FPGA-Specific Arithmetic Optimizations of Short-Latency Adders

Hong Diep Nguyen, Bogdan Pasca, Thomas B. Preusser

FPGA-Specific Arithmetic Optimizations of Short-Latency Adders

LIP Research Report RR2010-35

Hong Diep Nguyen[1], Bogdan Pasca[1],Thomas B. Preußer[2]
[1]LIP (ENSL-CNRS-Inria-UCBL), Ecole Normale Superieure de Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France
Email: {hong.diep.nguyen, bogdan.pasca}@ens-lyon.org
[2]Institute of Computer Engineering
TU Dresden, Germany
Email: thomas.preusser@tu-dresden.de

***Abstract*—Integer addition is a pervasive operation in FPGA designs. The need for fast wide adders grows with the demand for large precisions as, for example, required for the implementation of IEEE-754 quadruple precision and eliptic-curve cryptography. The FPGA realization of fast and compact binary adders relies on hardware carry chains. These provide a natural implementation environment for the ripple-carry addition (RCA) scheme. As its latency grows linearly with the operand width, wide additions call for acceleration, which is quite reasonably achieved by addition schemes built from parallel RCA blocks. This study presents FPGA-specific arithmetic optimizations for the mapping of carry-select/increment adders targeting the hardware carry chains of modern FPGAs. Different trade-offs between latency and area are presented. The proposed architectures represent attractive alternatives to deeply pipelined RCA schemes.**

***Keywords*-FPGA; addition; carry-chain; carry-select; carry-increment**

I. Introduction

One of the most prevalent operations in digital arithmetic is the addition. It is part of virtually all implementations of more complex operators including the rather fundamental multiplication, the computation of scalar products or the calculation of vector magnitudes. It is present in unrolled formulations as well as in many iterative computation approaches.

FloPoCo [1] is a tool that is capable of generating VHDL[1] code for a wide variety of arithmetic operators. It provides a vast library of builtin operators, which may be used by themselves or may be combined to form complex custom data flows. The generator is able to attune the constructed implementation for a desired target operation frequency and draws from a great pool of knowledge to optimize the pipeline depth and the implementation size according to the user specification.

This paper describes a new implementation option for wide binary adders as implemented in FloPoCo. This implementation builds on the carry-select addition approach to accelerate the addition in comparison to the ripple-carry implementation, which is standard on FPGA devices. However, it features quite a few measures that optimize the mapping of the carry-select addition onto contemporary FPGA devices. These include: (a) an optimized computation of the inter-block carries, (b) the use of shorter comparators to compute the speculative block carries when the associated sum is not needed, and (c) the elimination of the high-fanout signal controlling the multiplexer for the final result selection.

After the description of the envisioned architectures, the generator strategies for frequency-optimized block splitting will be detailed. The resulting complexities in terms of LUT counts and the achievable timings will be derived and verified experimentally. The proposed architectures are also faced against pipelined RCA schemes in terms of LUT-count complexity. Pipelining options are discussed when high frequencies, unreachable by the combinatorial versions, are required. The final implementation of the generator will be integrated in the open-source framework offered by FloPoCo.

A. Background

1) FPGA: **F**ield **P**rogrammable **G**ate **A**rrays are circuits that are designed to be reconfigured after manufacturing. Generally, the device layout is composed of *logic blocks* that can be configured to implement any logical-function (function is tabulated into very small memories) and a *reconfigurable interconnect network* that connects these logic blocks.

A simplified view of the logic blocks present in Xilinx Virtex4 [2] and Virtex5 [3] FPGAs is presented in Figure 1. Among other components, it contains:

- a function-generator (**L**ook-**U**p **T**able):
  On Virtex4 the LUT has a capacity of 16 bits, being able to implement any 4-input logic function. The Virtex5 LUT with a capacity of 64 bits may either implement an arbitrary 6-input function output on `O6`, or it may be sliced into two 5-input functions sharing the same set of inputs. Then both outputs `O6` and `O5` are used.
- the fast carry-chain logic:
  FPGAs typically implement the binary word addition as a ripple-carry adder (RCA) in a way that one logic block assumes the operation of one full adder. The carries between these full adders are forwarded across the designated carry chains. The mapping of the full adder

[1]**V**ery-high-speed integrated circuit **H**ardware **D**escription **L**anguage

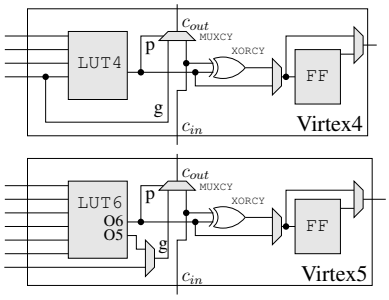

Fig. 1. The Basic *Logic Blocks* of Virtex4 and Virtex5

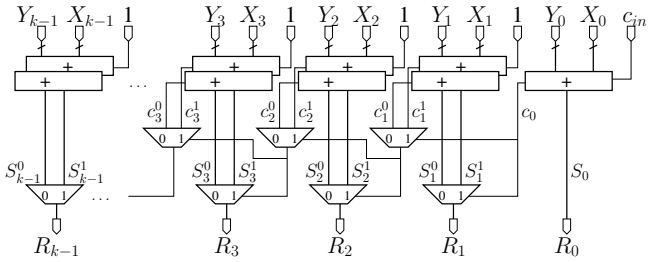

Fig. 2. Classic Carry-Select Architecture

on the logic blocks is performed as follows:

$$\begin{aligned} s &= a \oplus b \oplus c \\ &= p \oplus c_{\text{in}} && \text{– XORCY} \quad (1) \\ c_{\text{out}} &= ab + (a \oplus b)c_{\text{in}} \\ &= a\overline{(a \oplus b)} + (a \oplus b)c_{\text{in}} \\ &= \overline{p}a + pc_{\text{in}} && \text{– MUXCY} \quad (2) \end{aligned}$$

$$\text{where} \quad p = a \oplus b \qquad \text{– LUT} \qquad (3)$$

The *general routing* between logic blocks (inputs on the left and outputs on the right of Figure 1) is about 3 times slower than a LUT delay. The carry-propagation chain (running vertically from c_{in} to c_{out} in Figure 1) is much faster than the general routing, typically 10-15 times faster. Therefore, it is desirable to map computations to this carry-chain whenever possible.

2) Classic Carry-Select Adder: The classic carry-select adder [4] block consists of two ripple-carry adders and one multiplexer. Each pair of adders computes the two possible block results, one speculating on a carry-in of 0 and one on a carry-in of 1. The carry-in then feeds the select line of the multiplexer to choose the correct sub-sum and carry-out bit.

Large additions can be split into multiple carry-select adder blocks. The sub-sums are computed all in parallel. The carry-in ripples through the multiplexer network to propagate the correct carry-outs. Figure 2 presents the architecture of such an addition that is split into multiple carry-select blocks. For clarity, the block carry-out multiplexers have been separated from the block result multiplexers.

The multiplexer network is generally fast. However, if greater performance is needed, a costly but faster carry look-ahead structure can be used for carry-bit computation.

Unfortunately, the multiplexer network maps poorly on FPGAs. This is because in FPGAs the routing delay between the multiplexers(implemented in LUTs) exceeds by 3 to 4 times the LUT delay. Despite this major drawback, this naive mapping manages to outperform the highly FPGA-optimized RCA for large additions.

B. Related Work

An initial study evaluating the the performance of fast addition schemes is presented in [5]. The study leads to the conclusion that the only fast addition schemes mapping relatively well to FPGAs are carry-skip and the carry-select, the later providing the best performances. The optimizations applied to the classical carry-select architectures are structural, speculative carry-bit computations being addressed by carry-skip structures. The carry-in computation for each carry-select block is done using the classical multiplexer network, which is slow in FPGAs.

A discussion on the synthesis of carry-select adder in modern FPGAs is presented in [6]. The study proposes bitwise computation of the speculative sums using XOR gates and an inverters. The impact of these optimizations in modern FPGAs is little, if any. Compared to our work, the circuit delay for a 128-bit addition is 7.739ns (Altera StratixIII) whereas our is 2.5ns for a 300-bit addition (Xilinx Virtex5), providing that the performances of the two FPGAs are similar.

Another variation of the carry-select architecture is presented in [7]. It is based on the idea of time-multiplexing the same adder resource for computing the two speculative sums and carry-bits. The design manages to reduce the area at the expense of latency. Its implementation requires low-level directives for mapping the circuit to hardware, thus lacking portability. The results are presented for a maximum addition size of only 32bits which makes it impossible to compare against.

A better mapping of the carry-select architecture to the FPGA logic is presented in [8]. There, the k-level multiplexer network is mapped to a $2k$-bit RCA, significantly improving the adder timings. Unfortunately, the $2k$ size of this network affects the maximum number of carry-select blocks, reducing the maximum adder size manageable by this architecture for a fixed frequency.

The current study presents a novel mapping of the multiplexer network to the carry chain based on the work of Preußer et al. [9] on mapping general prefix computations to the carry-chain. The multiplexer network is mapped to one k-bit RCA and a carry-recovery circuit which, most of the time may be fused with other computations in modern FPGA. In addition, this study also provides structural improvements of the carry-select scheme based on specific FPGA feature of using the the faster and smaller comparator structures for speculative carry-bit computations instead of adders.

TABLE I
INTER-BLOCK CARRY PROPAGATION CASES

c_k^0	c_k^1	c_k	–	Case
0	0	0	–	**K**ill
0	1	c_{k-1}	–	**P**ropagate
1	0	$*$	–	*Impossible*
1	1	1	–	**G**enerate

II. FPGA-SPECIFIC MAPPING OF THE CARRY-SELECT ADDER

A. Acceleration of Inter-Block Carries

The inter-block carries of the carry-select adder take a shortcut through the multiplexer network skipping a complete block with a single multiplexer stage. This advantage is mostly given away if the multiplexers are implemented using standard LUTs connected through the general-purpose routing network. To compete with the fast carry propagation within a block, the inter-block carry propagation must also exploit the available carry-chain structures. This will be achieved by the technique described by Preußer and Spallek [9].

As shown in Table I, the different cases of the propagation of the inter-block carries can be easily distinguished by the values of the speculative block carry outputs. As c_k^0 implies c_k^1, the line $c_k^0\overline{c_k^1}$ can be neglected in the truth table. All others perfectly coincide with the carry propagation in a full adder so that the plain binary word addition of the bit vectors (c_k^0) and (c_k^1) produces the correct carry propagation.

Having an addition with the correct carries inside is of limited value if these cannot be accessed. While a direct tapping of the carry signals is, indeed, possible on the Virtex architectures, such a solution is not portable and would require the use of device-specific, low-level component primitives. A better alternative is offered through Equation 1, which allows to infer the incoming carry from the obtained sum bit s_k so that a standard addition operator suffices to implement the core carry-chain implementation:

$$\begin{aligned} c_{k-1} &= s_k \oplus p_k \\ &= s_k \oplus \overline{c_k^0} c_k^1 \end{aligned} \tag{4}$$

and hence (see also Table I):

$$\begin{aligned} c_k &= c_k^0 + c_{k-1}c_k^1 && \mid \text{by Eq. 4} \\ &= c_k^0 + \left(s_k \oplus \overline{c_k^0}c_k^1\right) c_k^1 \\ &= c_k^0 + \overline{s}\, c_k^1 \end{aligned} \tag{5}$$

The carry computation circuit with the resulting recovery of the carries from the sum bits is depicted in Figure 3. Note that the recovery computation can often be merged into the further processing of the recovered carry signal.

B. The AAM Carry-Select Architecture

The Add-Add-Multiplex (AAM) architecture derives directly from the classic carry-select architecture. The multiplexer chain computing the carry bits is replaced with the much faster carry-computation-circuit (CCC) and carry-recovery (CR) circuit. Figure 4 highlights the three stages of the AAM Carry-Select architecture:

1) For each block, two sums are computed, one for each possible value of the block carry-in. Both of these additions are extended to compute the block carry-out.
2) The two bit vectors formed by the block carries speculating on a carry-in of 0 and 1 are added in the CCC using a fast short ripple-carry adder. The output sum bits and their two respective speculative input carries are fed to the CR circuit, which recovers the proper block carry outputs.
3) The computed block carries are used to select the proper speculative block sum for the adder output.

The AAM adder uses a multiplexer to select among the two block sums. The multiplexer is a 3-input function, the two sum-bits and the carry-bit generated by the CR. For FPGAs with 5-input LUTs, the CR can be merged with the multiplexing. This is the case for modern FPGAs like Virtex5 and Virtex6 having 6-input LUTs. Having only 4-input LUTs available such as on Virtex4 devices, the CR introduces an extra LUT and a supplementary wire delay. On these architectures, adders with a low block count and, thus, a short CCC should prefer the carry-add-cell architecture described by de Dinechin et al. [8]. It uses extra intermediate propagating stages ($p = 1$), which provide direct access to the inverted propagated carry through Equation 1. As soon as the combined delay of these extra stages exceeds the delay of a CR, the AAM will become the superior choice also on these architectures.

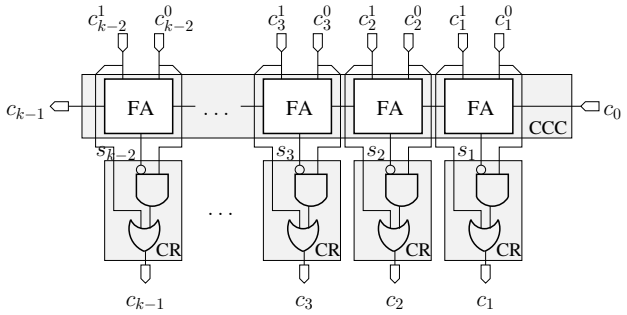

Fig. 3. Carry Computation Circuit with Carry Recovery

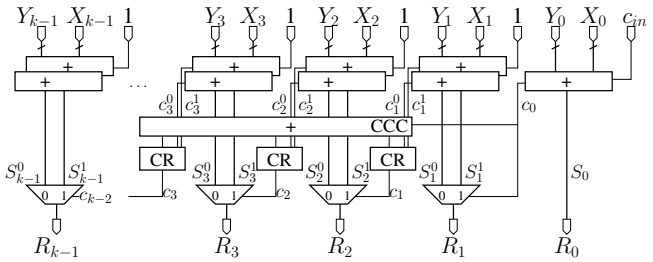

Fig. 4. The AAM Carry-Select Architecture

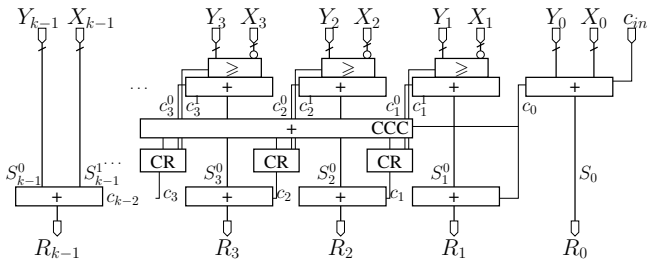

Fig. 5. The CAI Carry-Increment Architecture

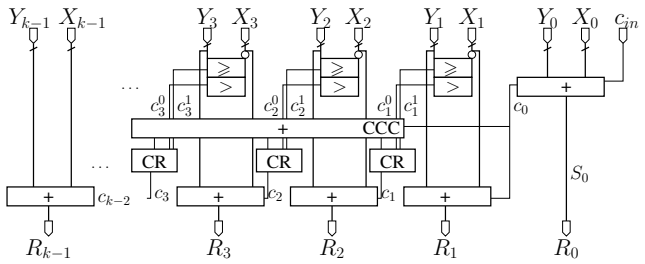

Fig. 6. The CCA Carry-Select Architecture

C. The CAI Carry-Increment Architecture

The Compare-Add-Increment (CAI) architecture adopts some features from the carry-increment adder, a widely adopted structural simplification of the carry-select scheme. In particular, the CAI only uses the block sums produced for the case of no incoming block carry. The final multiplexer stage is replaced by another adder, which adds the actual incoming carry and, thus, corrects the produced sum if necessary. Note that the choice of this incrementer instead of a multiplexer does not increase the number of occupied LUTs.

As the CAI does not need the sum speculating on an incoming block carry, the corresponding adder only serves the purpose of computing the associated carry-out of the speculative block sum $X_k + Y_k + 1$. This can, however, be obtained by the simple comparison:

$$c_k^1 \texttt{ <= '1' when } X_k \geq \texttt{not}(Y_k) \texttt{ else '0';} \quad (6)$$

All in all, the CAI offers the following improvements:

1) The use of a comparator for the computation of c_i^1 is, at most, as complex as the replaced addition. On Virtex5 and Virtex6 devices, the number of required LUTs is even cut in half as every stage on the carry chain processes two adjacent input positions rather than just one. This is possible as the sum bits are not asked for.
2) The number of registers required in a pipelined implementation is almost cut in half as only one of the two speculative block sums must be stored.
3) The wide fanout of the computed block carries for the control of the multiplexers is eliminated.

The resulting architecture is sketched in Figure 5. On FPGAs with 5-input LUTs, the CR is merged into the LSB computation of the final addition.

D. The CCA Carry-Select Architecture

The Compare-Compare-Add architecture takes the CAI architecture one step further. It uses two comparators to generate both c_i^1 and c_i^0.

$$c_k^0 \texttt{ <= '1' when } X_k > \texttt{not}(Y_k) \texttt{ else '0';} \quad (7)$$

The final step is turned from an incrementer into a complete adder computing $X_k + Y_k + c_k$.

The greatest benefit of this implementation is achieved on FPGAs with 5-input LUTs. Not only can the CR be merged into the LSB computation of the final addition but the whole critical path is shortened as the computation of both speculative block carries is only half as wide as a true adder. The architectures is outlined in Figure 6.

III. Frequency-directed adder design

Most FPGA designs have a clearly defined target operating frequency f. Assembling basic operators conceived for the same frequency ensures that: 1) the main design will run close to this frequency[2] and 2) the resource consumption will be minimal[3].

Our goal is to bring high performance adders to the open-source FloPoCo project whose main feature is assembling components built for the same target frequency. In order to comply with this interface, we need to design our architectures so that they are tuned for frequency f.

In an operator built for frequency f, all datapaths are smaller than $1/f = T$. For our architectures, the datapaths may contain operations such as: additions, comparisons, multiplexations and other general logic. Moreover, in the case of FPGAs one also has to account for the delays of the wires connecting these components (see Section I-A1 for a general information on the ratio between wire delays and component delays).

Components such as logic functions of up to 4-inputs on Virtex4 and up to 6-inputs on Virtex5 devices are implemented in LUTs. They have a *fixed* delay that we generically denote by δ_{LUT}. RCA and comparators (also implemented using the dedicated carry-chain) allow variable delays for inputs bit pairs, and have variable delays for their output bits.

In the RCA architecture the carry-bit ripples through, setting the correct value for the sum-bit and the carry-out bit at every step. It is natural that the result MSB is obtained later than the LSB. The availability of the sum-bits is given by the following equation:

$$\delta_{s_j} = \delta_{LUT} + j\delta_{carry} + \delta_{xor} \quad (8)$$

[2]It is impossible to guarantee that the the top designs runs at frequency f. As the main design gets more complex the pressure on the placement tool increases and makes it more difficult to find good placements, therefore introducing large net delays which impact the global frequency

[3]The design will not be over-pipelined, so no useless registers will be used

Moreover, we enforce that, for each bit j of the addition the incoming carry be synchronized with the computation of the propagate signal p (see Equation 1), which has a delay equal to δ_{LUT}. The availability of the carry-in bit at the j^{th} bit of the result is given by the formula:

$$\delta_{c_j} = \delta_{LUT} + j\delta_{carry} \tag{9}$$

The full-adder inputs may then arrive as late as:

$$\delta_{x_j} = j\delta_{carry} \tag{10}$$

The comparator has just one output. The delay of the output depends on the FPGA. On Virtex4 the delay of a k-bit the comparator is equal to that of a k-bit RCA. On Virtex5 the same comparator maps in half the LUTs. The delay is given by the equation:

$$\delta_{cmp}(k) = \delta_{LUT} + \lceil (k/2) \rceil \delta_{carry} + \delta_{xor} \tag{11}$$

The three proposed addition architectures follow the same philosophy: parallel computations of speculative carry-bits, fast-carry bit computation using the CCC and CR, final result computation using the recovered carry-bits at the outputs of CR. The data dependences between these stages together with the RCA implementation of the CCC give different computation scheduling strategies for the three architectures. The computation scheduling can then be directly translated to obtain the corresponding block sizes for a given frequency f.

A. Block-splitting strategies

We denote by L the addition size. Our objective is finding a length k vector of block sizes denoted by $(l_{k-1}...l_0)$, $L = \sum_0^{k-1} l_i$ such that the circuit delay does not exceed the target period T.

1) The AAM Carry-Select Architecture: Our objective is to schedule the inner and outer the computations of the carry-select blocks such that all computational datapaths are are smaller than T. The constraints given by the timing model will allow us to determine the block sizes. A visual indication of a tight computation scheduling for the AAM architecture is given in Figure 4.

Considering the imposed constraints, the CCC is a $k-2$-bit RCA having a delay of the MSB sum bit $\delta_{s_{k-2}}$ (Eq. 8). The sum-bit inputs the select line of the $k-1^{th}$ block multiplexer (Figure 4), having a delay δ_{MUX}.

On the other hand, as CCC is implemented as an RCA, it allows the inputs to be delayed at most as specified in Equation 10. As the speculative carries (c_i^1 and c_i^0) are also computed using RCAs, this allows the size of successive blocks to increase by exactly one bit.

We therefore choose to fix the 1^{st} block size, $l_1 = 1$bit. For a given frequency f, this sets the maximum value of k as:

$$\delta_{RCA}(1) + \delta_w + \delta_{RCA}(k-2) + \delta_w + \delta_{MUX} = T \tag{12}$$

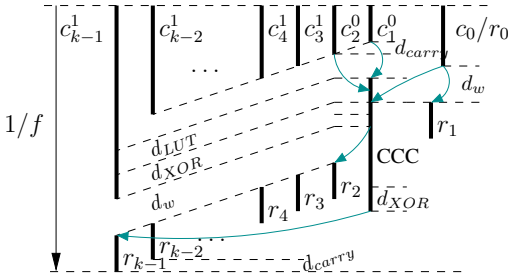

Fig. 7. Computation scheduling for the AAM architecture

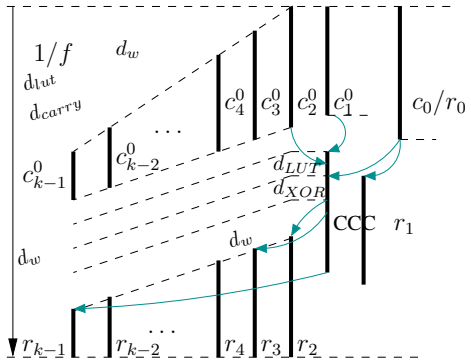

Fig. 8. Computation scheduling for the CAI architecture

As successive-block size increases by exactly one bit, the size of block, $l_{k-2} = k-2$. Blocks 1 to $k-2$ total of $(k-2)(k-1)/2$ bits.

The l_{k-1} and l_0 block sizes are the solutions of the equation:

$$\delta_{RCA}(l_{k-1}) = T - (\delta_w + \delta_{MUX}) \tag{13}$$

$$\delta_{RCA}(l_0) = \delta_{RCA}(l_1) + \delta_{LUT} \tag{14}$$

The maximal addition size for frequency f is $l_0 + (k-2)(k-1)/2 + l_{k-1}$. Figure 7 presents the computation scheduling for this architecture, together with the data dependences leading to determining block sizes.

2) The CAI Carry-Increment Architecture: The CAI architecture computes the speculative c_i^1 bit using Equation 6. On Virtex5 devices this comparison takes half the resources needed to obtain c_i^1 using a RCA. The latency improvement is given by the difference between Equation 8 and 11. However, this latency improvement is lost by using a RCA for computing c_i^0.

The third stage of the CAI architecture is an incrementation of the speculative sum for a 0 carry-in (S_i^0) with the carry-in obtained by the CCC. The incrementation is implemented as a RCA in FPGAs.

The output delays of the sum-bits of CCC are given in Equation 8. The difference between successive sum bits is δ_{carry}. The sum-bits are used as carry-in bits for the final stage adder. If we enforce that all the result bits be synchronized (Figure 8) this leads to successive blocks having a size decreased by 1-bit.

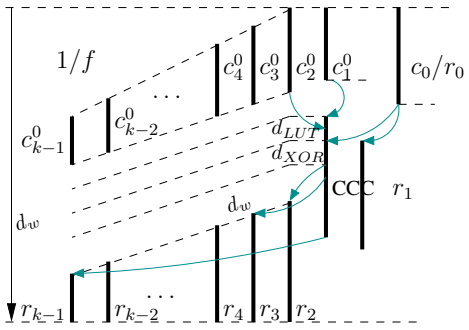

Fig. 9. Computation scheduling for the CCA architecture

We choose to fix the size of the $k-1^{th}$ block, $l_{k-1} = 1$ bit which leads to $l_2 = k-2$. Moreover, the difference in input delay between the speculative carry bits of l_2 and of l_1 for CCC is δ_{carry}. This leads to $l_1 = l_2 - 1 = k-3$.

Given the constraint that the carry-out of block 0 is the carry-in of CCC, the size of this block is the solution of the equation:

$$\delta_{RCA}(l_0) = \delta_{RCA}(l_1) + \delta_{LUT} \quad (15)$$

The maximal adder size for this architecture for frequency f is $(k-2)(k-1)/2 + k - 3 + l_0$.

3) The CCA Carry-Select Architecture: The CCA architecture uses comparators for computing the two speculative caries, c_i^0, c_i^1 (Equations 7,6). When compared to the CAI architecture, the latency of the first stage is reduced.

However, the block splitting strategy remains the same. The size of the first chunk is now the solution of the equation:

$$\delta_{cmp}(l_1) + 2\delta_w + \delta_{LUT} + \delta_{XOR} + \delta_{RCA}(l_2) = T \quad (16)$$

where $l_2 = k-2$.

The number of blocks (k) is now the solution of the equation:

$$\delta_{cmp}(l_2) + \delta_w + \delta_{LUT} + \delta_{XOR} + \delta_w + \delta_{RCA}(l_3) = T \quad (17)$$

The size of block 0 is:

$$\delta_{RCA}(l_0) = \delta_{cmp}(l_1) + \delta_{LUT} \quad (18)$$

B. Area complexity of the designs

Once the block-splitting procedure is finished, we can closely approximate the area of the circuit on the FPGA. The value is further used in the FloPoCo addition generator to choose among the proposed architectures and the pipelined architectures presented in [8]. The design taking fewer resources is chosen for implementation.

In this section we present the LUT-count formulas for the proposed architectures on a Virtex5. The same formals also hold for the new Virtex6 FPGA. Similar formulas can be derived for Virtex4 devices. These formulas are deduced based on the resources occupied by the basic blocks:

- 2:1 n-bit multiplexer occupies n LUTs.
- n-bit RCA takes n LUTs
- n-bit comparator takes $\lceil n/2 \rceil$ LUTs on Virtex5/6 and n LUTs on Virtex4.

1) The AAM Carry-Select Architecture:

$$\begin{aligned} LUTs &= \sum_0^{k-1} l_i + \sum_1^{k-1} l_i + k - 2 + \sum_1^{k-1} l_i \\ &= 3L - 2l_0 + (k-2) \end{aligned}$$

2) The CAI Carry-Increment Architecture:

$$\begin{aligned} LUTs &= \sum_0^{k-2} l_i + \sum_1^{k-2} \left\lceil \frac{l_i}{2} \right\rceil + k - 2 + \sum_1^{k-1} l_i \\ &\approx \frac{5}{2}L - \frac{3}{2}l_0 - \frac{3}{2}l_{k-1} + (k-2) \end{aligned}$$

3) The CCA Carry-Select Architecture:

$$\begin{aligned} LUTs &= l_0 + 2\sum_1^{k-2} \left\lceil \frac{l_i}{2} \right\rceil + k - 2 + \sum_1^{k-1} l_i \\ &\approx 2L - l_0 - l_{k-1} + (k-2) \end{aligned}$$

Although the block sizes $(l_{k-1}, ..., l_0)$ and the number of blocks k are different in the above formulas, we can still conclude on the order of magnitude of the implementations. Where implementable, the CCA outperforms CAI in implementation size due to the less costly comparators in Virtex5/6 devices. The AAM falls behind due to the approximately $3L$ area complexity caused by the RCAs.

Judging only on these results one could imagine that the CCA is the best architecture. However, as shown in the following, the range of L covered by AAM is much greater, so different trade-offs depending on the value of L have to be taken into account.

4) Comparison with pipelined-RCA schemes: The immediate advantages of the proposed addition architectures when compared to pipelined RCA architectures is the reduction of pipeline stages of the design. We are interested in the area cost we have to trade to get this advantage. Consequently, we have compared the area magnitude of our architectures to that of state-of-the-art pipelined RCA architectures presented in [8].

Table II synthesizes resource estimation formulas for Virtex5 FPGAs. Please note that the values of k and $(l_0, ..., l_{k-1}$ might be different for all these architectures, only the addition size L remains constant. The proposed addition architectures represent very attractive alternatives to the pipelined RCA schemes. For more than two pipeline levels the CCA architecture takes approximately as many resources as the pipelined schemes while at the same time reducing pipeline depth. For larger number of pipeline depths, the proposed architectures takes fewer resources, providing that it can match the frequency.

TABLE II
RESOURCE ESTIMATION FORMULAS FOR THE PROPOSED ARCHITECTURES AGAINST THOSE OF PIPELINED RCA SCHEMES PRESENTED IN [8]. THE TARGET FPGA IS VIRTEX5 AND THE ADDITION SIZE IS L

Architecture	LUT-FF pairs	Pipeline Depth
AAM	$3L - 2l_0 + (k-2)$	0
CAI	$\frac{5}{2}L - \frac{3}{2}l_0 - \frac{3}{2}l_{k-1} + (k-2)$	
CCA	$2L - l_0 - l_{k-1} + (k-2)$	
Classical [8]	$3L - l_0$	2
	$3L$	3
	$3L + l_0$	4
Alternative [8]	$3L - 2l_0$	2
	$3L - l_0$	3
	$3L$	4

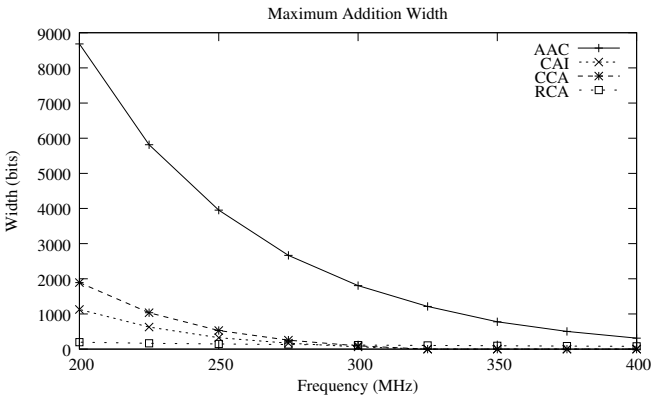

Fig. 10. Maximum addition sizes for the 3 architectures on a Virtex5 FPGA

5) Pipelining options: The architectures presented so far are all combinatorial. They allow reducing the number of pipeline stages by effectively replacing deeply pipelined RCA. However, for very large values of L the architectures are unable to reach the desired frequency. Pipelining them is a solution for these contexts. Inserting a pipeline stage in our architectures allows covering much larger addition sizes at the expense of 1 pipeline depth.

The AAM architecture can be effectively pipelined by inserting one register level after the speculative computations of the first stage. The architecture will be divided in two, having two critical paths: the block RCA addition on one side and the CCC RCA addition and the multiplexer delay on the other side. Inserting the register at this phase has no impact on the size of the architecture. The registers are combined with the LUTs (Figure 1) for free.

For the CAI architecture, the register level can be similarly inserted after the first computations. Although several registers are combined with LUTs, there is a small increase of $2l_{k-1}$ LUT Flip-Flop pairs for buffering the final block inputs. One solution to decrease this to l_{k-1} would be to apply the speculative sum computation for $c_{in} = 0$ to it. Inserting the register before the last computation phase requires in addition buffering the CCC outputs, therefore yielding a less attractive solution.

The CCA architecture can easily pipelined. The first two levels are regrouped to balance the size of the adders at the last level. Unfortunately, pipelining this architecture is expensive, having to pay an extra $2L - l_0$ LUT Flip-Flops pairs for it.

One should only consider the pipelined implementations when none of the combinatorial versions are capable of reaching the requested frequency. When deciding what pipelined architecture to use, one should first try the CAI architecture, and, if this one also fails, one should go with the pipelined AAM architecture.

IV. REALITY-CHECK

We have implemented a generic architectural generator for the proposed architectures. It inputs the addition width, target operating frequency and target FPGA (all Virtex FPGAs are currently supported) and generates a hardware description of the addition architecture in a portable, human-readable VHDL file. The timing parameters of the Virtex FPGAs have been obtained from the corresponding user manuals, and confirmed by synthesis results.

We were first interested in finding the maximum adder size or our proposed addition architectures for a fixed target frequency f. We focused on frequencies in the 200-400 MHz interval on a Xilinx Virtex5 FPGA. The maximal addition sizes for the proposed architectures and that of the classical RCA are presented in Figure 10. As expected, the latency optimized architectures, AAM and CCA outperform the CAI architecture.

The latency difference between the first stage of the CCA and CAI architectures (comparator vs. adder) translate into a maximum adder difference of more than 750-bits for $f = 200$MHz in favor of the CCA, more than 60% larger than the adder size supported by the CAI architecture. As the frequency increases, the block size decreases, minimizing the latency difference between the two architectures. The two architectures are unable to perform additions for $f > 300$MHz.

The AAM architecture is capable of managing much larger additions than the CAI or the CCA architectures. This can be explained by the relatively constant, and at the same time short delay of the third stage multiplexers. The architecture is capable of performing additions of over 8000 bits at a frequency of 200 MHz and over 300 bits for a frequency of 400 MHz.

Next we decided to compare AAM against an optimized pipelined implementation of the RCA (maximum unpipelined RCA adder size is 80 bits for $f = 400$ MHz). For a 300-bit adder implementation, the pipelined RCA implementation takes 3 pipeline levels and consumes 940 LUT Flip-Flop pairs against the AAM implementation that requires no pipeline levels and 952 LUT Flip-Flop pairs. The AAM implementation manages to reduce cycle count by 3 for a minor resource increase.

This result validates the theoretical complexities discussed in section III-B4, that in certain circumstances (large adder size, high target frequency) the proposed architectures provide real alternatives to the deeply pipelined RCA adders.

The area of an adder implementation plays an important role

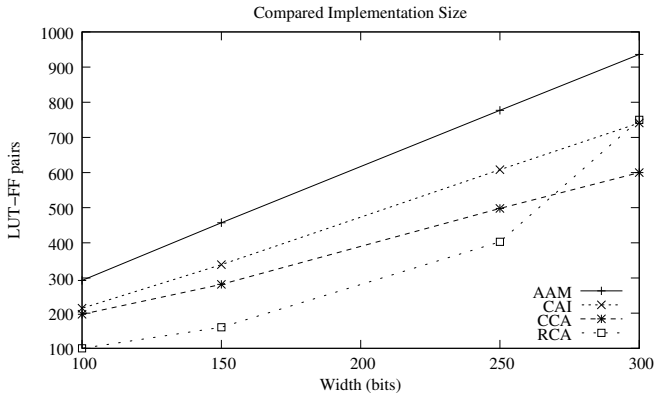

Fig. 11. The size of the 3 architectures and optimized pipelined RCA architectures for adders ranging from 100 to 300 bits, for a f=250 MHz on Virtex5 FPGA

in the decision process of using it in a wider context. Addition is a pervasive operation in FPGA designs, and therefore choosing the smallest adder implementation is desired. Figure 11 presents the implementation cost of the three architectures for adders ranging from 100 to 300 bits. The same figure also presents the area of the pipelined RCA adder. Out of the three proposed architecture, the CCA takes the smallest area, then CAI and finally, the largest one is the AAM. Both the CCA and CAI manage to obtain smaller implementations than the optimally pipelined RCA adder for an addition width of 300 bits.

V. Conclusion

This paper presents three efficient mappings of the carry select/increment adders on modern FPGAs. The core idea behind these mappings is mapping the multiplexer network computing the carry-bits on the dedicated fast-carry lines present in current FPGAs.

The first proposed architecture, the AAM is derived directly from the classic carry-select architecture. It benefits from the short latency of the stage-3 multiplexers to implement very large additions at high frequencies. The second architecture, the CAI, is a variation of the classical carry-increment scheme. It uses fewer resources than the AAM architecture due the use of comparators for speculative carry-bit computation for $c_{in} = 1$. The third stage uses an incrementer to fix the speculative sums computed for a $c_{in} = 0$. The third architecture, CCA, reduces the critical delay of the first stage path by using comparators for obtaining the speculative carries at the first stage. It uses fewer resources than the CAI architecture and it has as shorter critical path.

For these architectures, advanced block-splitting strategies are presented based on internal timings of adders and comparators, and accounting for the significant wire delays of FPGA circuits. Resource estimation formulas are also provided for Virtex5 devices in order to integrate the architectures in the FloPoCo adder generator.

The presented architectures are capable of replacing large, deeply pipelined RCAs. As large and as deeply pipelined the RCA, as small becomes the cost penalty. The gain with respect to the pipelined RCA is found in the severely reduced number of pipeline stages. This reduction might reduce synchronization cost between data-paths and therefore reduce the area of the top design.

These architectures have been implemented and are available in the FloPoCo open-source framework.

References

[1] F. de Dinechin, C. Klein, and B. Pasca, "Generating high-performance custom floating-point pipelines," in *Field Programmable Logic and Applications*. IEEE, Aug. 2009.

[2] *Virtex-4 FPGA User Guide*, Xilinx, 2008.

[3] *Virtex-5 FPGA User Guide*, Xilinx, 2009.

[4] M. D. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[5] S. Xing and W. W. Yu, "FPGA Adders: Performance Evaluation and Optimal Design," *IEEE Design and Test of Computers*, vol. 15, pp. 24–29, 1998.

[6] R. Yousuf and N. ud din, "Synthesis of carry select adder in 65 nm fpga," in *TENCON 2008 - 2008 IEEE Region 10 Conference*, 2008, pp. 1 –6.

[7] P. Devi, A. Girdher, and B. Singh, "Article:improved carry select adder with reduced area and low power consumption," *International Journal of Computer Applications*, vol. 3, no. 4, pp. 14–18, June 2010, published By Foundation of Computer Science.

[8] F. de Dinechin, H. D. Nguyen, and B. Pasca, "Pipelined FPGA adders," in *Field Programmable Logic and Applications*. IEEE, Aug. 2010.

[9] 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.