Performance Evaluation of FPGA Implementations of High-Speed Addition Algorithms

William W.H. Yu and Shanzhen Xing

Department of Industrial & Manufacturing Systems Engineering
The University of Hong Kong, Pokfulam Road, Hong Kong

ABSTRACT

Driven by the excellent properties of FPGA's and the need for high-performance and flexible computing machines, interest in FPGA-based computing machines has increased dramatically. Fixed-point adders are essential building blocks of any computing systems. In this work, various high-speed addition algorithms are implemented in FPGA's devices, and their performance is evaluated with the objective of finding and developing the most appropriate addition algorithms for implementing in FPGA's, and laying the ground-work for evaluating and constructing FPGA-based computing machines. The results demonstrate that the performance of adders built with the FPGA's dedicated carry logic combined with some other addition algorithms will be greatly improved, especially for larger adders.

Keywords: FPGA, addition, performance evaluation, carry-ripple adder, carry-completion adder, carry-lookahead adder, carry-skip adder, carry-select adder

1. INTRODUCTION

Recent studies[1-9] have demonstrated that the reconfigurable computing systems indeed have the feasibility and potential for improving the performance of a system by modifying its hardware or architecture by the software in real time to match the computational characteristics of the individual application. As the densities and speeds of the SRAM-based FPGA's (Field Programmable Gate Arrays) continue to increase, FPGA-based reconfigurable and custom computing systems have become one of the hottest research topics in computer science and computer engineering. Fixed-point adders are essential building blocks in any arithmetic units in a computing system. Their performances or speeds of operation depend on the carry propagation delay. In order to reduce the worst-case carry propagation times, various high-speed fixed-point addition algorithms have been studied extensively in the area of designing fixed VLSI processors[11-23]. However, an addition algorithm optimally implemented in one technology may not necessarily be so in a different technology. The elemental building blocks are gates in fixed VLSI technology and CLB's (Configurable Logic Blocks) in FPGA devices. The implementation techniques of addition algorithms in the two technologies are different and thus their performance and cost parameters. Therefore, this work undertakes the performance evaluation of various available addition algorithms implemented in Xilinx 4000 series devices in an effort to determine their suitability to FPGA's. The paper aims to lay some ground-work for evaluating and constructing FPGA-based reconfigurable computing systems.

2. BASIS OF PERFORMANCE EVALUATION

In fixed VLSI technology, the assessment of different addition algorithms is usually based on the conventional analytic approach. The gate-count model is used for area/cost evaluation and the gate-delay units model for operational time evaluation. However, with FPGA's technology, the gate numbers and the gate delay units serve no useful purposes in performance and cost evaluations, because the basic functional units in FPGA's are CLB's rather than the basic logic gates in the fixed VLSI.

In order to evaluate FPGA's implementations of different addition techniques, the evaluation should more appropriately be based on the features of FPGA's. Each FPGA device[10] includes three major reconfigurable elements: configurable logic blocks (CLB's), I/O blocks (IOB's), and interconnections. These elements all contribute to the propagation delay of the processing unit implemented in the FPGA devices. The IOB's provide the interface between the internal and external signals. The programmable interconnect resources connect the inputs and outputs of the CLB's and IOB's into an appropriate

0-8194-2316-5/96/$6.00

network. Any XC4000 series CLB is capable of implementing up to two four-variable or one nine-variable logic functions. The logic functions in a CLB is accomplished in a table-look-up operation. Any logic functions of more than nine variables require two or more CLB's. Generous on-chip buffering makes block delays insensitive to loading by the interconnect structure, though all interconnect delays are layout-dependent. The total propagation delay depends of the amount of resources used.

To evaluate the performances of the different addition algorithms, the parameters: operation time (T), cost / area (C) and performance:cost ratio (η) will be examined. Obviously, the cost/area (C) of a design implemented in FPGA's should be calculated in terms of the number of required CLB's. The operational time (T) is obtained from the timing simulation results with Xilinx software which uses the actual block and routing delay times from the routed design. The simulation results allow a much more accurate assessments of the behaviors of the implementations under worst-case conditions to be made. The performance:cost ratio (η) is defined as the reciprocal of the cost multiplied by the operational time in this work. Therefore, the comparison of FPGA's implementations of different addition techniques can be based on the value of η. If one technique has a larger value of η, it will be considered to be better than another.

3. EXISTING ADDITION ALGORITHMS AND THEIR FPGA IMPLEMENTATIONS

Based on the way carries are propagated, the classical high-speed fixed-point addition algorithms mainly include carry-ripple, carry-completion, carry-skip, carry-lookahead, and carry-select addition algorithms[11-14]. In order to reduce the cost/area and the carry-propagation time or both of them, numerous variations of these classical approaches have been studied[15-23] and newer ones are still being developed and implemented with the fixed VLSI technology.

The different addition algorithms have been implemented with different part-types of the widely used Xilinx 4000 series devices. This paper reports the performance evaluation for the carry-ripple, carry-completion, carry-lookahead, carry-skip and carry-select adders.

3.1 Carry-ripple adder

The carry-ripple adder is one of the oldest and simplest adder designs. The *n*-bit carry-ripple adder is easily implemented using the dedicated carry logic (Figure 1) which is one of the excellent features of Xilinx 4000 series devices. The carry logic circuit is independent of the function generators, but shares some of the same input with the function generators. Each CLB can implement approximately 40 different functions and carry modes.

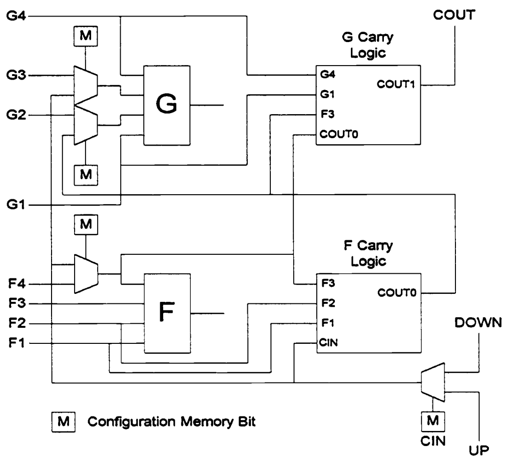


Figure 1. The dedicated carry logic in each CLB

Table 1 shows the performance parameters of the carry-ripple adders of sizes from 8 to 80 bits which are implemented in different Xilinx 4000 series part-types.

Table 1. Implementation data of carry-ripple adders

Adder Width		8	16	24	32	40	48	56	64	72	80
4003	C	6	10	14	18	22	26	30	34	38	*
	T	19.4	27.7	29.1	35.1	41.4	49.3	59.6	61.1	67.3	*
	η	859	361	245	158	110	78	56	48	39	*
4005	C	6	10	14	18	22	26	30	34	38	42
	T	20.4	25	28.7	35	41.1	47.9	55.2	61.2	71.5	80.1
	η	817	400	249	159	111	80	60	48	37	23
4008	C	6	10	14	18	22	26	30	34	38	42
	T	21.7	25.5	30.8	34.4	40.7	49.7	52.2	61.7	67.7	76.1
	η	768	392	232	162	112	77	64	48	39	31
4010	C	6	10	14	18	22	26	30	34	38	42
	T	22.6	27.3	30.4	35.2	41	49.9	55.9	61.9	68.5	79.3
	η	738	366	235	158	111	77	60	48	38	30
4013	C	6	10	14	18	22	26	30	34	38	42
	T	25	26.8	31.5	37.8	41.7	52.4	58.4	64.4	71.3	79.7
	η	667	373	227	147	109	73	57	46	37	30

Note: * – can not be implemented in one device.

3.2 Carry-completion adder

The carry-completion adder is obtained by modifying the carry-ripple adder to include the carry-completion detection logic. Because this adder is asynchronous, the operational time of this adder varies according to the operands although the worst-case operational time of this adder can still be linearly proportional to the length n of the adder. Table 2 shows the performance parameters for carry-completion adders. In order to compare this algorithm with others, the average operational times are taken rather than the worst-cast operational times of the adders.

Table 2. Implementation data of carry-completion adders

Adder Width		8	16	24	32	40	48	56	64	72	80
4003	C	20	37	56	71	90	*	*	*	*	*
	T	33.1	48.8	59.4	60.2	70	*	*	*	*	*
	η	151	55	30	23	16	*	*	*	*	*
4005	C	20	37	56	74	91	110	127	144	161	180
	T	37	49.6	57	61.7	74.2	71.9	80.1	84.6	87.8	104.1
	η	135	54	31	22	15	13	10	8	7	5
4008	C	20	38	56	74	93	111	130	145	165	183
	T	34.1	49.5	57.2	60.5	75.5	80.6	99	86.6	100.4	103.2
	η	147	53	31	22	14	11	8	8	6	5
4010	C	20	38	53	73	91	111	126	146	163	182
	T	36.5	49.6	60.8	57	70.7	71.5	100.4	96	101.4	110.5
	η	137	53	31	24	16	13	8	7	6	5
4013	C	20	38	53	74	90	104	129	147	165	185
	T	36.9	52.2	63.4	67.7	70.6	82.7	89	96.2	98.3	125.6
	η	136	50	30	20	16	12	9	7	6	4

Note: * – can not be implemented in one device.

3.3 Carry-lookahead (CLA) adder

Theoretically, fundamental CLA adders can be constructed and always results in a constant addition time independent of the width of the adder if the CLA unit can be freely expanded. Due to the rapid increase in the fan-out and fan-in required to implement the carry generation and the carry propagation functions as the adder size increases. Such designs are not practical but for the smallest adders. Therefore, large adders are generally implemented modifying the fundamental approach with a multilevel structure or combining CLA algorithms with some others to reduce the fan-in and fan-out difficulties. These approaches will usually result in additional delay, and the operational time of the adder will be no longer a constant. The cost and performance data of FPGA implementations of multilevel CLA adders are shown in Table 3.

Table 3. Implementation data of multilevel CLA adders

Adder Width		8	16	24	32	40	48	56	64	72	80
4003	C	30	63	90	100	*	*	*	*	*	*
	T	36.8	44.1	52.4	59.1	*	*	*	*	*	*
	η	91	36	21	17	*	*	*	*	*	*
4005	C	30	64	89	121	144	182	*	*	*	*
	T	37.5	41	55.2	60.6	61.2	63.8	*	*	*	*
	η	89	38	20	14	11	9	*	*	*	*
4008	C	30	64	90	121	144	180	209	239	270	306
	T	36.4	46.6	53.4	61.5	62	66.9	66.9	68	70.8	75.1
	η	92	34	21	13	11	8	7	6	5	4
4010	C	30	64	90	121	144	182	210	240	266	303
	T	37.6	44.5	57.6	58	65.7	64.2	67.6	66.8	79	74.9
	η	89	35	19	14	11	9	7	6	5	4
4013	C	30	64	90	120	144	180	210	235	256	306
	T	39.4	44.3	60.1	61.2	66	71.8	71.5	70.9	80.6	78.1
	η	85	35	18	14	11	8	7	6	5	4

Note: * – can not be implemented in one device.

Table 4. Implementation data of carry-skip adders

Adder Width		8	16	24	32	40	48	56	64	72	80
4003	C	11	17	26	33	40	49	54	60	68	*
	T	27.7	29.8	45.9	55.4	60.6	67.3	66.7	69.1	68.2	*
	η	328	197	84	55	41	30	28	24	22	*
4005	C	11	18	27	34	40	49	52	60	67	74
	T	27.3	31	49.7	53.4	59.2	63.1	64.5	69.5	68.7	78.4
	η	333	179	75	55	42	32	30	24	22	17
4008	C	11	18	28	34	39	47	53	60	68	74
	T	27.5	33.4	51.4	51.3	54.9	53.4	64.8	67.9	71.2	69.9
	η	331	166	69	57	47	40	29	25	21	19
4010	C	10	18	27	34	40	48	53	60	68	74
	T	27.7	32.6	48.6	51.4	60.1	55.3	67.8	69.5	70.8	75
	η	361	170	76	57	42	38	28	24	21	18
4013	C	10	18	27	35	40	49	53	60	68	74
	T	27	32.1	54.7	55.2	62	59.9	69.8	70	73.6	78.9
	η	370	173	68	52	40	34	27	24	20	17

Note: * – can not be implemented in one device.

3.4 Carry-skip adder

The carry-skip adder is built from the carry-ripple adder. An *n*-bit ripple adder is partitioned into blocks and carry-skip logic is added to each block. The worst-case carry propagation delay in carry-skip adders highly depends on the configurations of such adders. In this work, only one-level carry-skip adders are implemented. This is because the dedicated carry logic is so efficient that there is likely to be little value beyond two or more skip levels for the adders as the carry-skip delay will increase quickly with the number of skip levels. Table 4 shows the best implementation data of several implementations with different configurations for each given width adder in terms of time and performance:cost ratio. However, these results do not necessarily represent the operation times of the optimally designed adders. It is believed that the performance of such adders should be much better if the configuration optimization is considered.

3.5 Carry-select adder

The carry-select adder was considered as a compromise between the carry-ripple and the CLA adders. In fact, there can be many variations of carry-select adders due to the addition algorithms used to generate the conditional sums within the blocks and to determine the carry-out of each block. In this work, carry-ripple technique is used for both the carry determination logic and the generations of the conditional sums. Like that of carry-skip adders, the implementation data used to compare with other algorithms are also chosen from several implementations with different configurations for each given width adder. In order to obtain the optimal solution of FPGA implementations, the partitioning of the carry-select adders, and the algorithms used within the blocks and for the carry determination logic should be further examined. Table 5 gives the performance parameters for carry-select adders with different sizes.

Table 5. Implementation data of carry-select adders

Adder Width		8	16	24	32	40	48	56	64	72	80
4003	C	14	23	31	39	47	55	80	100	*	*
	T	24.7	31.5	35.2	34.8	39.2	41.7	51.3	50.7	*	*
	η	289	138	92	74	54	44	24	20	*	*
4005	C	14	23	31	39	47	55	80	92	108	121
	T	25.7	33.3	35.3	41.3	41.8	38.4	52.7	53	63.5	61.2
	η	278	131	91	62	51	47	24	21	15	14
4008	C	14	23	31	39	47	55	79	92	107	120
	T	26.3	32.5	34.1	37.3	43	43.5	52	55.2	65.5	55.8
	η	272	134	95	69	49	42	24	20	14	15
4010	C	14	23	31	39	47	55	79	92	108	121
	T	25.8	33.1	36.1	41.8	42.7	46.6	54.6	55.5	59.5	62.9
	η	277	131	89	61	50	39	23	20	16	13
4013	C	14	23	31	39	47	55	80	92	108	121
	T	29.3	34.4	37	44.9	48.8	57.7	61.7	56.3	70.7	73.4
	η	244	126	87	57	44	32	20	19	13	11

Note: * – can not be implemented in one device.

4. COMPARISONS

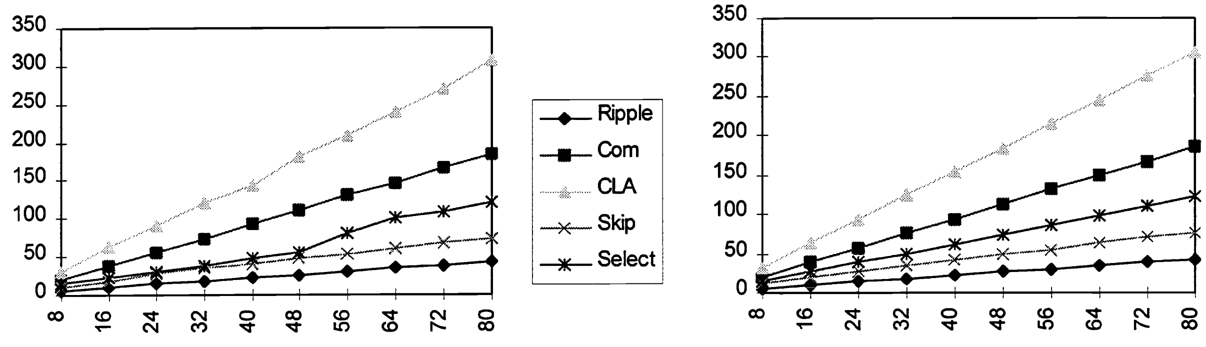
It is noted from Table 1 to Table 5 that the performance parameters of a specific addition technique implemented in different parttypes of FPGA's devices is slightly different. In general, the operational times resulted from smaller parttypes are slightly shorter than that resulted from larger parttypes. In order to compare the FPGA's implementations of different addition algorithms in a concrete context, the empirical formulas are derived from the worst implementation data of various addition algorithms in terms of cost, time and performance:cost ratio. These are collectively shown into Table 6. These empirical formulas can be used to quickly evaluate performance of the five addition algorithms. The graphs of the worst implementation data and the corresponding empirical formulas are shown in Figure 2 to Figure 4.

On the basis of the above tables and graphs, we can conclude that the carry-ripple adder has the lowest cost and the highest performance:cost ratio. Its highly regular structure and the effective use of the CLB's dedicated carry logic are the major reasons for this. Therefore, it should be the choice where simplicity and cost are critical factors.

The carry-completion adder has the worst operational time, the second worst cost and the second worst performance:cost ratio. Moreover, its asynchronous feature makes it extremely unsuitable to the designs of synchronous processing elements because the time lost in resynchronization may overweigh any gains from using the algorithms.

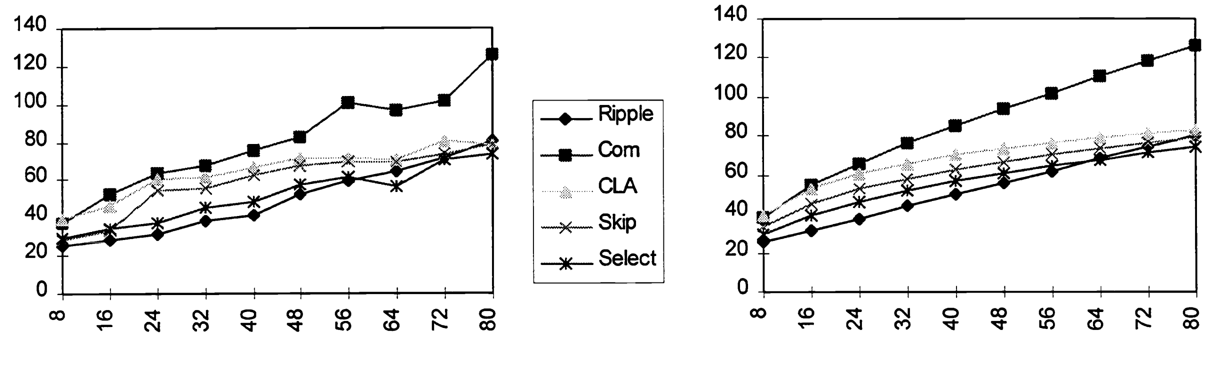
Table 6. The empirical evaluation formulas of different adders

Adder	Cost	Time	$\frac{\text{Performance}}{\text{Cost}}$
Carry-Ripple	$0.5n+2$	$0.77n+19$	$\frac{1}{0.4n^2+11n+38}$
Carry-Complete	$2.3n+1.7$	$(0.1n+12)\log_2 n$	$\frac{1}{(0.23n^2+27.8n+20.4)\log_2 n}$
CLA	$3.8n+2$	$13.2\log_2 n$	$\frac{1}{(26.4+50.2n)\log_2 n}$
Carry-Skip	$0.9n+5$	$(0.02n+11)\log_2 n$	$\frac{1}{(0.02n^2+10n+55)\log_2 n}$
Carry-Select	$1.5n+2.1$	$(0.03n+9.4)\log_2 n$	$\frac{1}{(0.05n^2+14.2n+19.7)\log_2 n}$



(a) The cost graphs of implementation data **(b)** The cost graphs of empirical formulas

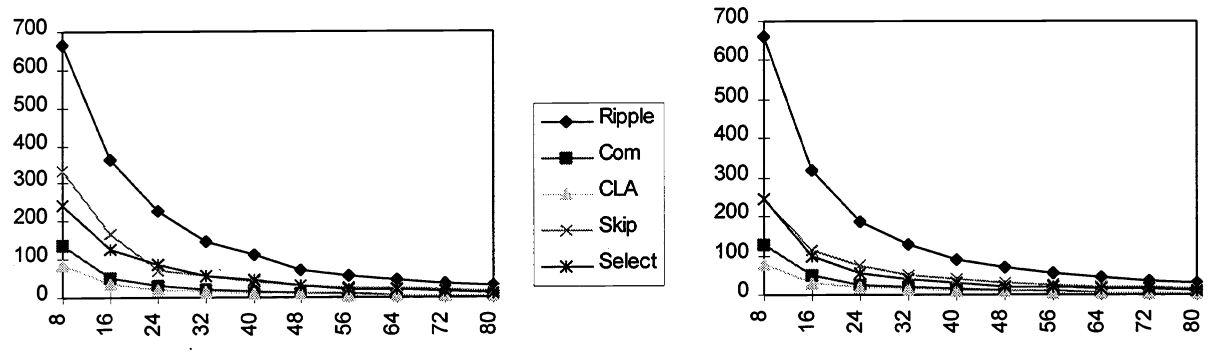
Figure 2. The cost graphs for different adders



(a) The time graphs of implementation data **(b)** The time graphs of empirical formulas

Figure 3. The operational time graphs for different adders

The fundamental CLA adder is theoretically the fastest adder. Because this adder has a high fan-in requirement, FPGA implementations of such adders larger than 9-bit must be implemented in multilevel. This leads to a longer operational time than is expected. The implementation results show that it has the highest cost and the worst performance:cost ratio. The major reasons for this are its irregular structure and that its inability to take advantage of the dedicated carry logic. Consequently, the pure CLA algorithm is impractical for FPGA's applications unless it is combined with some other algorithms which would give reasonable performance.



(a) The performance:cost ratio graphs of implementation data

(b) The performance:cost ratio graphs of empirical formulas

Figure 4. The performance:cost ratio graphs for different adders

The carry-skip adder is the next cheapest one in cost and the next best in performance:cost ratio. However, the operational time of this adder compares less favorably to that of the carry-ripple adder. This makes the carry-skip adder not the best choice of addition algorithms to be implemented in FPGA's. At the writing of this paper, this is not yet conclusive. The configuration optimization of this adder has yet to be examined. When the adder has been optimally designed, it could be a candidate for the best performing addition algorithm to be implemented in FPGA's.

From the tables and graphs above, the carry-select adder appears to be the most appropriate choice for FPGA's implementations. This adder has the best operational time when the adder width is larger than 56 bits at the medium cost. Although the cost does not appear to be very good, the algorithm does have the advantage of the regular structure and almost same the performance:cost ratio as that of the carry-skip adder. Moreover, other algorithms can be easily applied in this adder. When combined with other algorithms and after the further examination of partitioning the adder, the performance parameter for this adder could still be significantly improved.

5. CONCLUSIONS

In this work, we have implemented the five classical fixed-point addition algorithms in the widely used XC4000 device. An attempt has been made to model the performance of the adders with the empirical formulas derived from the data resulted from their implementations. The following conclusion can be drawn:

① For fast applications, the carry-skip adder and the carry-select adder appear to be the most appropriate solutions due to their excellent performance:cost ratio and the reasonable cost. ② For low cost applications, the carry-ripple adder is the most appropriate solution. Although the operational time is not as good as that of some others for larger adders, it does have a very simple and regular structure and the highest performance:cost ratio which make it attractive for the FPGA's applications and especially for the parallel applications. ③ The CLA and carry-completion adders seem to be the worst performers because of their high cost and low performance:cost ratio. ④ In general, those algorithms which have regular structures and can take advantage of the dedicated carry logic feature are suitable for FPGA implementations.

6. LIMITATIONS OF THE PRESENT EVALUATION

In practice, a successful evaluation is affected by a large number factors. In order to make a more accurate evaluation, the following factors should be considered. ① The two-operand addition is the most fundamental operation in computers. Therefore, the adders can be easily evaluated by isolating the hardware associated with two-operand adders from the rest of the computer. However, the adders are indeed tied to the rest of the computer. The compatibility of an adder with other components of the computer such as the ALU, memory, control circuitry, etc., should be carefully considered. Consequently, more work has to be done to link these results to the rest of a computing system. ② The results are obtained by implementing adders in a single chip. When all components of a computer are considered and multiple chips are used, the I/O resources should be taken account into the evaluation. Moreover, the present evaluation is based on the XC4000-families, therefore, it is difficult to say that the results is equally valid for other FPGA devices. ③ The regularity of an adder should be taken account into the evaluation because it is important to both the cost and design. However, it is very difficult to quantify the regularity in the evaluation.

REFERENCES

1. T.G.Rauscher and A.K.Agrawala, "Dynamic problem-oriented redefinition of computer architecture via microprogramming," ***IEEE Transactions on Computers,*** Vol.C-27, No.11, Nov. 1978, pp.33-40.
2. P.M.Athanas and H.F.Silverman, "Processor reconfiguration through instruction-set metamorphosis," ***Computer***, March 1993, pp.11-19.
3. J.Davidson, "FPGA implementation of a reconfigurable microprocessor," ***Proc. 1993 IEEE Custom Integrated Circuits Conference***, 1993, pp.3.2.1-3.2.4.
4. C.Iseli and E.Sanchez, "Beyond superscalar using FPGAs," ***Proc. 1993 IEEE International Conference on Computer Design***, 1993, pp.486-490.
5. N.W.Bergmann, J.C.Mudge, and L.R.Cirroco, "FPGA-based custom computers," ***Proc. 11th Australian Microelectronic Conference. Microelectronics, Meeting the Needs of Modern Technology***, 1993, pp.197-202.
6. N.W.Bergmann and J.C. Mudge, "An analysis of FPGA–based custom computers for DSP applications," ***Proc. 1994 IEEE International Conference on Acoustics, Speech & Signal Processing***, 1994, pp.II-513-516.
7. J.Schewel, M.Thornburg, and S.Casselman, "Transformable computers & hardware object technology," ***Proc. 9th International Parallel Processing Symposium***, 1995, pp.518-522.
8. ***Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines***, California, April, 1993.
9. ***Proceedings of IEEE Workshop on FPGA's for Custom Computing Machines***, California, April, 1994.
10. ***The Programmable Gate Array Data Book***, Xilinx, San Jose, Calif., 1994.
11. Kai Hwang, ***Computer Arithmetic-Principles, Architecture, and Design***, John Wiley & Sons, New York, 1979.
12. Joseph J.F. Cavanagh, ***Digital Computer Arithmetic-Design and Implementation***, McGraw-Hill, New York, 1984.
13. Israel Koren, ***Computer Arithmetic Algorithms***, Prentice Hall, New Jersey, 1993
14. Amos R. Omondi, ***Computer Arithmetic Systems-Algorithms, Architecture and Implementations*** Prentice Hall, Hertfordshire, 1994.
15. D.Salomon, "A design for an efficient NOR-gate only, binary-ripple adder with carry-completion-detection logic," ***The Computer Journal***, Vol.30, No.3, 1987, pp.283-285.
16. R.W.Doran, "Variants of an improved carry-lookahead adder," ***IEEE Transactions on Computers***, Vol.C-37, No.9, Sept. 1988, pp.1110-1113.
17. B.W.Y.Wei and C.D.Thompson, "Area-time Optimal adder design," ***IEEE Transactions on Computers***, Vol.39, No.5, May 1990, pp.666-675.
18. T.Lynch and E.E.Swartzlander, "A spanning tree carry-lookahead adder," ***IEEE Transactions on Computers***, Vol.41, No.8, Aug. 1992, pp.931-939.
19. B.S.Fagin, "Fast addition of large integers," ***IEEE Transactions on Computers***, Vol.41, No.9, Sept. 1992, pp.1069-1077.
20. A.Guyot, B.Hochet, and J.M.Muller, "A way to build efficient carry-skip adders," ***IEEE Transactions on Computers***, Vol.C-36, No.10, Oct. 1987, pp.1144-1152.
21. P.K.Chan and M.D.F.Schlag, "Analysis and design of CMOS Manchester adders with variable carry-skip," ***IEEE Transactions on Computers***, Vol.39, No.8, Aug. 1990, pp.983-992.
22. P.K.Chan, M.D.F.Schlag, C.D.Thomborson, and V.G.Oklobdzija, "Delay optimization of carry-skip adders and block carry-lookahead adders using multidimensional dynamic programming," ***IEEE Transactions on Computers***, Vol.41, No.8, Aug. 1992, pp.920-930.
23. A.Tyagi, "A reduced-area scheme for carry-select adders," ***IEEE Transactions on Computers***, Vol.42, No.10, Oct. 1993, pp.1163-1170.