FPGA Adders: Performance Evaluation and Optimal Design

SHANZHEN XING
WILLIAM W.H. YU
University of Hong Kong

Delay models and cost analyses developed for ASIC technology are not useful in designing and implementing FPGA devices. The authors discuss costs and operational delays of fixed-point adders on Xilinx 4000 series devices and propose timing models and optimization schemes for carry-skip and carry-select adders.

KNOWING THE COSTS AND DELAY functions of fundamental building blocks enables designers to optimize costs and propagation delays of the larger units built from them. A fundamental building block of an arithmetic logic unit (ALU) is the binary adder. In this article, we examine the implementation of fixed-point adders on Xilinx 4000 series FPGA chips and cost and delay functions of various addition algorithms. On the basis of this study, we propose optimization schemes for the design of FPGA carry-skip and carry-select adders.

Adder cost and performance

Although many writers have discussed VLSI fixed-point addition techniques,[1-9] gate-count and gate-delay unit models in their studies are not useful for evaluating costs and performance of FPGA adders. In our study, we obtain operational times from Xilinx timing-simulation software instead of from the gate-delay models used for fixed VLSI designs. Instead of gate counts, we measure cost as the number of configurable logic blocks (CLBs) used. The performance-to-cost ratio is cost divided by operational time. In making comparisons, we rank techniques with larger performance-cost ratios as having better performance.

Carry-ripple adder. Adders differ in the ways their carries propagate. The most basic is the carry-ripple adder. The Xilinx 4000 series' dedicated carry logic designed for sequential carry propagation makes implementing n-bit carry-ripple adders easy. We have implemented carry-ripple adders ranging in length from 8 to 80 bits on different part types.

We have also implemented carry-complete and carry-look-ahead adders. By comparison with the ripple adder, their high costs, complexities, and high fan-in and fan-out requirements[3,4] make them unsuitable for implementation on FPGA devices.

The carry-ripple adder is a basic building block of other adders. The timing models we use in our optimization analyses of carry-skip and carry-select adders are functions of the carry-ripple adder's worst-case operational time.

Timing models. We partition an n-stage adder into x blocks. Each block has n/x stages. We define the timing models of block k, where $1 \le k \le x$, as follows:

- *Carry-ripple delay, $R(y_k)$.* The total delay of a carry entering block k, rippling through subsequent stages, and emerging from the block is

$$R(y_k) = \lambda_1 + \delta y_k \quad (1)$$

where y_k is the number of stages in block k, δ is the incremental delay of a single stage, and λ_1 is a constant.

0740-7475/98/$10.00 © 1998 IEEE

- *Carry-generate delay, $G(y_k)$.* The total delay of a carry generated at the first stage of the block, rippling through subsequent stages, and emerging from the block is

$$G(y_k) = \lambda_2 + \delta\,(y_k - 1) \qquad (2)$$

where λ_2 is a constant.
- *Carry-terminate delay, $T(y_k)$.* The total delay of a carry entering the block, rippling through subsequent stages, and terminating at the last stage is the same as the carry-generate delay; $T(y_k) = G(y_k)$.

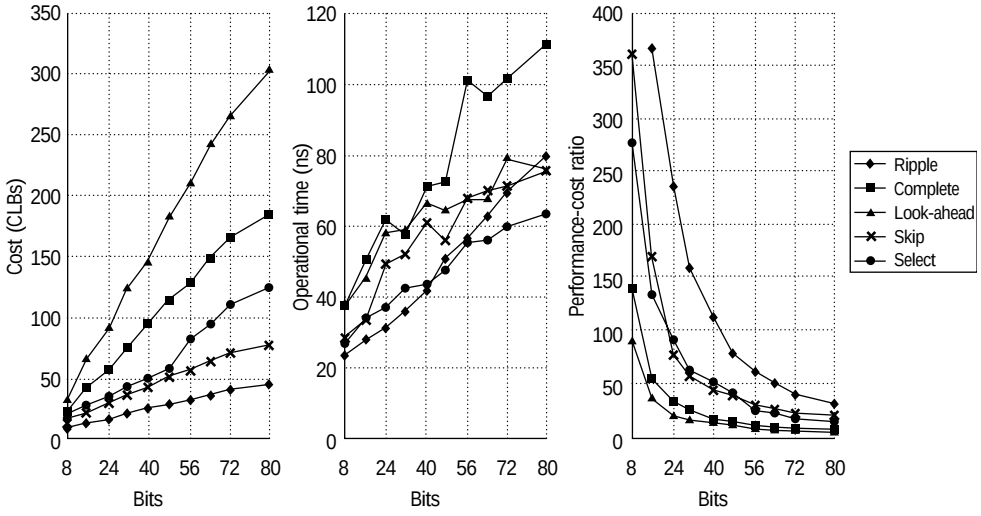

Figure 1. *Performance parameters of nonoptimized FPGA adders in comparison with carry-ripple adders.*

Carry-skip adders

Observing that a carry may skip any addition stages on certain addend and augend bit values, researchers developed the carry-skip technique to speed up addition in the carry-ripple adder. One can construct a carry-skip adder by partitioning a carry-ripple adder into blocks of the same or various sizes and adding carry-skip logic to each block. Carry-skip logic determines when a carry entering the block may skip directly to the next block. Using a multilevel structure, carry-skip logic determines whether a carry entering one block may skip the next group of blocks. Because multilevel skip logic introduces longer delays, it may be of little value beyond three or four levels even with fixed VLSI technology. Implemented on Xilinx 4000 devices, a carry takes much longer to propagate through multilevel carry-skip logic than through efficient, dedicated carry logic. Therefore, here we examine only single-level FPGA carry-skip adders.

Implementation of nonoptimized adders. The operational time of carry-skip adders greatly depends on their configurations. We investigated the implementation of various configurations of each adder of a given length and selected those that gave the best performance data. Figure 1 shows performance parameters of nonoptimized carry-skip adders of sizes from 8 to 80 bits. Our results show that the nonoptimized carry-skip adder performs no better than the carry-ripple adder, with a small increase in cost. However, it was worth investigating whether the optimized carry-skip adder would perform better than the carry-ripple adder.

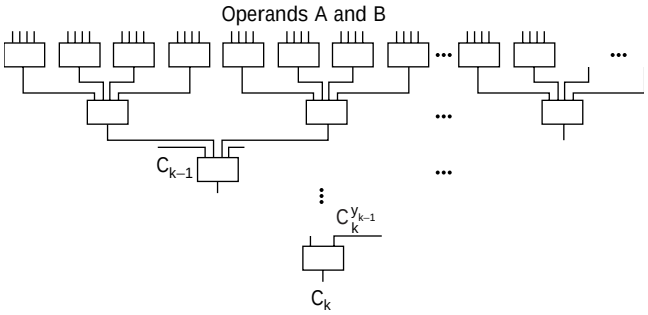

Figure 2. *Carry-skip logic structure.*

Optimization analysis. Many researchers have extensively studied optimization of carry-skip adders and have suggested many timing models for fixed VLSI technology.[2,6-8] As we have said, these models cannot be used for FPGA circuit analysis, which is based on CLB number and route delay. Therefore, we have developed the following formulation of a carry-skip timing model for optimization analysis.

Carry-skip delay. Factors such as carry-skip logic structure, carry-skip logic mapping, and CLB placement and routing at the implementation stage contribute to carry-skip delay. We assumed that the carry-skip logic structure is the dominant factor, and our implementation results later corroborated that assumption.

To use the CLB array structure effectively, we propose the general tree structure for a carry-skip logic block shown in Figure 2. Each rectangle represents a function generator. C_{k-1} and C_k are the carry-in and carry-out of block k, and $C_k^{y_{k-1}}$ is the carry-out produced by the block. The figure shows that

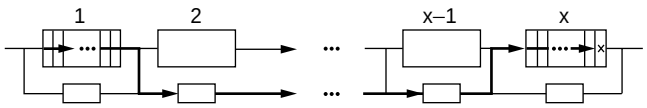

Figure 3. *Worst-case carry propagation path of carry-skip adders.*

the multilevel carry-skip logic has $2y_k + 2$ inputs. The first level has approximately $N_1 = (2y_k)/8$ CLBs, and level i has $N_i = N_{i-1}/8$ CLBs. Thus, the total number of CLBs needed to implement a y_k-bit carry-skip logic structure is $N = N_1 + N_2 + N_3 + \ldots \cong \lceil 1 + (5/16)y_k \rceil$.

Carry-skip delay includes constant look-up table delay, interconnect delay between function generators within CLBs, and interconnect delay between CLBs. Of these, interconnect delay between CLBs is dominant. Therefore, in calculating carry-skip delay, we consider only inter-CLB delay. On the other hand, in CMOS technologies, interconnect delay is linearly proportional to the square of the length of interconnect lines.[10] Therefore, the carry-skip delay expression is

$$S(y_k) = \lambda_3 + \beta l^2 \tag{3}$$

where λ_3 is a constant, β the coefficient of linearity, and l the effective length of the interconnect lines.

From Figure 2, we can assume l is approximately proportional to the number of carry-skip logic layers, and we express it as

$$l = \gamma \log_4(1 + 3N) = \gamma \log_4[4 + (15/16)y_k] \cong \gamma \log_4(4 + y_k) \tag{4}$$

Substituting Equation 4 into Equation 3 gives

$$S(y_k) = \lambda_3 + \alpha \log_4^2(4 + y_k) \tag{5}$$

where $\alpha = \beta\gamma^2$ is a constant coefficient, and λ_3 is the delay of carry-in and carry-out logic. Equation 5 shows that carry-skip logic implemented on an FPGA device is neither a constant nor a linear function as reported by other researchers.[2,6,7]

Configuration optimization. An n-bit carry-skip adder partitioned into x blocks has a configuration $Y = \{y_1, y_2, \ldots, y_{x-1}, y_x\}$, and $n = \sum_{k=1}^{x} y_k$. The optimization problem is to determine a configuration that gives the adder the minimum worst-case carry propagation (operational) time. Figure 3 shows the worst-case carry propagation path, which takes the carry generated at the adder's first stage the longest propagation time to reach the final stage. The carry generated at the first stage ripples through the first block, skips the subsequent blocks to the last block, and ripples through to the last stage. This worst-case propagation delay occurs when the operand pair are 010101…101 and 001010…011, and $C_{in} = 0$. The worst-case propagation time is the sum of the carry-generate delay of the first block, the skip-logic delays of the subsequent $(x - 2)$ blocks, and the carry-terminate delay of the final block:

$$D_w = G(y_1) + \sum_{k=2}^{x-1} S(y_k) + T(y_x) \tag{6}$$

Equation 6 implies the following criteria:

1. $x > 2$. Otherwise, the carry-skip adder has no advantage over the carry-ripple adder because there are no carry-skip operations.
2. $R(y_k) \geq S(y_k)$. The carry-ripple delay of any block is greater than or equal to the carry-skip delay.
3. $R(y_i) \leq \sum_{k=i}^{x-1} S(y_k) + T(y_x)$. The carry-ripple delay of any block is less than or equal to the delay of the carry to skip the block and subsequent blocks and ripple through the last block, terminating at the last stage.

Minimizing Equation 6 is equivalent to minimizing the sizes of the first and last blocks and the number of blocks. There are two simple steps to obtaining the optimal configuration. The first is to use criterion 2 to determine the last block's minimum size. The second is to use criterion 3 to determine the length of the other blocks recursively, starting from the second-to-last block until all n bits have been assigned.

Comparisons. Here we compare analytical and implemented performance improvements of the carry-skip adder using the proposed optimization scheme. Constants and coefficients in Equations 1, 2, and 5 differ slightly among different types of parts. We base our results on the Xilinx XC4010PQ160-5 chip.

Analytical comparisons. We derive the timing model parameters from the first-order approximation of implemented operational times shown in Figure 1. The parameters of Equations 1, 2, and 5 are $\lambda_1 = 13.5$, $\lambda_2 = 12.5$, $\lambda_3 = 11$, $\delta = 0.8$, and $\beta = 1.3$. Analytically, if the proposed adder is smaller than 54 bits, it will have no speed advantage over the carry-ripple adder. We obtained optimized configurations of adders from 56 to 112 bits. The operational times of the theoretical adders are 1% to 24% faster than carry-ripple adders.

Implementation comparisons. Figure 4 shows implementation results for optimized FPGA adders. The results show an improvement of from 0.6% to 16% for optimized adders of sizes from 64 to 112 bits. The implementation results show slightly less improvement than the analytical results but ac-

curately reflect theoretical predictions. The 56-bit adder has no speed advantage over the carry-ripple adder. Compared with the nonoptimized adders in Figure 1, the proposed adders have similar costs but shorter operational times.

Carry-select adders

Each block of a carry-select adder generates two sets of sums, one for the carry-in 0 and the other for 1. The carry-select logic selects the appropriate set of sums upon arrival of the carry bit.

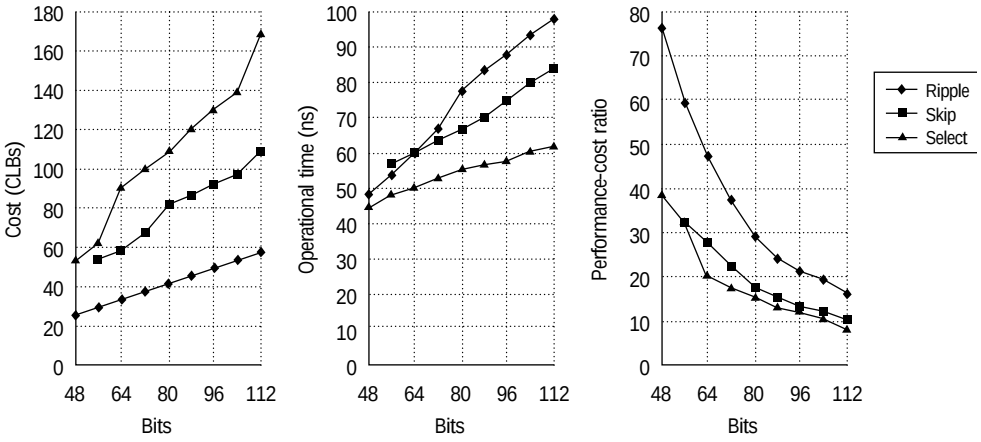

Figure 4. *Performance parameters of optimized FPGA adders in comparison with carry-ripple adders.*

Implementation of nonoptimized adders. Various block schemes and addition techniques can implement a carry-select adder. We investigated different combinations of block schemes and addition techniques before undertaking optimization analyses. Again see Figure 1 for costs, operational times, and performance-cost ratios of the best-performing set of nonoptimized adders. Propagation delays of nonoptimized carry-select adders are comparable to those of carry-ripple adders. Next we investigated whether optimized carry-select adders would have better operational times.

Optimization analysis. Our optimization studies of carry-skip adders led us to expect that configuration optimization would result in speed improvements. We propose three carry-select adder configurations and optimization schemes. The three configurations are the select-ripple-ripple (S-R-R), select-skip-ripple (S-S-R), and select-skip-skip (S-S-S) adders. The optimization schemes proposed here use the timing models given earlier for carry-ripple and carry-skip adders.

S-R-R adder. In each S-R-R adder block, two carry-ripple chains produce conditional sums and carry-outs. Each block's carry-select logic selects the appropriate sum and carry-out. Carry-select operation delay is a constant μ independent of block size. The problem of optimization is to determine the adder configuration $Y = \{y_1, y_2, \ldots, y_{x-1}, y_x\}$ for the minimum worst-case propagation time. The criterion for determining block sizes is that the carry-select signal must synchronize with the conditional sums and carry-outs. This criterion is

$$R(y_k) = (k-2)\,\mu + R(y_1) \quad (\text{for } k \geq 2) \tag{7}$$

Substituting Equation 1 into Equation 7 with $n = y_1 + \sum_{k=2}^{x} y_k$ gives

$$y_1 = 1/x[n - (\mu/\delta)] - (\mu/2\delta)x + 3\mu/2\delta \tag{8}$$

The worst-case propagation time is the ripple delay of block 1 plus the propagation time of the carry-select signal:

$$T = (x-1)\mu + R(y_1) = (1/x)(\delta n - \mu) + (\mu/2)x + [\lambda_1 + (\mu/2)]$$

We find the nearest integer to x, where $x = [2(\delta n - \mu)/\mu]^{1/2}$, by equating the derivative dT/dx to 0. This gives us the near-optimal adder block number. The optimal solution is $n \geq (3\mu)/\delta$ for $x \geq 2$.

S-S-R adder. We construct an S-S-R adder by adding carry-skip logic to each S-R-R block. Carry-select logic selects the conditional sums and carry-outs generated by the ripple chains within each block. Carry-skip logic determines when the carry entering the block may skip directly to the next block. The worst-case carry propagation path is the same as that shown in Figure 3. To benefit from the carry-skip technique and synchronize the arrivals of block carry-ins and conditional sums, we must meet the following criteria:

1. $x > 2$. This criterion is the same as for the carry-skip adder.
2. $S(y_k) \leq R(y_k) + \mu$. Skip delay must be less than or equal to the total of ripple delay and carry-select delay.
3. $T(y_x) \leq \sum_{k=2}^{x-1} S(y_k) + G(y_1)$. The last block's conditional sum generation time must synchronize with arrival of the carry-in.
4. $R(y_k) \leq T(y_x)$. Any block's ripple delay must be less than

or equal to the last block's conditional sum generation time.

Substituting $R(y_k)$ and $T(y_x)$ into criterion 4 gives $y_k \le y_x - \theta$, where $\theta = 1 - [(\lambda_1 - \lambda_2)/\delta]$. This means that the last block is the largest. The worst-case propagation time is

$$T = \sum_{k=2}^{x-1} S(y_k) + G(y_1) + \mu \quad (9)$$

Criterion 2 determines the first block size. Setting $y_k = y_x - \theta$ for $k = 2, 3, \ldots, x-1$, Equation 9 becomes

$$T = (x-2)\, S(y_k) + G(y_1) + \mu \quad (10)$$

where $y_k = (n - y_1 - \theta)/(x-1)$. Minimizing Equation 10 gives the optimal number of blocks and their sizes. (We use the quasi-Newton search method to find the minima.)

S-S-S adder. The S-S-S adder block configuration is the same as that of the S-S-R adder. We use an additional carry-skip network to examine the adder carry-in and block carry-outs and to determine all block carry-ins. The critical path is the same for both adders. The criteria for the S-S-S adder are the same as those for the S-S-R adder except for criterion 3. To synchronize the conditional sums generated by the last block and the carry-in by the carry-skip network, criterion 3 for the S-S-S adder becomes

$$T(y_x) \le S(n - y_1 - y_x) + G(y_1) \quad (13)$$

and the worst-case carry propagation time is

$$T = S(n - y_1 - y_x) + G(y_1) + \mu \quad (14)$$

The optimization process is the same as for the S-S-R adder.

Comparisons. Now we summarize and compare theoretical and implementation results for carry-select adders. For theoretical comparisons, parameters for Equations 1, 2, and 5 are the same as those given for carry-skip adders: $\lambda_1 = 13.5$, $\lambda_2 = 12.5$, $\lambda_3 = 11$, $\delta = 0.8$, and $\beta = 1.3$. The carry-select logic delay μ is 12 ns for the XC4010PQ160-5 device.

Analytical comparisons. Optimization analyses show that S-R-R and S-S-R adders smaller than 48 bits, as well as S-S-S adders smaller than 56 bits, have no speed advantage over carry-ripple adders. Optimized theoretical S-R-R, S-S-R, and S-S-S adders are 13% to 39%, 15% to 36%, and 7% to 43% faster than carry-ripple adders.

Implementation comparisons. Operation speeds of the three optimized adders larger than 48 bits are very similar. The S-R-R adder is the most economical to implement, at a cost about 50% less than the other two adders. The speed improvement of the S-R-R adder over the carry-ripple adder is 7% to 36%.

OUR STUDY REVEALS that performance parameters of a specific addition technique implemented in different FPGA part types differ slightly. In general, adders implemented on lower-density parts have slightly shorter operational times than adders on higher-density parts, but their costs are almost the same.

The results in Figures 1 and 4 show that the carry-ripple adder has the lowest cost and highest performance-cost ratio because of its highly regular structure and its effective use of the CLB's dedicated carry logic. Therefore, it is preferable where simplicity and cost are critical factors. The carry-complete and carry-look-ahead adders are the least suitable for implementation on FPGA devices due to their high costs, irregular structures, and inability to use the dedicated carry logic.

The optimized carry-skip adder is second lowest in costs and second best in performance-cost ratios. However, the operational time of an optimized carry-skip adder smaller than 56 bits compares less favorably to that of the carry-ripple adder. Thus, the carry-skip adder is not the best choice for designs using smaller adder units.

The optimized S-R-R adder has the lowest cost of the three carry-select adders and hence the best performance-cost ratio. For implementation of adders larger than 48 bits, the optimized S-R-R adder is the most appropriate choice. When it is longer than 48 bits, it has the best operational time at a reasonable cost increase over carry-ripple adders. Although it is not cheaper to implement than the carry-skip adder, the technique does have the advantages of regular structures and almost the same performance-cost ratio as the carry-skip adder.

Our results also show that the timing models proposed here are valid and the optimization schemes are effective. This article can serve as a useful reference for designing FPGA adders. Designers can easily extend these schemes to FPGA devices other than the Xilinx 4000s, provided the devices have similar dedicated carry logic and structure. D&T

References

1. A.R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementations*, Prentice-Hall, Hertfordshire, UK, 1994.
2. S. Majerski, "On Determination of Optimal Distributions of Carry Skip in Adders," *IEEE Trans. Electronic Computers*, Vol. EC-16, No. 1, 1967, pp. 45-58.

3. D. Salomon, "A Design for an Efficient NOR-Gate-Only, Binary-Ripple Adder with Carry-Completion-Detection Logic," *Computer J.*, Vol. 30, No. 3, 1987, pp. 283-285.
4. R.W. Doran, "Variants of an Improved Carry-Lookahead Adder," *IEEE Trans. Computers*, Vol. C-37, No. 9, Sept. 1988, pp. 1110-1113.
5. B.W.Y. Wei and C.D. Thompson, "Area-Time Optimal Adder Design," *IEEE Trans. Computers*, Vol. 39, No. 5, May 1990, pp. 666-675.
6. A. Guyot, B. Hochet, and J.M. Muller, "A Way to Build Efficient Carry-Skip Adders," *IEEE Trans. Computers*, Vol. C-36, No. 10, Oct. 1987, pp. 1144-1152.
7. P.K. Chan and M.D.F. Schlag, "Analysis and Design of CMOS Manchester Adders with Variable Carry-Skip," *IEEE Trans. Computers*, Vol. 39, No. 8, Aug. 1990, pp. 983-992.
8. P.K. Chan et al., "Delay Optimization of Carry-Skip Adders and Block Carry-Lookahead Adders Using Multidimensional Dynamic Programming," *IEEE Trans. Computers*, Vol. 41, No. 8, Aug. 1992, pp. 920-930.
9. A. Tyagi, "A Reduced-Area Scheme for Carry-Select Adders," *IEEE Trans. Computers*, Vol. 42, No. 10, Oct. 1993, pp. 1163-1170.
10. N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A Systems Perspective*, Addison-Wesley, Reading, Mass., 1985.

Shanzhen Xing is working toward his PhD degree in the Department of Industrial and Manufacturing Systems Engineering, University of Hong Kong. His research involves computer arithmetic and reconfigurable computing systems.

William W.H. Yu is a lecturer in the Department of Industrial and Manufacturing Systems Engineering, University of Hong Kong. Previously, he was an associate professor in the Department of Electronic Engineering, Chung Yuan University, Taiwan, where he taught for 12 years. His research interests include artificial neural networks and reconfigurable computing systems. Yu is a member of the IEEE.

Send questions or comments about this article to William W.H. Yu, Dept. of Industrial and Manufacturing Systems Engineering, University of Hong Kong, Pokfulam Road, Hong Kong; wwhyu@hkucc.hku.hk.