelerating Computations on FPGA Carry Chains by Operand Compaction

Thomas B. Preußer, Martin Zabel, Rainer G. Spallek
Institute of Computer Engineering
Technische Universität Dresden
Dresden, Germany
Email: {thomas.preusser, martin.zabel, rainer.spallek}@tu-dresden.de

***Abstract*—This work describes the carry-compact addition (CCA), a novel addition scheme that allows the acceleration of carry-chain computations on contemporary FPGA devices. While based on concepts known from the carry-lookahead addition and from parallel prefix adders, their adaptation by the CCA takes the context of an FPGA as implementation environment into account. These typically provide carry-chain structures to accelerate the simple ripple-carry addition (RCA). Rather than contrasting this scheme with the hierarchical addition approaches favored in hard-core VLSI designs, the CCA combines the benefits of both and uses hierarchical structures to shorten the critical path, which is still left on a core carry chain. In contrast to previous studies examining the asymptotically superior parallel prefix adders on FPGAs, the CCA is shown to outperform the standard RCA already for operand widths starting at 50 bits. Wider adders such as used in extended-precision floating-point units and in cryptographic applications even benefit from increasing speedups. The concrete mapping of the CCA as achieved for current Xilinx and Altera architectures is described and shown to be very favorable so as to yield a high speedup for a very modest investment of additional LUT resources.**

***Keywords*-FPGA, Carry Chains, Addition**

I. Introduction

The carry-chain structures found in virtually all modern FPGA architectures provide fast direct signal links between adjacent logic blocks. Designed for the acceleration of the binary word addition, they have also attracted many other applications seeking for an acceleration of their critical paths (cf. [1]–[3]). Techniques for mapping combinational functions to carry-chain structures are known for the Stratix carry chain specifically [4] and for additive carry chains in general [5]. The great benefits of these structures have been re-affirmed for contemporary devices for the classic example of binary word addition by Vitoroulis and Al-Khalili [6]. They showed that carry chains enable a simple ripple-carry adder (RCA) to outperform asymptotically superior parallel prefix networks for operand widths far beyond 250 bits.

Our work challenges the rather high bound of 250 bits and demonstrates that carry-chain computations of only about 50 stages within the chain already benefit from an advanced hierarchical carry computation. The key feature of the presented approach is a compacted carry path, which is, nonetheless, kept *within* the carry chain. Thus, it draws from both an advanced adder architecture and the available special-purpose hardware structures instead of seeing them as opposing implementation alternatives. Additionally, the hierarchical carry compaction offers a low and constant signal fanout. This makes it superior to other approaches such as carry-select addition.

The compaction of long carry chains bears further advantages for the placement and routing of larger designs. Due to the designated direct signal links constituting the chain, these computations must be placed as a big bulky chunk of logic occupying a rectangular area with a large aspect ratio. Breaking this big monolithic structure regains placement freedom and allows greater locality by more flexible shaping.

This work proposes a novel addition scheme, the carry-compact addition (CCA). The CCA is directly tailored towards an implementation on contemporary FPGA devices. It shortens the core carry-chain computation as well as the overall critical delay. It, thus, accelerates the computation and increases the mappability of the component. Although its presentation focuses on addition, the CCA scheme can be used to back arbitrary carry-chain computations by the transformation proposed by Preußer and Spallek [7].

In the remainder of this paper, Sec. II will introduce the arithmetic foundations of the related parallel-prefix and carry-lookahead schemes. Sec. III will derive the CCA as an adaptation of the same fundamental concepts for an optimized implementation on a contemporary Virtex-5/6 architecture[1]. Interesting design metrics as its logic resource consumption and the length of the remaining carry chain will be analyzed in Sec. IV. The actual performance of the CCA has been explored experimentally as described in Sec. V. After Sec. VI has discussed the generalization of the addition scheme to other FPGA architectures and higher radices, Sec. VII will conclude this paper.

II. Arithmetic Foundations

As shown in Fig. 1, the carry chains available on modern FPGA architectures provide fast direct links for rippling carry signals between adjacent logic blocks. They avoid the slow general-purpose programmable routing network and, thus, make the basic RCA an attractive implementation

[1]This architecture has also been adopted for the low-cost Spartan-6 device family.

1063-6889/11 $26.00 © 2011 IEEE
DOI 10.1109/ARITH.2011.22

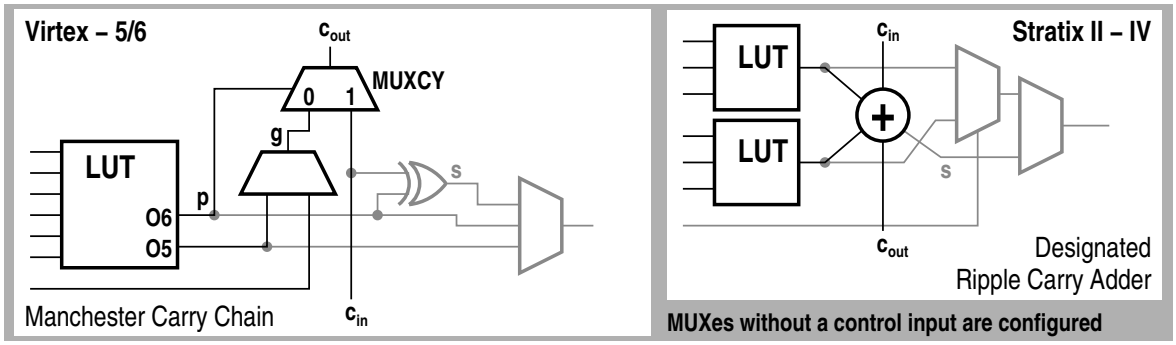

Figure 1. Simplified Carry-Chain Structures in the Logic Blocks of Contemporary FPGA Devices

choice on an FPGA. For an addition of the bit vectors $a+b=s$, each logic block on the carry chain will typically implement the function of one full adder computing one sum bit s_i and the carry c_{i+1} for the next bit position:

$$s_i = p_i \oplus c_i \quad \text{– Sum Bit} \tag{1}$$

$$c_{i+1} = \overline{p_i} g_i + p_i c_i \quad \text{– Carry} \tag{2}$$

$$\text{with} \quad g_i = a_i b_i \quad \text{– Generate} \tag{3}$$

$$p_i = a_i \oplus b_i \quad \text{– Propagate} \tag{4}$$

The critical path of the direct implementation of these equations spans the whole width of the argument and sum vectors due to the carry computation of Eq. (2). Its delay, thus, grows linearly with the operand size, i.e. in $O(n)$.

In hard-core VLSI adders, hierarchical carry computation schemes are used to shorten the critical path. A technique common to parallel prefix networks and carry-lookahead structures is the composition of *generate* and *propagate* signals (*GP*-signals) for *groups* of consecutive bit positions [8], [9]:

$$(G_{i+1:i}, P_{i+1:i}) = (g_{i+1} + p_{i+1} g_i, p_{i+1} p_i) \tag{5}$$

$$= (g_{i+1}, p_{i+1}) \circ (g_i, p_i) \tag{6}$$

The grouping operator $\circ$ is idempotent and associative. It can be applied to compute the *GP*-signals for wider groups spanning a consecutive range i to j of bit positions:

$$(G_{j:i}, P_{j:i}) = \bigcirc_{\kappa=i}^{j} (g_\kappa, p_\kappa) \tag{7}$$

The possible tree structure allows a computation spanning n bit positions to be completed within $O(\log(n))$ logic levels.

Parallel prefix networks employ this approach to compute the $c_{i+1} = G_{i:0}$ for all or some equidistant bit positions. The latter sparse carry computation is completed by appropriate RCA blocks filling the gaps. The grouping hierarchy of a carry-lookahead adder, on the other hand, may terminate arbitrarily in a skip logic passing inter-group carries after Eq. (2). A computation walking *up* the group hierarchy finally applies Eq. (2) internally to expand the inter-group carries to the carries for the individual sum bit positions.

Table I
CHOSEN COMPUTATION OF COMPACTED PSEUDO ADDENDS

g_{i+1}	p_{i+1}	b_i	a_i	$G_{i+1:i}$	$P_{i+1:i}$	Case	$B_{i+1:i}$	$A_{i+1:i}$
0	0	*	*	0	0	K	0	0
0	1	0	0	0	0	K	0	0
0	1	0	1	0	1	P	0	1
0	1	1	0	0	1	P	1	0
0	1	1	1	1	0	G	1	1
1	*	*	*	1	0	G	1	1

While grouping induces extra effort to compute the *GP*-signals and to derive the individual carries, it also reduces the whole group to a single stage within the carry propagation path. Acknowledging the tremendous speed advantage of the hardware carry chains on FPGAs, we adopt the grouping approach but apply it only selectively when a benefit can be expected. The hierarchical grouping shortens the critical path, which is, nonetheless, kept on the carry chain.

III. FPGA ADAPTATION

The traditional grouping by the carry-lookahead approach produces *GP*-signals describing the carry-related behavior of a block. As to avoid a device-specific manual approach to integrate these grouped stages into an FPGA carry chain, we follow the suggestion by Preußer and Spallek [7] and re-code the same information into two pseudo addend bits A and B so that:

$$\begin{array}{lll} \textbf{\textit{Generate}}: & A = B = 1 & \text{iff } G = 1, \\ \textbf{\textit{Propagate}}: & A \oplus B = 1 & \text{iff } P = 1, \text{ and} \\ \textbf{\textit{Kill}}: & A = B = 0 & \text{otherwise.} \end{array} \tag{8}$$

This allows the compacted groups to function as regular addend bits. The core carry chain can, thus, be inferred portably from using a standard addition operator on compacted argument vectors. Hence, we have called the resulting addition scheme *carry-compact addition* (CCA).

Observe that the addend recoding is not at all unique as the propagation case can be encoded as a one (1) either in A or in B. Since the lookup tables (LUTs) of FPGAs

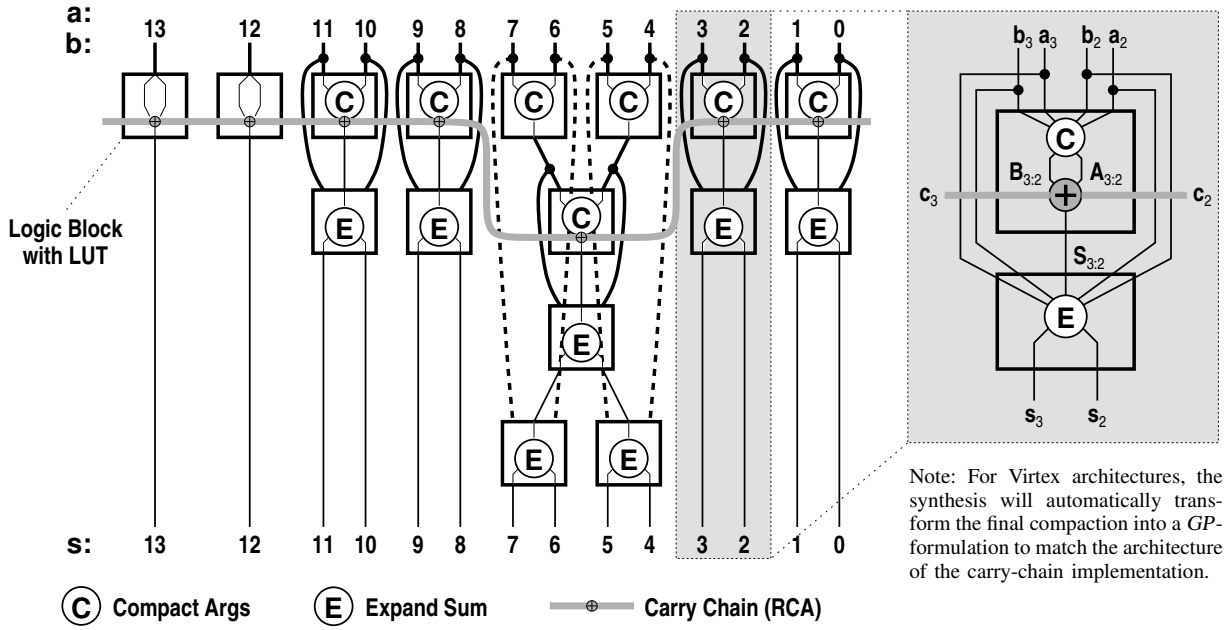

Figure 2. 14-Bit Radix-4 Carry-Compact Adder for a Virtex-5/6 Architecture ($N = 14, L = 2$)

implement arbitrary configurable functions on the supplied inputs, an FPGA adaptation must most importantly reduce the number of required function inputs. As shown in Tab. I, it is possible to derive two 3-input functions for a radix-4 CCA with a grouping degree of two (2):

$$A_{i+1:i} = g_{i+1} + p_{i+1}a_i \tag{9}$$
$$= b_{i+1}a_{i+1} + (b_{i+1} \oplus a_{i+1})\, a_i \tag{10}$$
$$B_{i+1:i} = g_{i+1} + p_{i+1}b_i \tag{11}$$
$$= b_{i+1}a_{i+1} + (b_{i+1} \oplus a_{i+1})\, b_i \tag{12}$$

The sum bits obtained by the ultimate addition for the grouped digits are no immediate result bits. Rather, they must be expanded to obtain the individual outputs for the two grouped input positions. In this computation, the group sum bits $S_{i+1:i}$ are used to infer the incoming group carry c_i as enabled by Eq. (1):

$$S_{i+1:i} = P_{i+1:i} \oplus c_i$$
$$c_i = S_{i+1:i} \oplus P_{i+1:i} \tag{13}$$
$$c_i = S_{i+1:i} \oplus ((b_{i+1} \oplus a_{i+1})\,(b_i \oplus a_i)) \tag{14}$$

This incoming group carry enables the local computation of both sum output bits:

$$s_i = p_i \oplus c_i$$
$$= (b_i \oplus a_i) \oplus c_i \tag{15}$$
$$s_{i+1} = p_{i+1} \oplus c_{i+1}$$
$$= p_{i+1} \oplus (g_i + p_i c_i)$$
$$= (b_{i+1} \oplus a_{i+1}) \oplus (b_i a_i + (b_i \oplus a_i)\, c_i) \tag{16}$$

Observe that with the expansion of c_i according to Eq. (14), both of these equations turn into 5-input functions with the same input set $\{S_{i+1:i}, b_{i+1}, a_{i+1}, b_i, a_i\}$.

The mapping of a radix-4 CCA to a Virtex-5/6 architecture is illustrated in Fig. 2. The 6-LUTs of these architectures may either implement an arbitrary 6-input function (output on `O6`), or they may be sliced into two 5-input functions sharing a common set of input signals (output on `O5` and `O6`) [10], [11]. Besides the core carry chain implementing an RCA, the structure contains compaction and expansion nodes, each of which fit into a single 6-LUT by using its two outputs `O6` and `O5`. Observe that the final compaction stage before the ripple-carry addition merges into the logic blocks with the carry chain.

Finally, regard the structural organization of the compaction. A few argument bits on the left side remained uncompacted and are, thus, added in a simple ripple-carry fashion. This is to account for the speed advantage of the carry chain and to hide the LUT delay of the expansion of the neighboring compacted groups. The same reasoning applies to the second level of the hierarchy. There, the compaction is also restricted on the right end of the chain as the additional compacting stage induces an extra LUT delay. Note that in practice, the delay of this LUT stage is heavily dominated by the inter-LUT routing rather than the LUT computation. The number of carry-chain stages needed to compensate a stage of LUT computation is a device parameter. Its choice is critical for the performance of the design since (a) too large a value will preclude valuable compaction opportunities and (b) too small a value will expose the critical path to the slower LUT computations.

IV. Design Metrics

For the evaluation of the CCA, we implemented a generic VHDL design parameterized by:

- N the width of the overall addition, i.e. its arguments and its sum, in bits, and
- L the length of the carry chain used for hiding the delay of a stage of LUT computation. $L = N$ is the greatest reasonable value as all $L \geq N$ result in a design degenerated into a plain RCA. In fact, this is true for $L = N - 1$ as well since it leaves only a single position, which cannot be compacted.

We further distinguished a full hierarchical CCA and a linear variant, in which the compaction hierarchy was not constructed beyond the very first compaction stage.

First, consider the design complexity in terms of the occupied LUTs. Recall from Fig. 2 that the first compaction level merges into the LUTs controlling the core carry chain. As one additional LUT completes the expansion for such a two-bit group, the linear CCA will occupy a total of n LUTs for an n-bit addition. This matches exactly the LUT count of the RCA on Virtex-5/6 devices. There, the CCA can thrive solely on a higher functional utilization of the involved LUTs without requiring additional ones.

If the compaction hierarchy is extended beyond the first level, two additional LUTs achieve the compaction and expansion of two subgroups. For the fully implemented hierarchy ($L = 0$), this will require $2n - 2$ LUTs for $n > 1$ and $n = 2^k (k \in \mathbb{N})$. If n is not a power of two, the LUT count remains strictly below this bound. A further reduction is achieved for $L > 0$, which hides the compaction and expansion delays in practical imlementations. As will be depicted in Fig. 6, a reasonable value of $L = 30$ results in a LUT overhead below 40% even for N just below 500 while also achieving more than twofold speedups. All in all, even the pessimistic bound of $2n - 2$ LUTs is a tremendous improvement over the parallel prefix architectures explored by Vitoroulis and Al-Khalili [6].

Another interesting metric is the length of the resulting core carry chain. The shorter it is, the more flexible the placement of the adder on the FPGA device becomes. Additionally, a shorter chain within a component allows it to adopt a more compact shape so that the locality of in- and outputs is increased.

Each compaction level $i = 0, 1, \ldots$ reduces the length of the carry chain in dependence of the parameter L. On the ith level, $(i + 1) \cdot L$ positions are preserved on the left and $i \cdot L$ positions are preserved on the right side of the carry-chain computation. The number of the remaining positions is halved by the compaction. Naming the initial bit width $n_0 = N$, this amounts to a stepwise reduction:

$$n_{i+1} = n_i - \left\lfloor \frac{n_i - (2i+1)\,L}{2} \right\rfloor \tag{17}$$

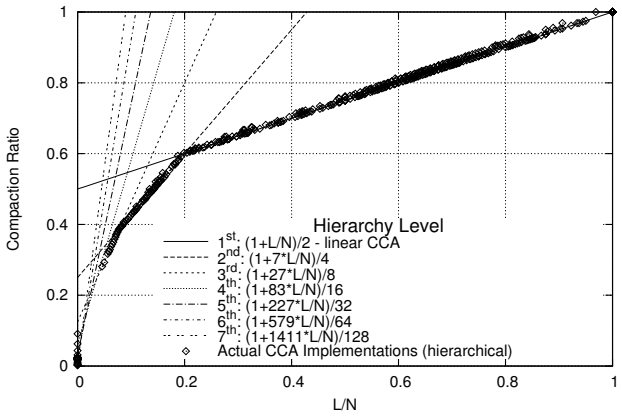

Figure 3. Compaction Ratio of CCA wrt. Plain RCA Implementation

The compaction terminates when $n_i \leq (2i + 1)\,L$. In particular, $L = N$ will produce a plain RCA and $L = 0$ will result in a fully constructed hierarchy with no remaining chain computation.

Neglecting the integer truncation in Eq. (17), the expression for the chain length can be made almost explicit:

$$n_i = \frac{1 + k_i \frac{L}{N}}{2^i} \cdot N \tag{18}$$
$$\text{with} \quad k_0 = 0$$
$$k_{i+1} = k_i + (2i + 1) \cdot 2^i$$

Hence, the compaction on a specific hierarchy level is a linear function of N with a coefficient depending on the $\frac{L}{N}$ ratio. The final carry-chain compaction ratio r of a completely constructed hierarchy with respect to a plain RCA is then determined by:

$$r = \min_{i \in \mathbb{N}} \left\{ \frac{1 + k_i \frac{L}{N}}{2^i} \right\} \tag{19}$$

The resulting piecewise linear function is sketched in Fig. 3. Note that its linear intervals become infinitesimally small as the $\frac{L}{N}$ ratio approaches zero (0). Each such interval represents one final depth of the compaction hierarchy where deep levels are only reached by sufficiently large N. Observe that due to the neglected integer truncation, true implementations will have marginally higher compaction ratios. For reference, the actual compaction ratios obtained for the parameter sets selected for our experimental timing evaluation are included in the figure.

V. Experimental Results

For the evaluation of the timing achievable by the CCA, the implemented design was synthesized for the modern Xilinx FPGA Virtex-6 LX75T by the Xilinx ISE 12.1 work flow. `LUTNM` attributes were used to constrain the logic placement of the two outputs of the compaction and of

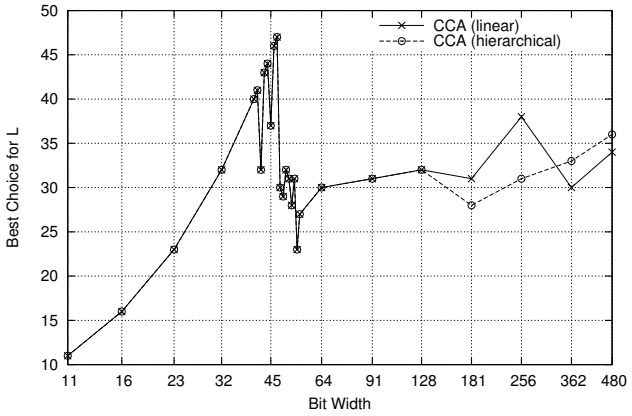

Figure 4. L Parameter Exploration for Virtex-6

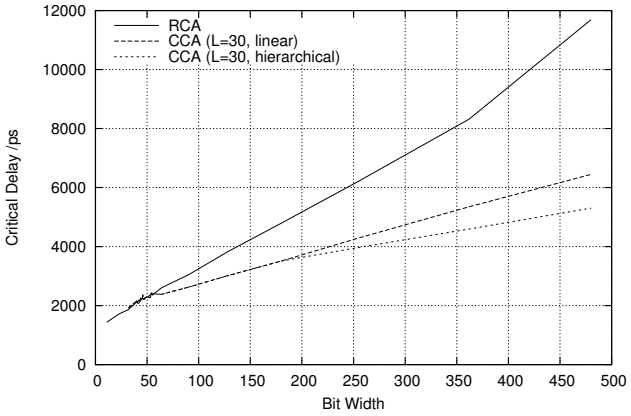

Figure 5. Critical Delays of RCA and CCA for Virtex-6

the expansion LUTs to a single Virtex-5/6 LUT site [12]. This does not only guarantee the envisioned effective use of LUT resources but also ensures that the achieved timing figures are not distorted in favor of the CCA by the use of faster `O6` paths only. Such figures would eventually fail when the design embedding the CCA grew to an extent that the mapper is forced to pack the addition into fewer LUTs by also using `O5` outputs. The measurement of the critical delay was performed by sandwiching the adder between input and output registers and repeated place-and-route runs with tightening clock contraints. We disallowed the registers to be placed into I/O blocks (IOBs) to allow their placement close by the adder structure.

For an initial overview over the design space, we chose N from the rounded powers of $\sqrt{2}$ between 11 and 362, inclusive and, additionally, the maximum chain length of 480 stages supported by the chosen target device. The values of L were explored in $\left\lfloor \sqrt{N} \right\rfloor$ roughly equidistant steps for each N. This choice increases the increment of the L exploration slowly with N so as to achieve a compromise between the effort and the granularity of the exploration. This empiric exploration revealed:

1) that the CCA performance improves over a standard RCA starting at an operand width of about 50 bits, and
2) that values of L around 30 produce the best performance for a given N.

Consequently, the exploration was intensified in these regions to cover all $N \in [40, 55]$ and all $L \in [27, 35]$.

Fig. 4 shows the best choices of L for each explored N. As L matches N for widths nearly up to 50 bits, the plain RCA is obviously the best implementation for these dimensions. From there, the graph follows a horizontal trend fluctuating around a value of 30. This fluctuation reveals the major drawback of the carry-compact adder. In contrast to the plain RCA, it depends heavily on the general-purpose routing and is heavily susceptible to differences in the routing success. Nonetheless, it is rather surprising that the initial linear growth of L with N obviously peaks above the horizontal level instead of joining with it directly at the point where the chains preserved by L match the delay of a LUT stage.

The viable employment of the CCA requires to relieve the designer from making a suitable choice for L. Since we expected a more or less constant value for covering the delay of an additional LUT stage, we computed the geometric mean over the critical delays determined for each specific $L \in [27, 35]$ for all explored $N \geq 50$. This yielded a minimum mean for $L = 30$, which we hence treat as the value of choice for our target device.

A direct comparison of the performances of the two CCA structures and the baseline RCA is shown in Fig. 5. As we chose $L = 30$, all three designs must be structurally identical up to this bit width. Although the CCA designs separate from the RCA thereafter, there is no clear winner up to $N = 50$ when the CCA finally prevails. Its linear and hierarchical variants are structurally equivalent as long as $N \leq 5 \cdot L = 150$. Although both quickly outperform the RCA, the hierarchical CCA enjoys a structural benefit for big bit widths n as its critical path only grows with $O(\log n)$ as compared to $O(n)$ for the linear variant.

Fig. 6, finally, illustrates the speedup achieved by the proposed CCA architectures. At an adder width of 256 bits, the speedup already grows beyond 1.4 and 1.5 for the linear and the hierarchical structure, respectively. At this bit width, the prefix adders analyzed by Vitoroulis and Al-Khalili [6] were still inferior to the basic RCA in spite of their rather high consumption of logic resources. At a width of 480 bits, the hierarchical CCA is already more than twice as fast as the baseline RCA. Finally, recall that the linear CCA did not require any additional LUT resources and observe that the LUT overhead of the hierarchical expansion compares very favorably against the achieved speedup.

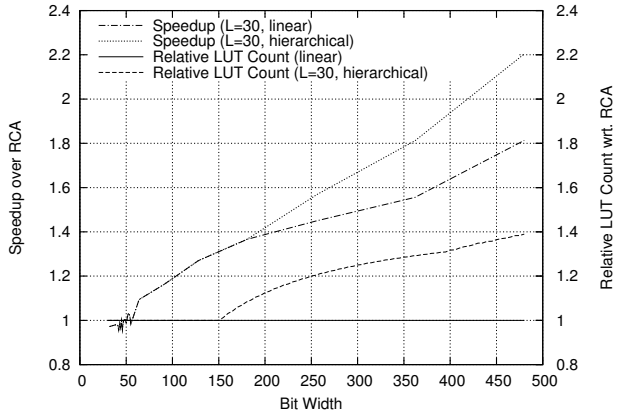

Figure 6. Speedup of the CCA over the RCA for Virtex-6

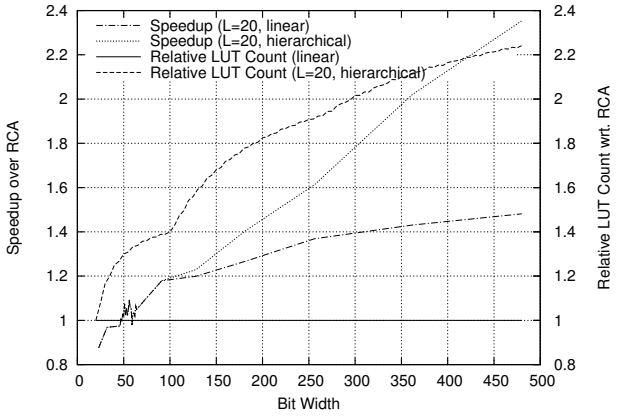

Figure 8. Speedup of the CCA over the RCA for Stratix-IV

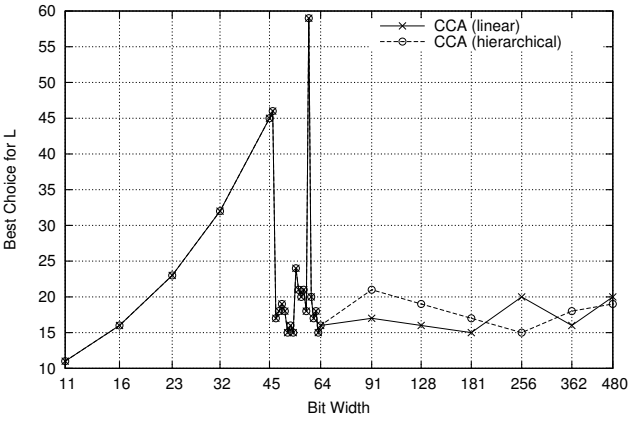

Figure 7. L Parameter Exploration for Stratix-IV

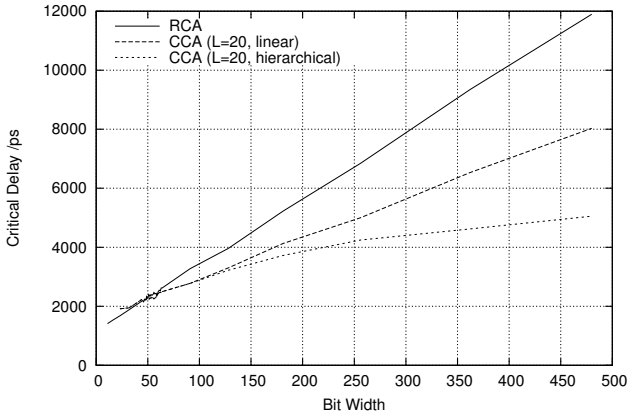

Figure 9. Critical Delays of RCA and CCA for Stratix-IV

VI. Generalizations

A. Altera Devices

While the proposed CCA was shown to map well to a Virtex-5/6 architecture, it is also suited for the contemporary devices by Altera. Review the chosen computation of the compacted pseudo addends as described by Eq. (10) and Eq. (12). Both of these addends can be computed in the same half of an Adaptive Logic Module (ALM). As such a half also controls one stage of the carry chain of the integrated RCA, the final compaction stage again merges into the core carry-chain computation. The expansion nodes, however, require a full ALM to expand both sum bits. Thus, the LUT costs increase by roughly 50% as compared to the Virtex-6 as shown by Fig. 8. Nonetheless, the number of logic levels for the expansion is maintained.

The achievable timing improvement was measured using Altera's Quartus II 10.0 software for a Stratix-IV GX70 FPGA, which is comparable to the Virtex-6 LX75T. The remaining measurement setup is identical to the Xilinx flow described above. The exploration of the best choice of L for several N (Fig. 7) shows again that the RCA is the best implementation for adder sizes up to 50 bits. For greater adder sizes, the computation of the geometric mean over the critical delays for each specific $L \in [15, 25]$ yields, that $L = 20$ is a suitable choice to hide the delay of an additional ALM stage. This value is lower than that obtained for the Virtex-6 ($L = 30$) and, thus, leads to higher speedups as seen in Fig. 9 and Fig. 8. The lower L of 20 also results in an earlier separation of the performance curves of the linear and the hierarchical CCA starting at $N = 5 \cdot L = 100$.

B. Higher Radices

The hierarchical depth of the CCA can be reduced by increasing the grouping degree to obtain higher-radix CCA structures. This is, however, only beneficial as long as the computation of a single compaction and expansion stage can be implemented on a single LUT level. Thus, the extension to a radix-8 compaction over three bit positions remains as the only feasible alternative. Generalizing the compaction

after Eq. (9) and Eq. (11), the pseudo addends of a radix-8 CCA are obtained as:

$$\begin{aligned} A_{i+2:i} &= G_{i+2:i+1} + P_{i+2:i+1} a_i \\ &= b_{i+2} a_{i+2} + (b_{i+2} \oplus a_{i+2})\, b_{i+1} a_{i+1} \\ &\quad + (b_{i+2} \oplus a_{i+2})\, (b_{i+1} \oplus a_{i+1})\, a_i \qquad (20) \\ B_{i+2:i} &= G_{i+2:i+1} + P_{i+2:i+1} b_i \\ &= b_{i+2} a_{i+2} + (b_{i+2} \oplus a_{i+2})\, b_{i+1} a_{i+1} \\ &\quad + (b_{i+2} \oplus a_{i+2})\, (b_{i+1} \oplus a_{i+1})\, b_i \qquad (21) \end{aligned}$$

These can be implemented in two Virtex-5/6 LUTs or in a single Stratix-II+ ALM. Thus, the radix-8 compaction compares well with the radix-4 implementation. They both have the same functional density and achieve a compaction by two bit positions within two Virtex-5/6 LUTs or a single Stratix-II+ ALM. The compaction ratio of 3:1 per LUT level makes the radix-8 compaction even superior to radix-4. Note, however, that the final compaction stage immediately preceding the carry-chain computation should be kept at radix-4 to allow it to merge into the logic blocks of the carry chain.

Unfortunately, the sum bit expansion out of a radix-8 group turns out to grow quite complex and requires three 7-input functions. This effort may be reduced slightly by exploiting device-specific features. For example, Xilinx architectures typically allow to tap the signal on the carry chain so that its reconstruction from the group sum can be omitted. Nonetheless, the radix-8 CCA suffers from a disproportion of its growth of width along with a further reduction in height due to the even shorter carry-chain length. This induces long routing paths as illustrated in Fig. 10 for a Virtex-5/6. While the radix-8 CCA achieves a better compaction of the carry chain than the radix-4 CCA, this is no longer its critical path. This is rather located in the horizontal LUT-based computation for the argument compaction and even more so for the sum bit expansion.

C. Sumless Operation

While the radix-8 CCA did not show any benefits for the calculation of a sum, it proves a valuable option if only the final carry output of the implemented addition is of interest. Prominent examples for such computations are integer comparators. As they can completely omit the sum bit expansion, its associated delay does no longer restrict the construction of the expansion hierarchy. Thus, the depth of the hierarchy can constantly grow towards the left by one LUT level every L carry-chain stages. For illustration, see the 16-bit comparator in Fig. 11. Recall the restriction of the compaction degree to two on the final level as to allow it to merge into the logic blocks controlling the carry chain. The small value of $L = 2$ is unrealistic and only chosen to allow a concise image.

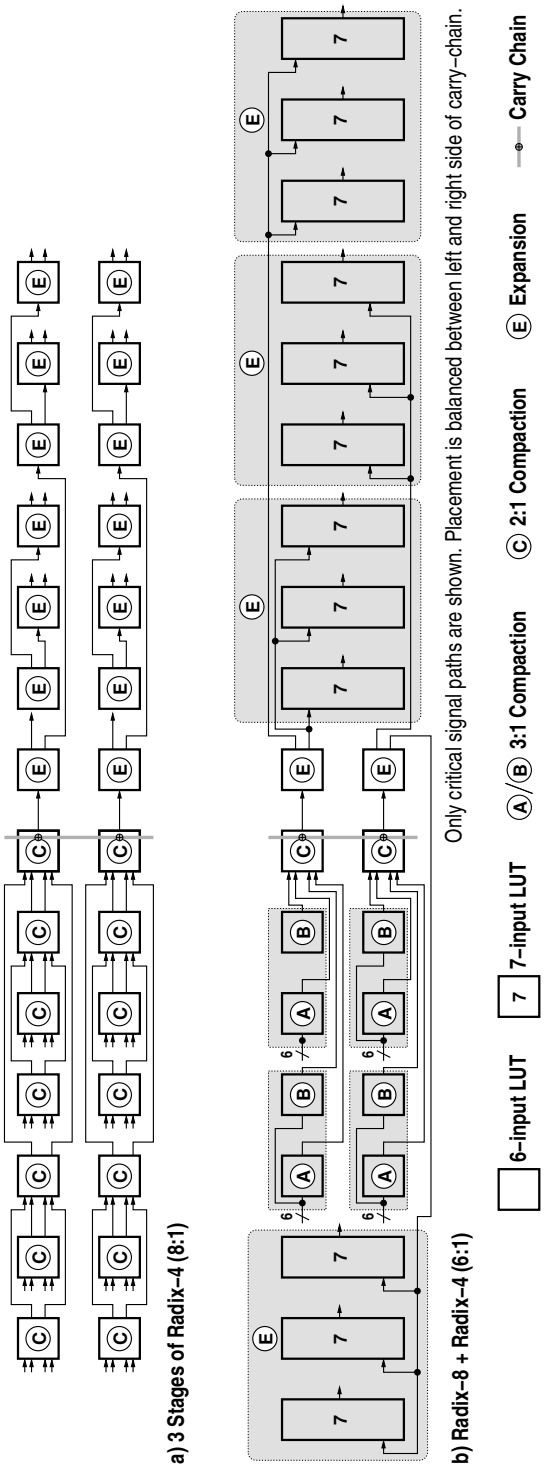

Figure 10: Placement of Radix-4 and Radix-8 Stages on a Virtex-5/6

VII. Conclusions

This paper has presented the carry-compact addition scheme. While its central concept is inspired by the carry-lookahead addition, it is distinguished by its internal use of compacted pseudo-addends and its selective formation of

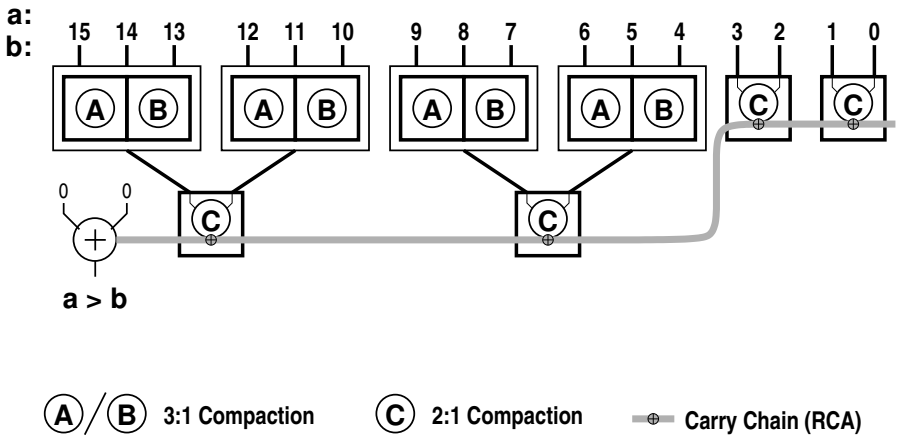

Figure 11. 16-bit Integer Comparator with Compacted Carry Path ($N = 16, L = 2$)

compaction groups. It is these features that qualify the carry-compact addition scheme well for an implementation on contemporary FPGA architectures as shown for the Xilinx Virtex-5/6 and Altera Stratix-II+ devices. In contrast to previous experiments of implementing hierarchical addition schemes on FPGAs, a carry-compact addition already outperforms the basic ripple-carry adder for operand widths starting at 50 bits. The area costs of this scheme in terms of the LUT usage have been shown to be modest. They equal those of the ripple-carry adder for a linear compaction and are bounded below the twofold (Virtex-5/6) or threefold (Stratix-II+) LUT counts even for an impractical full construction of the compaction hierarchy.

We believe that the CCA addition scheme is an attractive option for implementing adders and other carry-chain computations on modern FPGA devices. As to relieve the designer from having to choose the most beneficial adder structure and parameters, a direct integration of this scheme as a standard implementation within the synthesis libraries would be most favorable. This would allow to capture device-specific parameters transparently. The improved synthesis results would increase both the customer satisfaction with the tool chain and the experienced FPGA device capabilities.

References

[1] J. Põldre and K. Tammemäe, "Reconfigurable multiplier for Virtex FPGA family," in *FPL '99: Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*. London, UK: Springer-Verlag, 1999, pp. 359–364.

[2] T. B. Preußer, M. Zabel, and R. G. Spallek, "About carries and tokens - re-using adder circuits for arbitration," in *IEEE Workshop on Signal Processing Systems (SIPS'05)*. IEEE Press, 2005.

[3] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *International Conference on Field Programmable Logic and Applications (FPL) 2009*. IEEE, Aug. 2009, pp. 242–249.

[4] M. T. Frederick and A. K. Somani, "Beyond the arithmetic constraint: depth-optimal mapping of logic chains in LUT-based FPGAs," in *FPGA '08: Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*. New York, NY, USA: ACM, 2008, pp. 37–46.

[5] T. B. Preußer and R. G. Spallek, "Enhancing FPGA device capabilities by the automatic logic mapping to additive carry chains," in *International Conference on Field Programmable Logic and Applications (FPL) 2010*. IEEE, Aug. 2010.

[6] K. Vitoroulis and A. Al-Khalili, "Performance of parallel prefix adders implemented with FPGA technology," in *Circuits and Systems, 2007. NEWCAS 2007. IEEE Northeast Workshop on*, Aug. 2007, pp. 498–501.

[7] T. B. Preußer and R. G. Spallek, "Mapping basic prefix computations to fast carry-chain structures," in *International Conference on Field Programmable Logic and Applications (FPL) 2009*. IEEE, Aug. 2009, pp. 604–608.

[8] R. E. Ladner, Michael, and J. Fischer, "Parallel prefix computation," *Journal of the ACM*, vol. 27, pp. 831–838, 1980.

[9] S. Knowles, "A family of adders," in *15th IEEE Symposium on Computer Arithmetic*, 2001, pp. 277–281.

[10] *Virtex-5 FPGA User Guide*, Xilinx Inc., May 2010, UG190 (v5,3): http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

[11] *Virtex-6 FPGA Configurable Logic Block User Guide*, Xilinx Inc., Sep. 2009, UG364 (v1.1): http://www.xilinx.com/support/documentation/user_guides/ug364.pdf.

[12] *Constraints Guide*, Xilinx Inc., Jul. 2010, UG625 (v12.2): http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_2/cgd.pdf.

Availability

The generic VHDL implementation of the carry-compact adder is available on http://vlsi-eda.inf.tu-dresden.de/.