Fast and Area Efficient Adder for Wide Data in Recent Xilinx FPGAs

Petter Källström and Oscar Gustafsson

Fast and Area Efficient Adder for Wide Data in Recent Xilinx FPGAs

Petter Källström and Oscar Gustafsson
Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden.
Email: {petter.kallstrom, oscar.gustafsson}@liu.se

***Abstract*—Most modern FPGAs have very optimised carry logic for efficient implementations of ripple carry adders (RCA). Some FPGAs also have a six input look up table (LUT) per cell, whereof two inputs are used during normal addition. In this paper we present an architecture that compresses the carry chain length to $N/2$ in recent Xilinx FPGA, by utilising the LUTs better. This carry compression was implemented by letting some cells calculate the carry chain in two bits per cell, while some others calculate the summary output bits. In total the proposed design uses no more hardware than the normal adder. The result shows that the proposed adder is faster than a normal adder for word length larger than 64 bits in Virtex-6 FPGAs.**

I. Introduction

When performing additions and other operations requiring a carry chain, this chain is likely to be in the critical path when the word length increases. Most contemporary FPGAs have look up tables (LUTs) with more than the two inputs required for addition/subtraction. As a consequence the LUTs are not fully utilised when implementing these fundamental arithmetic operations. The main idea presented in this paper is to minimise the carry chain length by better utilisation of the LUTs. This method is demonstrated on Xilinx Virtex-6 FPGAs, and should work on all, Series-6, -7, and UltraScale FPGAs. Without the output pipeline requirement it also works on Virtex-5.

The basic Virtex-6 logic cell architecture is depicted in Fig. 1 [1]. When implementing a normal adder (using the + operator), the look up table LUT6 implements a propagate (p) signal, and take one input as a generate (g) bit via the `ax` wire. Those controls the carry chain and forms the sum output. A *slice* contains four logic cells, with a carry look ahead functionality [1], [2, p. 90]. Due to this, and the dedicated carry routing between slices, the carry chain is very fast, compared to the other logic [3]. As an illustration, evident from the results, an 88-bit normal adder takes only about twice as long time as an 8-bit adder.

Additions of numbers with very long word length are used in, e.g., elliptic cryptography [4], [5], [7] or the IEEE-754 binary128 precision format [6]. Those are such cases where the carry chain is likely to limit the clock frequency, despite the fast carry logic. One simple yet effective way to reduce the clock period constraint is to pipeline the adder, by inserting a register in the middle of the carry chain [7]. This, however, requires many extra pipeline registers before and after the adder, and will increase the latency.

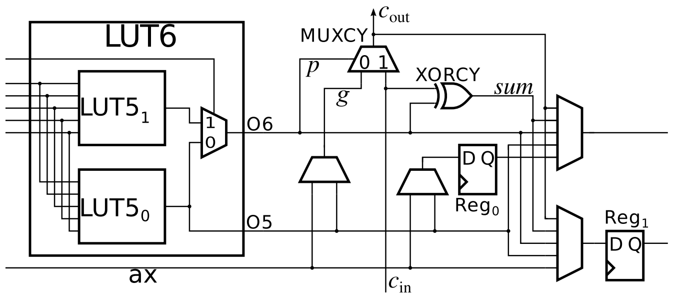

Fig. 1. Architecture in Spartan-6 and Virtex-6.

Different versions of carry acceleration methods, like carry select or carry bypass, have been adjusted and evaluated for FPGAs [5], [8]–[12]. They all have in common that the area (number of cells) increases rapidly, and they reduce the delay only for very wide operations (or not at all for investigated sizes). In [13], a method is proposed for the Virtex-5, where the adder is composed of three parts. The first part calculates the least significant half as a normal adder. The second part has an accelerated carry calculation of the least significant half, to generate the carry out of the least significant half. Finally, the third part calculates the most significant half as a normal adder, using the accelerated carry from the second part. In this way, the carry chain in the second part is shortened by 50%, leading to a total carry delay reduction of 25%. The cost is about 25% more hardware required for the second part.

II. Proposed Method

Similarly to [13], the proposed architecture is composed of three parts. For an N-bit adder, the architecture is depicted in Fig. 2, and explained as follows. The N input bits from each operand are split into K least significant bits (LSBs) and M most significant bits (MSBs), where $K + M = N$.

First, we have the carry compression `(cc)` cells, calculating every second carry bit. Each `(cc)` cell spans two "positions" along the carry chain, and so, the K bits are calculated using only $K/2$ cells. To archive this, each `(cc)` cell takes two a and two b bits, and calculates the combined propagate, p, and generate, g, signals as

$$p = (a_i \oplus b_i)(a_{i+1} \oplus b_{i+1}) \qquad (1)$$

$$g = a_{i+1}b_{i+1} + (a_{i+1} + b_{i+1})a_i. \qquad (2)$$

The p and g signals are mapped to the `O6` and `O5` LUT outputs, respectively, and used in the carry chain logic in the

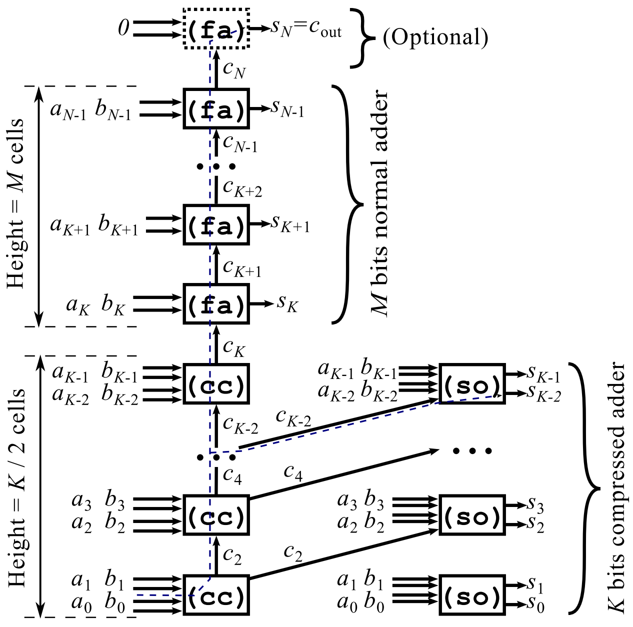

Fig. 2. Overview of the proposed carry compression design.

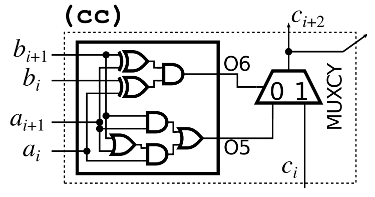

Fig. 3. Details of the `(cc)` cell.

cells, as depicted in Fig. 3. The generated carry bit is also connected to the ordinary unregistered output of the `(cc)` cell, as it is also needed to compute the sum.

For each of the $K/2$ `(cc)` cells, there is a sum out `(so)` cell, calculating two result bits per cell. The `(so)` cell gets the same two a and b bits as the corresponding `(cc)` cell, together with the carry bit from the previous `(cc)` cell. The LUT implements a function corresponding to a two-bit adder with carry in, generating the two desired sum bits. This is illustrated in Fig. 4.

The carry out from the last `(cc)` cell is fed as a carry in to the first of M full adder `(fa)` cells, calculating the M MSBs of the result. Those cells work exactly as a normal adder (that you get when using the + operator in, e.g., VHDL). That is, a `(fa)` gets one a and one b bit, and calculates the corresponding p and g signals, provided to the carry logic, as

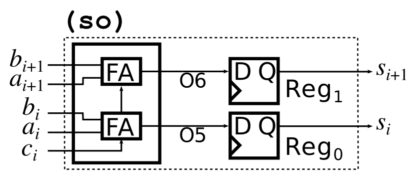

Fig. 4. Details of the `(so)` cell.

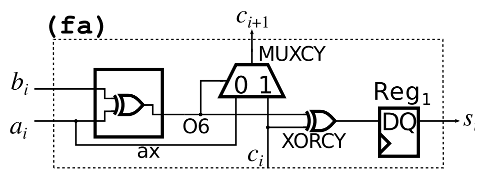

Fig. 5. Details of the `(fa)` cell.

TABLE I
NUMBER OF CELLS OF DIFFERENT TYPES.

Cell type	Number of cells
`(cc)`	$\frac{K}{2} = \frac{N-M}{2}$
`(so)`	$\frac{K}{2} = \frac{N-M}{2}$
`(fa)`	$M = N - K$ (optionally $M + 1$)[a]
Total	$K + M = N$ (optionally $N + 1$)

[a] Optionally one extra cell for carry out.

shown in Fig. 3. Optionally, an additional `(fa)` cell can be appended, to get the carry out bit.

The number of cells are summarized in Table I, using the relation $N = M + K$. As seen, there are N cells in total (excluding input registers), hence, no additional cells are required compared to a normal N-bit adder.

A. Critical Path of the Proposed Design

The proposed design has two candidate critical paths (illustrated with dashed lines in Fig. 2). Both start with the least significant input bits, and they end in the last `(so)` and `(fa)` cells respectively. For a given word length N, the path ending in the last `(so)` cell, denoted candidate path `so`, with a delay t_1, is reduced if M is increased (since K decreases). In the same time, the path ending in the last `(fa)` cell, candidate path `fa` with delay t_2, is increased (since the total carry length increases). The actual critical path is the longest (i.e. slowest) of the two candidates, and it is typically desired to be as short as possible.

To find the value of $M = M_{\text{opt}}$ where the critical path is minimum assume the following notation:

- t_{in}: delay from input register to carry chain (c_2, through first `cc`)
- t_{carry}: average delay from c_i to c_{i+2} in Fig. 3, or from c_i to c_{i+1} in Fig. 5
- t_{fa}: carry chain to `(fa)` register delay (c_i to Reg_1 in Fig. 5)
- t_{so}: carry chain to `(so)` register delay (c_i in Fig. 3 to Reg_1 in Fig. 4)
- t_1: total delay of candidate path `fa`
- t_2: total delay of candidate path `so`
- t_{cp}: total critical path delay.

Typically $t_{\text{so}} \gg t_{\text{fa}}$, due to the need of extra routing and logic. Then the delays can be linearly modelled as

$$t_1 = t_{\text{in}} + (K/2)\, t_{\text{carry}} + t_{\text{so}} \tag{3}$$

$$t_2 = t_{\text{in}} + (K/2 + M)\, t_{\text{carry}} + t_{\text{fa}} \tag{4}$$

$$t_{\text{cp}} = \max(t_1,\ t_2)\,, \tag{5}$$

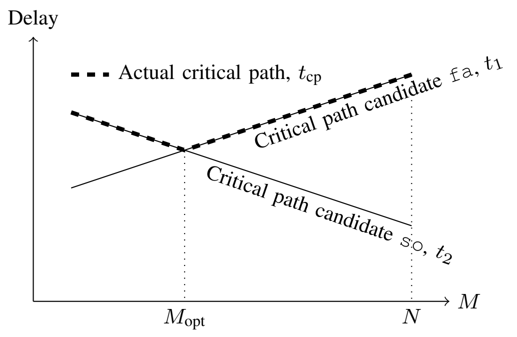

Fig. 6. Illustration of the two candidate critical paths and the actual critical path, for a fixed N and varying M.

and illustrated as in Fig. 6.

Rewriting (3) and (4) using $M + K = N$ gives

$$t_1 = t_{\text{in}} + \left(\frac{N-M}{2}\right) t_{\text{carry}} + t_{\text{so}} \tag{6}$$

$$t_2 = t_{\text{in}} + \left(\frac{N+M}{2}\right) t_{\text{carry}} + t_{\text{fa}}. \tag{7}$$

Assuming N is large enough, $t_1 = t_2$ when $M = M_{\text{opt}}$, we get

$$M_{\text{opt}} = \frac{t_{\text{so}} - t_{\text{fa}}}{t_{\text{carry}}}. \tag{8}$$

Note that M_{opt} is a ratio of the carry speed compared to other routing and logic in the FPGA. Especially, it is not dependent on N or K, and only depends on the FPGA speed grade, family etc.

In practice, the carry look-ahead will cause the timing to deviate slightly from the model, as supported by the result. Due to this, and the integer nature of K, M and N, it is unlikely to get exactly equal delays for the candidates. The linear model is nevertheless a good approximation.

There is an important exception from the previous discussion. When $M = N$ there are no `(cc)` or `(so)` cells, leading to only one critical path candidate, as the entire architecture turns out to be a normal adder (as generated by the + operator, with manual placement). We used this case as the reference adder.

III. Results

The proposed design was implemented on a Virtex-6 device (part no `xc6vlx130T-2-ff1156`). To measure the timing, registers where placed on inputs, and the reported maximum clock period was used. Hence, the results include potential clock skew. In order to reduce uncontrollable routing delays in the comparisons, everything was manually placed, according to the floorplan in Fig. 7. The test set up has two columns containing the proposed design. To the left of those, two columns of input register cells are placed. The $\texttt{R}_{\texttt{dual}}$ cells utilise both $\texttt{Reg}_0$ and $\texttt{Reg}_1$ in order to decrease the routing delay. The $\texttt{R}_{\texttt{single}}$ cells utilise only $\texttt{Reg}_1$.

To determine M_{opt}, the optimal M, the proposed design was synthesised for varying M and a number of different word

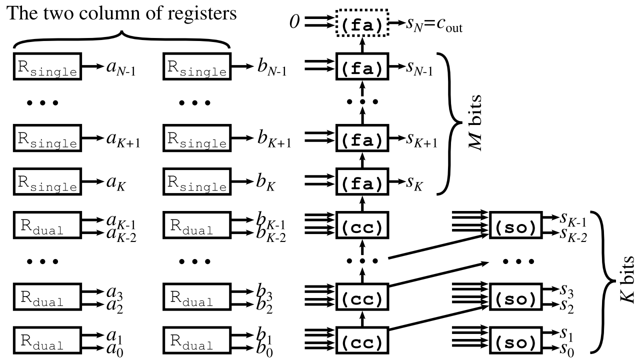

Fig. 7. Test set up floorplan of proposed architecture.

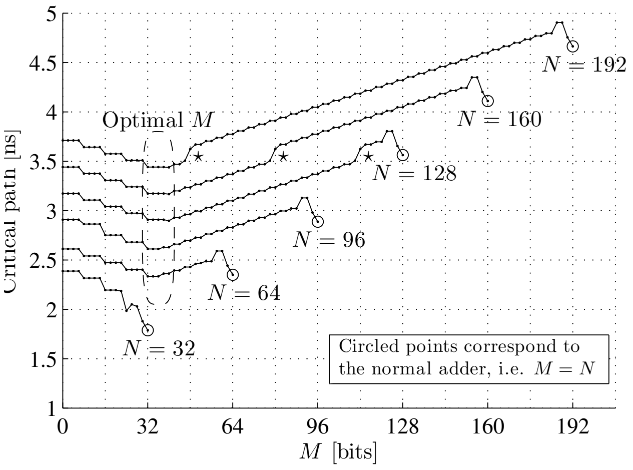

Fig. 8. Critical path for proposed design.

lengths. The results in Fig. 8 show that $M_{\text{opt}} \approx 36$ gives the fastest design for this device, independent of N. The carry look-ahead effects, and the relation to the linear model in Fig. 6 is also visible.

Figure 9 shows the timing of the normal adder and the proposed adder with $M = 36$, as a function of the word length. The cases $M = 0$ and $M = 56$ are also included to illustrate non-optimal choices of M.

For longer word length, the results have a "step" behaviour, marked with $\star$ in Figs. 8 and 9. This is caused by a sudden change in clock skew, evident from the synthesis reports, affecting the later cells in the design. Since it is related to the physical placement within the FPGA, the effect will probably act differently, or not at all, without manual placement.

The results show that for a small non-zero K the delay is longer than for $K = 0$. This can be seen by the increased delay to the left of the $M = N$ points in Fig. 8, and marked by $\diamond$ in Fig. 9. The cause is the multiplexer that the data from $\texttt{Reg}_0$ passes, as seen in Fig. 1, which adds a small delay. Therefore, some of the bits from the $\texttt{R}_{\texttt{dual}}$ cells have a longer delay than those from $\texttt{R}_{\texttt{single}}$. In practice, this means that the $\texttt{R}_{\texttt{dual}}$provides a slower t_{in} than the $\texttt{R}_{\texttt{single}}$does. In the normal adder set up, all data is stored in $\texttt{R}_{\texttt{single}}$ cells, and hence, the corresponding effect is not seen. If only the speed of the adder itself, from inputs to outputs, is of interest, the proposed

TABLE II
DELAY AND RESOURCE USAGE FOR THE PROPOSED METHOD AND THE METHOD IN [13].

	This work Virtex-6 (-2)						**Zicari [13]**† Virtex-5 (-2)					
	T_{prop}	T_{norm}		A_{prop}	A_{norm}		T_{prop}	T_{norm}		A_{prop}	A_{norm}	
N	ns	ns	**Ratio**	LUT	LUT	**Ratio**	ns	ns	**Ratio**	LUT	LUT	**Ratio**
32	N/A	1.788	N/A	32	32	**1.00**	1.92	2.07	**0.93**	41	31	**1.32**
48	2.195	2.084	**1.05**	48	48	**1.00**	2.12	2.32	**0.91**	61	48	**1.27**
64	2.334	2.350	**0.99**	64	64	**1.00**	2.52	2.79	**0.90**	81	65	**1.25**
96	2.611	2.888	**0.90**	96	96	**1.00**	3.30	3.65	**0.90**	121	97	**1.25**
128	2.907	3.562	**0.81**	128	128	**1.00**	(3.99)	(4.44)	(**0.90**)	(161)	(129)	(**1.25**)
192	3.441	4.663	**0.74**	192	192	**1.00**	(5.37)	(6.02)	(**0.89**)	(241)	(193)	(**1.25**)
256	4.080	5.763	**0.71**	256	256	**1.00**	(6.75)	(7.60)	(**0.89**)	(321)	(257)	(**1.25**)

† Extrapolated values for $N \geq 128$.

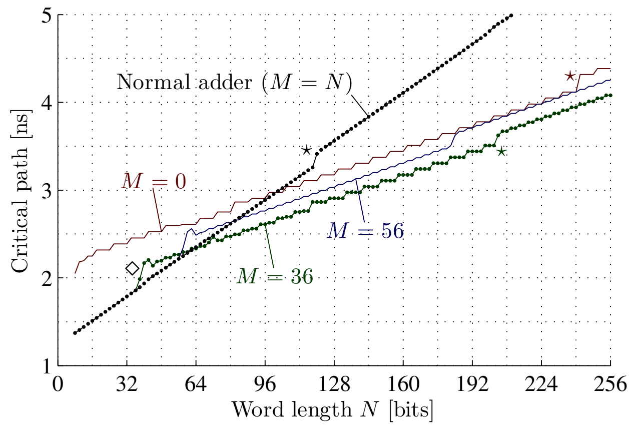

Fig. 9. Critical path for the normal and the proposed adder with $M = 36$.

design is effective for word lengths $N \geq M_{\text{opt}}$. However, for a registered adder, as in the test set up, the word length must be larger than M_{opt} in order for the proposed design to be effective, $N \geq 64$ in the used device.

Results are provided in Table II for the proposed method and the method in [13], compared with the normal adder. From the results of the proposed adder it can be seen that the relative savings in delay increase with the word length. For 256 bits, the delay is reduced 29% without any increase in resource usage. Theoretically, the ratio will approach 0.5 as the word length increases, since the first $K = N - M$ bits of the carry chain is an increasing part of the delay, and those bits are processed twice as fast by the `(cc)` cells.

For the method in [13], the clock period is reduced by around 10% for the considered word lengths, with an area increase of about 25%. As mentioned earlier, the reduction in the critical path will approach 25%, as $N \to \infty$. Note that the reported results in Table II for Zicari are obtained directly from [13] for $N \leq 96$ and extrapolated for larger values. As the results in [13] are for a different FPGA family, the ratio compared to a normal adder in the corresponding FPGA family is more relevant compared to the absolute delay values.

The pipelined adder is able to speed up long adders, on the cost of extra hardware. The pipelined adder proposed by de Dinechin [7] contains several partial adders. It should be possible to speed up those using our proposed carry compression techniques.

IV. CONCLUSIONS

We propose an adder structure for recent Xilinx FPGAs, that doubles the carry speed in parts of the adder. The total speed gain approaches 50% when the word length increases. To the best of our knowledge, this is the only published carry accelerating implementation that does not increase the area, compared to a normal adder.

REFERENCES

[1] *UG364: Virtex-6 FPGA Configurable Logic Block*, Xilinx Inc.
[2] B. Parhami, *Computer arithmetic*. Oxford university press, 1999, vol. 20, no. 00.
[3] *DS152: Virtex-6 FPGA Data Sheet: DC and Switching Characteristics*, Xilinx Inc.
[4] *Elliptic Curve Cryptography*, SEC Std. 1, 2009.
[5] H. D. Nguyen, B. Pasca, and T. B. Preusser, "FPGA-specific arithmetic optimizations of short-latency adders," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, 2011, pp. 232–237.
[6] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std. 754, Rev. 2008, Aug. 2008.
[7] F. de Dinechin, H. D. Nguyen, and B. Pasca, "Pipelined FPGA adders," in *Proc. Int. Conf. Field-Programmable Logic Applicat.*, 2010, pp. 422–427.
[8] H. M. Kamboh and S. A. Khan, "FPGA implementation of fast adder," in *Proc. Int. Conf. Comput. Convergence Technology*, 2012, pp. 1324–1327.
[9] S. Xing and W. W. H. Yu, "FPGA adders: Performance evaluation and optimal design," *IEEE Des. Test. Comput.*, vol. 15, no. 1, pp. 24–29, 1998.
[10] D. H. K. Hoe, C. Martinez, and S. J. Vundavalli, "Design and characterization of parallel prefix adders using FPGAs," in *Proc. Southeastern Symp. Syst. Theory.*, Mar. 2011, pp. 168–172.
[11] R. Priya and J. S. Kumar, "Implementation and comparison of effective area efficient architectures for CSLA," in *Proc. IEEE Int. Conf. Emerging Trends Comput. Commun. Nanotechnol.*, Mar. 2013, pp. 287–292.
[12] K. Vitoroulis and A. J. Al-Khalili, "Performance of parallel prefix adders implemented with FPGA technology," in *Proc. IEEE Northeast Workshop Circuits Syst.*, Aug. 2007, pp. 498–501.
[13] P. Zicari and S. Perri, "A fast carry chain adder for Virtex-5 FPGAs," in *Proc. IEEE Mediterranean Electrotech. Conf.*, 2010, pp. 304–308.