International Journal of Electrical and Electronic Engineering & Telecommunications

ISSN 2319 – 2518 www.ijeetc.com
Vol. 3, No. 3, July 2014

Research Paper

DESIGN AND IMPLEMENTATION OF 32 BIT UNSIGNED MULTIPLIER USING CLAA, CSLA, ETA

Veersh B Jalihal[1]* and Naseeruddin[1]

*Corresponding Author: Veersh B Jalihal, ✉ veeresh.jalihal@gmail.com

This project deals with the comparison of the VLSI design of the Carry Look-Ahead Adder (CLAA) based 32-bit unsigned integer multiplier and the VLSI design of the carry select adder (CSLA) based 32-bit unsigned integer multiplier. Both the VLSI design of multiplier mUltiplies two 32-bit unsigned integer values and gives a product term of 64-bit values. The CLAA based multiplier uses the delay time of 99 ns for performing mUltiplication operation where as in CSLA based multiplier also uses nearly the same delay time for mUltiplication operation. But the area needed for CLAA multiplier is reduced to 31% by the CSLA based multiplier to complete the multiplication operation. These multipliers are implemented using Altera Quartus II and timing diagrams are viewed through avan waves. ETA is able to ease the strict restriction on accuracy, and at the same time achieve tremendous improvements in both the power consumption and speed performance.

Keywords: CLAA, CSLA, Delay, Area, Array multiplier, VHDL modeling, Simulation

INTRODUCTION

Digital computer arithmetic is an aspect of logic design with the objective of developing appropriate algorithms in order to achieve an efficient utilization of the available hardware. The basic operations are addition, subtraction, multiplication and division. In this, we are going to deal with the operation of additions implemented to the operation of multiplication. The repeated form of the addition operations and shifting results in the multiplication operations.

Given that the hardware can only perform a relatively simple and primitive set of Boolean operations, arithmetic operations are based on a hierarchy of operations that are built upon the simple ones. In VLSI designs, speed, power and chip area are the most often used measures for determining the performance and efficiency of the VLSI architecture.

Multiplications and additions are most widely and more often used arithmetic computations performed in all digital signal processing applications. Addition is a

[1] Department of ECE, BITM Bellary, Karnataka, India.

fundamental operation for any digital multiplication. A fast, area efficient and accurate operation of a digital system is greatly influenced by the performance of the resident adders.

Adders are also very important component in digital systems because of their extensive use in these systems.

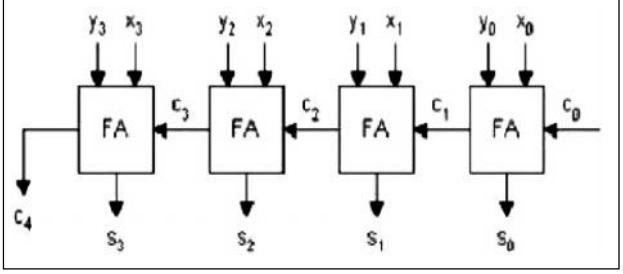
In this project we are going to compare the performance of different adders implemented to the multipliers based on area and time needed for calculation. On comparison with the Carry Look-Ahead Adder (CLAA) based multiplier the area of calculation of the carry select adder (CSLA) based multiplier is smaller and better with nearly same delay time. Here we are dealing with the comparison in the bit range of *n***n* (32*32) as input and 2 n (64) bit output.

Hence, to design a better architecture the basic adder blocks must have reduced delay time consumption and area efficient architectures. The demand is of DSP style systems for both less delay time and less area requirement for designing the systems. Our interest is in the basic building blocks of arithmetic circuits that dominate in DSP applications, VLSI architectures, computer applications and where ever reduced area computation is needed.

CARRY LOOK-AHEAD ADDER

Carry look-ahead adder can produce carries faster due to parallel generation of the carry bits by using additional circuitry. This technique uses calculation of carry signals in advance, based on input signals. The result is reduced carry propagation time. For example, ripple adders are slower but use the least energy.

Figure 1: Carry Look-Ahead Adder



Let *Gi* is the carry generate function and *Pi* be the carry propagate function. Then we can rewrite the carry function as follows:

$$Gi = Ai.Bi \quad ...(1)$$

$$Pi = (Ai \text{ xor } Bi) \quad ...(2)$$

$$Si = Pi \text{ xor } Ci \quad ...(3)$$

$$Ci + I = Gi + Pi.Ci \quad ...(4)$$

Thus, for 4-bit adder, we can compute the carry for all the stages as shown below:

$$C1 = GO + PO.CO \quad ...(5)$$

$$C2 = G1 + P1.C1 = G1 + P1.GO + P1.PO.CO \quad ...(6)$$

$$C3 = G2 + P2.C2 = G2 + P2.G1 + P2.P1.GO + P2.P1.PO.CO \quad ...(7)$$

$$C4 = G3 + P3.C3 = G3 + P3.G2 + P3.P2.G1 + P3.P2.P1.GO + P3.P2.P1.PO.CO \quad ...(8)$$

In general, we can write:

The sum function:

$$SUM_i – A_i \text{ xor } B_i \text{ xor } C_i – P_i \text{ xor } C_i \quad ...(9)$$

The carry function:

$$C_i + I = G_i + P_i.C_i \quad ...(10)$$

In general, we can write the algorithm as:

If Carry in = 1, then the sum and carry out are given by,

Sum(*i*) = *a*(*i*) xor *b*(*i*) xor '1' ...(11)

Carry(*i* + 1) = (*a*(*i*) and *b*(*i*)) or (*b*(*i*) or *a*(i)) ...(12)

If Carry in = 0, then the sum and carry out are given by,

Sum(*i*) = *a*(*i*) xor *b*(*i*) ...(13)

Carry(*i* + 1) = (*a*(*i*) and *b*(*i*)) ...(14)

The sum function:

$$S_i = C_i S_i^0 - C_i S_i^1 \quad ...(15)$$

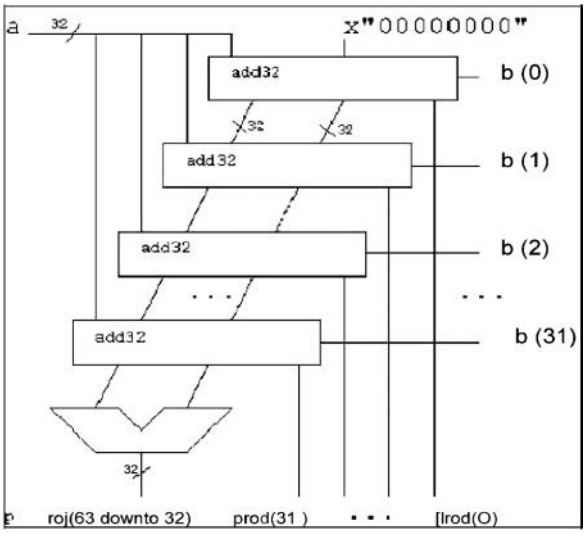
The carry function:

$$C_{i-1} = C_i C_{i+1}^0 + C_i C_{i-1}^1 \quad ...(16)$$

MULTIPLTER FOR UNSIGNED DATA

Multiplication involves the generation of partial products, one for each digit in the multiplier, as in Figure 3. These partial products are then summed to produce the final product. The multiplication of two n-bit binary integers results in a product of up to 2 n bits in length (Stallings, xxxx).

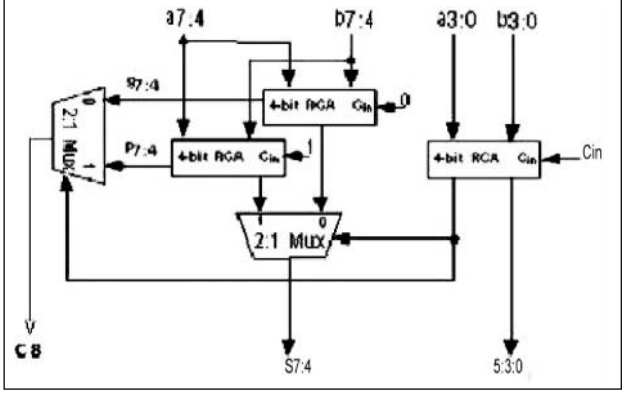
Figure 2: A Partial Schematic of the Multiplier



CARRY SELECTADDER

The concept of CSLA is to compute alternative results in parallel and subsequently selecting the correct result with single or multiple stage hierarchical techniques. In CSLA both sum and carry bits are calculated for two alternatives Cin = O and 1. Once Cin is delivered, the correct computation is chosen using a mux to produce the desired output. Instead of waiting for Cin to calculate the sum, the sum is correctly output as soon as Cin gets there. The time taken to compute the sum is then avoided which results in good improvement in speed.

Figure 3: Figure Carry Select Adder



We used the following algorithm to implement the multiplication operation for unsigned data.

ERROR TOLERANT ADDER

The commonly used terminologies in Error Tolerant addition are overall error and accuracy. They are defined by the equations discussed below. Overall Error (OE):

$$OE = |Rc - Re| \quad ...(17)$$

where *Re* is the result obtained by the Error tolerant addition technique, and *Rc* is the correct result (all the results are represented as decimal numbers).

Accuracy (ACC): In the case of the error tolerant design, the accuracy of an addition process is used to indicate how "correct" the output of an adder is for a particular input. Its value ranges from 0-100%.

$$ACC\% = (1 - (OE/Rc)) \times 100 \quad ...(18)$$

In the conventional adder circuit, the delay is mainly due to the carry propagation from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). Glitches in the carry propagation also cause significant power dissipation.

Therefore, if the carry propagation can be eliminated or curtailed, a great improvement in speed performance and power consumption [8] can be achieved. This new addition arithmetic can be illustrated via an example shown below.

In error tolerant addition technique, we first split the input operands into two parts: an accurate part that includes higher order bits and the inaccurate part that consists of lower order bits. The length of each part need not necessary be equal. The addition process starts from the middle, i.e., starting point in Figure 4 towards the two opposite directions at the same time.

In the example of Figure 4, the two 8-bit input operands, A = "10110111" (183) and B = "01101101" (109),are divided equally into 4 bits each for the accurate andinaccurate parts. The addition of the higher order bits (accurate part) of the input operands is carried from right toleft (LSB to MSB) and normal addition method is applied. This is to preserve its correctness since the higher order bitsplay a more important role than the lower order bits. The lower order bits of the input operands (in accurate part) require a special addition mechanism. No carry signal will beconsidered at any bit position to eliminate the carry propagation path. To minimize the overall error due to theelimination of the carry chain, a special strategy is adapted, and as follows: 1) check every bit position from left to right (MSB to LSB); 2) if both input bits are "0" or different, normal one-bit addition is performed and the operation proceeds to next bit position; 3) if both input bits are "1", the checking process stopped and from this bit onward, all sumbits to the right are set to "1". The addition

Figure 4: Arithmetic Procedure of Error Tolerant Adder

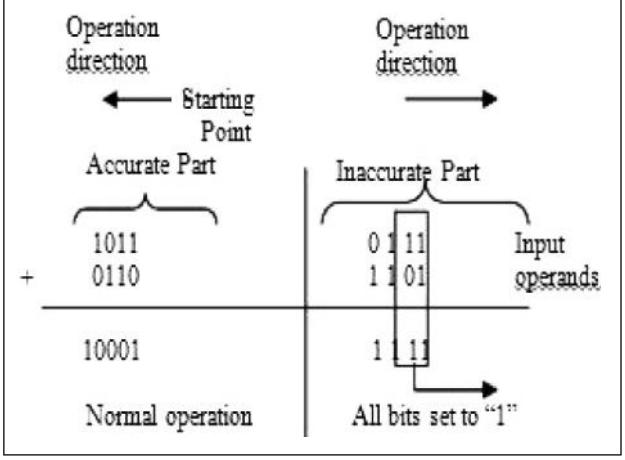
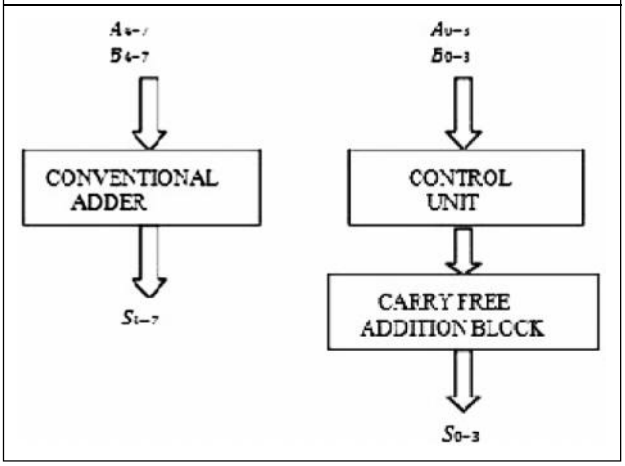


Figure 5: Block Diagram of Error Tolerant Adder



mechanism described can be easily understood from the example. For theaddition of the MSB part in modified booth multiplication we have adopted this technique.

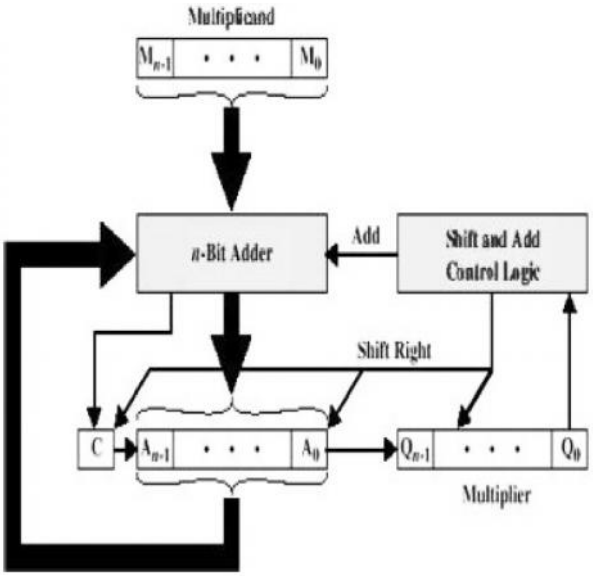
MULTIPLICATION ALGORITHM

Let the product register size be 64 bits. Let the multiplicand registers size be 32 bits. Store the multiplier in the least significant half of the product register. Clear the most significant half of the product register.

Repeat the following steps for 32 times:

- If the least significant bit of the product register is "1" then add the multiplicand to the most significant half of the product register.
- Shift the content of the product register one bit to the right (ignore the shifted-out bit).
- Shift-in the carry bit into the most significant bit of the product register. Figure 6 shows a block diagram for such a multiplier.

Figure 6: Multiplier of Two n-bit Values



VHDL SIMULATIONS

The VHDL simulation of the two multipliers is presented in this section. In this, waveforms, timing diagrams and the design summary for both the CLAA and CSLA based multipliers are shown in the figures. The VHDL code for both multipliers, using CLAA and CSLA, are generated. The VHDL model has been developed using Altera Quartus II and timing

Figure 7: Waveform for a CLAA Based Multiplier

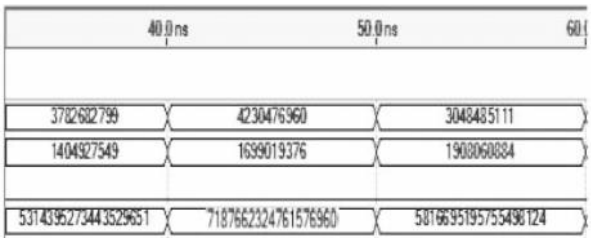


Figure 8: Waveform for CSLA Based Multiplier

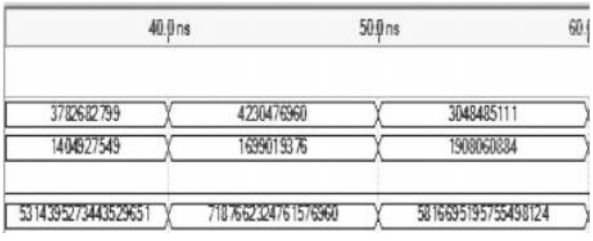


Figure 9: Timing Analysis for CLAA Based Multiplier

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	98.565 ns	a[3]	sum[63]	--	--	0

Figure 10: Timing Analysis for CSAA Based Multiplier

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tpd	N/A	None	99.553 ns	a[1]	sum[63]	--	--	0

Figure 11: Design Summary of CLAA Multiplier

Quartus II Version	8.1 Build 163 10/28/2008 SJ Web Edition
Revision Name	CLAmultiplier
Top-level Entity Name	CLAmultiplier
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Met timing requirements	Yes
Total logic elements	2,957 / 33,216 (9 %)
Total combinational functions	2,957 / 33,216 (9 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	129 / 475 (27 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

diagrams are viewed through avan waves. The multipliers use two 32-bit values.

Figure 12: Figure Design Summary of CSLA Multiplier

Quartus II Version	8.1 Build 163 10/28/2008 SJ Web Edition
Revision Name	csamultiplier
Top-level Entity Name	csamultiplier
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Rnal
Met timing requirements	Yes
Total logic elements	2,039/33.216 (6'4)
Total combinational functions	2,03:9 / 3.3.216(6 +4)
Dedicated logic registers	0 / 3.3.216(0%)
Total registers	0
Total pins	129/475 (27 %)
Total virtual pins	0
Total memory b-s	0 1 483,840 (0 '".)
Embedded Multiplier 9-bit elements	0 I 70 (0 '4)
Total PLLs	0/4(0%)

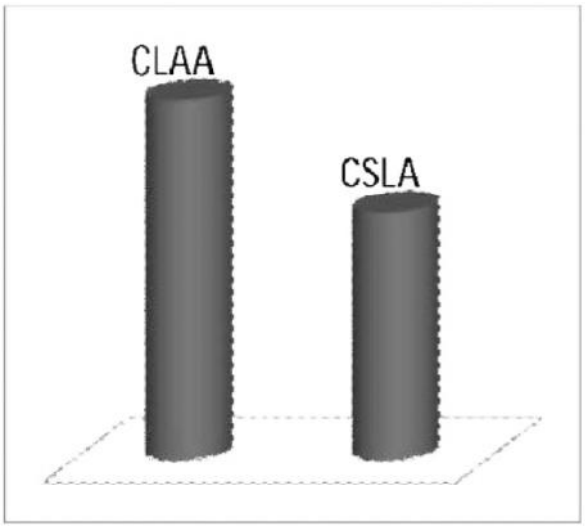
Under the worst case, the multiplier with a carry look-ahead adder uses time = 98.5 ns, while the multiplier with the carry select adder uses time = 99.5 ns.

PERFORMANCE ANALYSTS

Area Analysis

The performance analysis for the area of CLAA and CSLA based multipliers are represented in the form of the diagram shown in Figure 13.

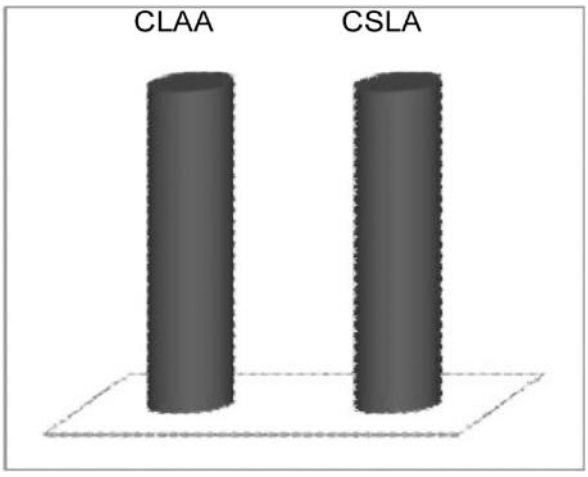
Figure 13: Figure Area Analysis Chart



Delay Analysis

The performance analysis for the delay time of CLAA and CSLA based multipliers are represented in the form of the diagram shown in Figure 14.

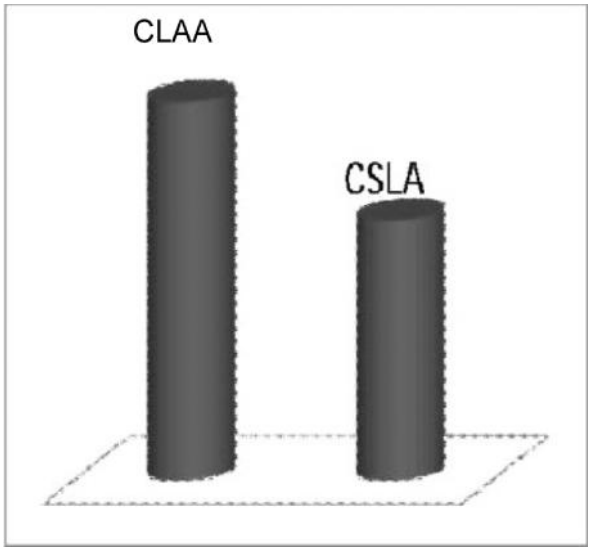
Figure 14: Figure Delay Analysis Chart



Area Delay Product Analysis

The performance analysis for the area delay product of CLAA and CSLA based multipliers are represented in the form of the diagram shown in Figure 15.

Figure 15: Figure Area Delay Product Analysis Chart



The area needed and delay for both the CLAA and CSLA implemented to the multiplier was analyzed and the comparison was shown in the figure in the form of a table.

Analysis Table

Tn this analysis table shown in Table 1, the delay time is nearly same, the area and the area delay product of CSLA based multiplier is reduced to 31% when compared to CLAA based multiplier.

Table 1: Analysis Table

Multiplier Type	Delay (ns)	Area	Delay Area Product
CLAA based multiplier	98.5	2957 logic cells	291264.5
CSLA based multiplier	99.5	2039 logic cells	202880.5
ETA	95	2021	

CONCLUSION

A design and implementation of a VHDL-based 32-bit unsigned multiplier with CLAA and CSLA was presented. VHDL, a Very High Speed Integrated Circuit Hardware Description Language, was used to model and simulate our multiplier. Using CSLA improves the overall performance of the multiplier.

Thus a 31% area delay product reduction is possible with the use of the CSLA based 32-bit unsigned parallel multiplier than CLAA based 32-bit unsigned parallel multiplier. The application for ETA are in those areas where there is no strict restriction on accuracy or when high speed performance is more important compared to accuracy.

FUTURE WORK

This 32-bit multiplier can be further extended to 64-bit multiplier and 128-bit multiplier using the proposed method for multiplication operation can be done as future work.

REFERENCES

1. Armstrong J R and Gray F G (2000), *VHDL Design Representation and Synthesis*, 2nd Edition, Prentice Hall, USA, ISBN: 0-13-021670-4.
2. Asadi P and Navi K (2007), "A Novel Highs-Speed 54-54 Bit Multiplier", *Am. J. Applied Sci.*, Vol. 4, No. 9, pp. 666-672.
3. Brown S and Vranesic Z (2005), *Fundamentals of Digital Logic with VHDL Design*, 2nd Edition, McGraw-Hill Higher Education, USA, ISBN: 0072499389.
4. Hasan Krad and Aws Yousi (2010), "Design and Implementation of a Fast Unsigned 32-bit Multiplier Using VHDL".
5. Meier P C H, Rutenbar R A and Carley L R (1996), "Exploring Multiplier

Architecture and Layout for Low Power", CIC'96.

6. Mohanty P S (2009), "Design and Implementation of Faster and Low Power Multipliers", Bachelor Thesis, National Institute of Technology, Rourkela.
7. Navabi Z (2007), *VHDL Modular Design and Synthesis of Cores and Systems*, 3rd Edition, McGraw-Hill Professional, USA, ISBN: 9780071508926.
8. Sertbas A and Ozbey R S (2004), "A Performance Analysis of Classified Binary Adder Architectures and the VHDL Simulations", *J. Elect. Electron. Eng.*, Vol. 4, pp. 1025-1030, Istanbul, Turkey.
9. Software Simulation Package: Direct VHDL, Version 1.2 (2007), Green Mounting Computing Systems Inc.,Essex, VT, UK.
10. Stallings W (2006), "Computer Organization and Architecture Designing for Peljormance", 71h ed., Prentice Hall, Pearson Education International, USA, ISBN: 0-13-185644-8.
11. Wakerly F (2006), *Digital Design-Principles and Practices*, 4th Edition, Pearson Prentice Hall,USA, ISBN: 0131733494.