Memoirs of the Faculty of Engineering, Okayama University, Vol.24, No. 2, pp.89-98, March 1990

Functional Testing of an ALU

Tokumi YOKOHIRA*

(Received January 10 , 1990)

SYNOPSIS

This paper considers a test set for an ALU with look ahead carry generators(LCGs). The ALU is logically partitioned into two groups of blocks, the group of one-bit operation units and LCG group. Each group is tested in parallel and exhaustively, independent of the other. And an easily testable design is applied to several blocks for decreasing the number of the input combinations of them. Under the easily testable design, a minimum test set for each group is generated, and the upper and lower bounds for a minimum test for the ALU are derived. The difference of the lower and upper bounds is not large, and a test set whose number of test vectors is equal to the upper bound can be easily obtained as the union of minimum test sets for two groups. Hence, the union can be used as a complete and practical test set for the ALU.

1. INTRODUCTION

An arithmetic login unit (ALU) is one of the main components of the processor, and must be precisely tested when it is used in the situation that high reliability are required.

ALU is one of combinational circuits. It can be, therefore, tested by the use of the classical test pattern generation methods, e.g., D-algorithm[1] and PODEM method[2]. However, in these methods, as the word size of ALU increases, the time required for generating test patterns also increases rapidly, and the detection of the multiple faults is not always guaranteed. Abraham et al.[3] and Becker[4] have proposed the methods to generate test patterns easily. However, they restricted the internal structure of ALU to a tree structure. Hence, it may be hard to apply their methods to usual ALU structures.

One solution for these problems is to partition an ALU into several blocks and test all of the blocks exhaustively[5,6]. By the use of this method, Sridhar et al.[7] have generated a test

*Department of Information Technology

set for the ripple carry ALU. However, ALU usually has a *look ahead carry generator* (LCG) for performing high speed operations. Hence, it is also hard to apply their methods to such ALUs.

In this paper, we consider a test set for an ALU with LCG by the use of a block partitioning method. The ALU is partitioned into two groups of blocks, the group of one-bit operation units and LCG group. Each group is tested in parallel and exhaustively, independent of the other. And an easily testable design is applied to several blocks for decreasing the number of the input combinations of them. Under such a policy, we will generate a practical and complete test set for the ALU.

In the next section, the configuration of the ALU is described. In section 3, the easily testable design are explained. Section 4 gives a minimum test set for each group, and a practical and complete test set for the ALU as the union of minimum test sets for two groups.

2. OPERATIONS AND CONFIGURATION OF ALU

2.1 Operations of ALU

Operations of an ALU are separated into the arithmetic operations and logical ones. Basic arithmetic and logical operations implemented on most ALUs are shown in Table 1, where n, x $(= x_{n-1} \cdots x_1 x_0)$ and y $(= y_{n-1} \cdots y_1 y_0)$ are the word size, the inputs of an n-bit ALU, and z $(= z_{n-1} \cdots z_1 z_0)$ is the output. And C_{in} is a value of of the processor carry flag.

Table 1 Operations of ALU

Arithmetic Operation		Logical Operation	
increment (INC)	$z = x + 1$	or (OR)	$z_i = x_i \vee y_i$
add (ADD)	$z = x + y$	and (AND)	$z_i = x_i \cdot y_i$
add with carry (ADC)	$z = x + y + C_{in}$	exclusive or (XOR)	$z_i = x_i \oplus y_i$
decrement (DEC)	$z = x - 1$	equivalence (EQU)	$z_i = \overline{x_i \oplus y_i}$
subtract (SUB)	$z = x - y$	complement (CMP)	$z_i = \overline{x_i}$
subtract with carry (SBC)	$z = x - y - C_{in}$	through (THR)	$z_i = x_i$

$(0 \leq i \leq n-1)$

2.2 Configuration of ALU

If $n = 4$, all of the operations shown in Table 1 can be realized by the 4-bit ALU illustrated in Fig. 1. The external signal SC is the vector of control signals in the ALU, provided from the processor control circuits, and specifies the operation to be executed. And the value of the external signal C_{out} is sent to the processor carry flag after each operation. All blocks in Fig. 1 are combinational circuits. Block A_i $(0 \leq i \leq 3)$ generates the external output signal z_i and internal signals P_i and G_i called the *carry propagate* signal and *carry generate* one, respectively,

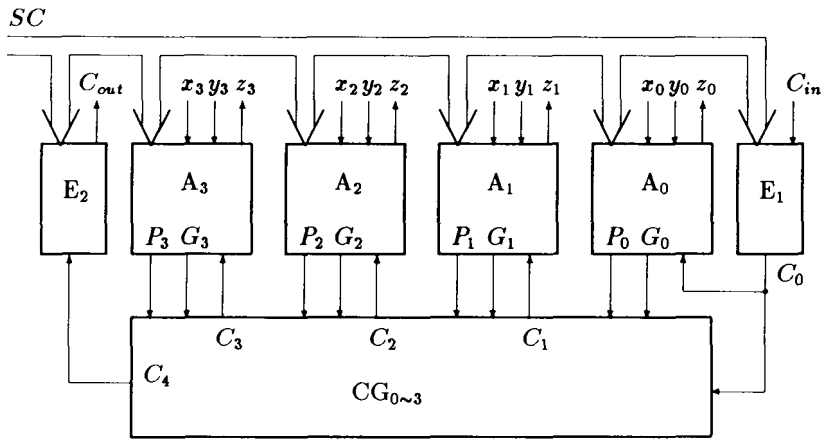

Fig. 1 Configuration of ALU

from the external input signals x_i and y_i and internal carry signal C_i. Internal configurations of all A_is are identical. Block E_1 generates the internal signal C_0 from C_{in}, and block E_2 generates C_{out} from the internal carry signal C_4. We call the group of blocks E_1, E_2 and all A_is the *operational unit* (OU). Block $CG_{0\sim3}$ is a *look ahead carry generator* (we call it LCG briefly,) which generates the internal carry signals (C_1, C_2, C_3 and C_4).

The relations between the inputs and outputs of OU are shown in Table 2, where *'d'* represents *"don't care"* value. SC_{Aj} and SC_{Lj} $(0 \leq j \leq 5)$ represent values of SC for each operation. And the relation between the inputs and outputs of $CG_{0\sim3}$ is as follows:

$$C_{i+1} = G_i \vee P_i G_{i-1} \vee P_i P_{i_1} P_{i-2} \vee \cdots \vee P_i P_{i-1} \cdots P_0 C_0 \quad (0 \leq i \leq 3). \tag{1}$$

When $n > 4$, OU has similar structure shown in Fig. 1, and LCG is usually constructed with a layered structure[8]. For example, when $n = 16$, LCG has two layers as illustrated in Fig. 2. All blocks in LCG, i.e., $CG_{t\sim t+3}$ $(t = 0, 4, 8, 12)$ and $CG_{0\sim15}$ have similar internal structures. $CG_{t\sim t+3}$ generates the internal signals $P_{t\sim t+3}$ and $G_{t\sim t+3}$ called the *block carry propagate* signal and *block carry generate* one. The relations between the inputs and outputs of $CG_{t\sim t+3}$ are as follows:

$$P_{t\sim t+3} = P_t P_{t+1} P_{t+2} P_{t+3}, \tag{2}$$

$$G_{t\sim t+3} = G_{t+3} \vee G_{t+2} P_{t+3} \vee G_{t+1} P_{t+3} P_{t+2} \vee G_t P_{t+3} P_{t+2} P_{t+1}, \tag{3}$$

$$C_{t+4} = G_{t\sim t+3} \vee P_{t\sim t+3} G_{t-4\sim t-1} \vee \cdots \vee P_{t\sim t+3} P_{t-4\sim t-1} \cdots P_{0\sim3} C_0. \tag{4}$$

Similarly, for n $(4^{k-1} < n \leq 4^k)$, LCG is constructed with k layers.

3. TEST FOR ALU

3.1 Assumptions for Test

For simplicity, in this paper, OU and LCG are tested separately. All of the blocks in OU and LCG are tested in parallel and exhaustively. In testing, values of the internal input lines

Table 2 Input-output relation of OU

Operation	SC	z_i	P_i	G_i	C_0	C_{out}
INC	SC_{A0}	$x_i \oplus C_i$	x_i	0	1	C_4
ADD	SC_{A1}	$x_i \oplus y_i \oplus C_i$	$x_i \vee y_i$	$x_i \cdot y_i$	0	C_4
ADC	SC_{A2}	$x_i \oplus y_i \oplus C_i$	$x_i \vee y_i$	$x_i \cdot y_i$	C_{in}	C_4
DEC	SC_{A3}	$\overline{x_i \oplus C_i}$	1	x_i	0	$\overline{C_4}$
SUB	SC_{A4}	$x_i \oplus \overline{y_i} \oplus C_i$	$x_i \vee \overline{y_i}$	$x_i \cdot \overline{y_i}$	$\overline{C_{in}}$	$\overline{C_4}$
SBC	SC_{A5}	$x_i \oplus \overline{y_i} \oplus C_i$	$x_i \vee \overline{y_i}$	$x_i \cdot \overline{y_i}$	0	$\overline{C_4}$
OR	SC_{L0}	$x_i \vee y_i$	d	d	d	d
AND	SC_{L1}	$x_i \cdot y_i$	d	d	d	d
XOR	SC_{L2}	$x_i \oplus y_i$	d	d	d	d
EQU	SC_{L3}	$\overline{x_i \oplus y_i}$	d	d	d	d
CMP	SC_{L4}	$\overline{x_i}$	d	d	d	d
THR	SC_{L5}	x_i	d	d	d	d

d : *don't care*, $(0 \leq i \leq 3)$

of each block are indirectly controlled through other blocks from the external lines of the ALU. Similarly, values of the internal output lines of each block are indirectly observed through other blocks at the external lines. We make the following assumptions about faults:

(1) All faults are solid ones.
(2) There are no bridge faults between two arbitrary blocks.
(3) Any sequential circuit is never produced by faults.
(4) Faults occur in one block at most.
(5) The *indistinguishable faults*[9], i.e., faults that don't have influence on the external output values of the ALU are neglected.

The assumptions (1) $\sim$ (4) made for simplifying the derivation of the test set for the ALU, and by the assumption (4) and (5), it is not necessary to consider the controllability and observability of all blocks.

3.2 Easily Testable Design

3.2.1 Easily Testable Design of OU

The number of test vectors, i.e., the number of $(n+2)$-tuple $(SC, x_{n-1}, y_{n-1}, \cdots, x_1, y_1, x_0, y_0, C_{in})$s for testing each block is equal to the number of all its input combinations. And, as

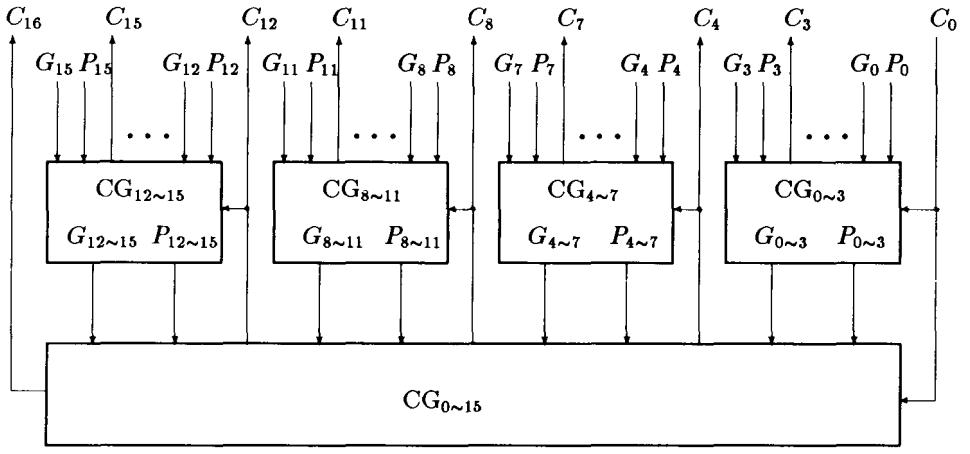

Fig. 2 Configuration of LCG with layered structure

shown in Fig. 1, C_4 and C_i $(0 \leq i \leq 3)$ are inputs of E_2 and A_i, respectively. However, in the logical operations, these values don't have influence on values of the outputs of E_2 and all A_is from Table 2. Therefore, these values are allowed to be fixed at either '0' or '1', and by being fixed, the number of test vectors for E_2 and all A_is can be decreased by half. In the logical operations, if an easily testable design as described below is applied to E_1 and all A_is, then, values of all C_is can be fixed at '0'.

It is trivial that, if all 'd's of the column C_0 in Table 2 are fixed at '0', then, the value of C_0 can be fixed at '0'. And, the situation that the value of C_0 is '0' also occurs in the arithmetic operations. Therefore, even if the value of C_0 is fixed at '0', the number of test vectors for $CG_{0\sim3}$ doesn't increase. Thus, we design E_1 so that all 'd's of the column C_0 are fixed at '0'.

Since C_0 is fixed at '0' and from the equations (1) $\sim$ (3), if the value of each (P_i, G_i) $(0 \leq i \leq 3)$ is $(0,0)$, then, the value of each C_{i+1} becomes '0'. And, the situation that $(P_i, G_i) = (0,0)$ also occurs in the arithmetic operations. Therefore, even if the value of each (P_i, G_i) is fixed at $(0,0)$, the number of test vectors for $CG_{0\sim3}$ doesn't increase. Therefore, we design all A_is so that all 'd's of the columns P_i and G_i are fixed at '0'.

As described above, after all, we design all A_is and E_1 so that all 'd's of the columns C_0, P_i and G_i are fixed at '0'. This design can be easily applied, and the amount of additional hardwares for it may be little.

3.2.2 Easily Testable Design of LCG

LCG has a layered structure as described above when n is a value more than four. As seen from Table 2, the equations (2) and (3), each $(P_{t\sim t+3}, G_{t\sim t+3})$ $(t = 0, 4, 8, 12)$ can be $(0,0)$, $(0,1)$, $(1,0)$ or $(1,1)$. However, from the equation (4), values of all outputs of $CG_{0\sim15}$ are identical for

either $(P_{t\sim t+3}, G_{t\sim t+3}) = (0,1)$ or $(1,1)$. Therefore, we design $CG_{t\sim t+3}$s that generate $(0,0)$, $(1,0)$ or $(1,1)$ as a value of $(P_{t\sim t+3}, G_{t\sim t+3})$. This design can be applied by changing the equation (2) into the equation (5),

$$P_{t\sim t+3} = G_{t+3} \vee G_{t+2}P_{t+3} \vee G_{t+1}P_{t+2}P_{t+3} \vee (G_t \vee P_t)P_{t+1}P_{t+2}P_{t+3}. \quad (5)$$

The equation (5) is the equation summed logically (OR) the right hand sides of the equations (2) and (3).

By designing $CG_{t\sim t+3}$ as described above, the number of test vectors for $CG_{0\sim 15}$ can be decreased from $2 \times 4^4 (= 512)$ to $2 \times 3^4 (= 162)$, and additional hardwares are only four OR gates.

4. TEST VECTORS

4.1 Test Vectors for OU

In the logical operations, the value of C_i $(0 \leq i \leq n-1)$ is fixed at '0' as described in subsection 3.2. Therefore, four input combinations of (x_i, y_i) are necessary and sufficient for testing A_i per logical operation. Similarly, one and two input combinations are necessary and sufficient for testing E_1 and E_2, respectively. And all input combinations for each block in OU can be applied, independent of the other blocks. Therefore, all blocks in OU can be exhaustively tested by using four test vectors shown in Table 3 per logical operation.

Table 3 Minimum test set for OU in the logical operations

SC	$x_{n-1}y_{n-1}x_{n-2}y_{n-2}$				.	.	.	x_0	y_0	C_{in}
SC_{L0}	0	0	0	0	.	.	.	0	0	0
SC_{L0}	0	1	0	1	.	.	.	0	1	1
SC_{L0}	1	0	1	0	.	.	.	1	0	d
SC_{L0}	1	1	1	1	.	.	.	1	1	d
SC_{L1}	0	0	0	0	.	.	.	0	0	0
SC_{L1}	0	1	0	1	.	.	.	0	1	1
.		.			.	.	.		.	
.		.			.	.	.		.	
.		.			.	.	.		.	
SC_{L5}	0	0	0	0	.	.	.	0	0	0
SC_{L5}	0	1	0	1	.	.	.	0	1	1
SC_{L5}	1	0	1	0	.	.	.	1	0	d
SC_{L5}	1	1	1	1	.	.	.	1	1	d

d : *don't care*

In the arithmetic operations, values of C_i $(0 \leq i \leq n-1)$ can be '0' and '1'. Hence, the derivation of a minimum test set for OU is pretty complicated. In case of ADC operation $(SC = SC_{A2})$, if eight input combinations of (x_0, y_0, C_0) are applied to A_0, then, four '0's and four '1's are generated as values of C_1, as seen from Table 2 and the equations described above. Similarly, for eight input combinations of (x_1, y_1, C_1), four '0's and four '1's are generated as values of C_2, and so on. This situation can be represented with a directed graph (we call it the *input-output* graph of ADC operation) in Fig. 3. The number labeled on each vertex represents

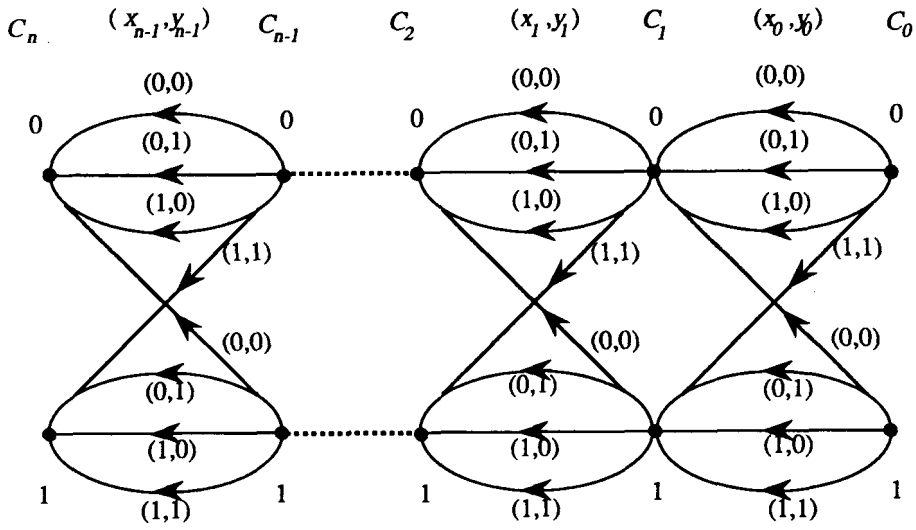

Fig. 3 Input-output graph ($SC = SC_{A2}$)

a value of C_i, and each arc represents that C_{i+1} is determined the value of C_i and the value of (x_i, y_i) labeled on it. In the input-output graph, eight arcs linking C_i and C_{i+1} correspond to eight input combinations of A_i. And, eight arc sequences reaching from C_0 to C_n can cover all of the arcs sufficiently, and reversely, eight ones are necessary. It is trivial that an arc sequence corresponds to a test vector for OU. Therefore, eight test vectors are necessary and sufficient for testing all of block in OU. In case of SBC operation, the input-output graph is similar to one in ADC operation. And in cases of ADD and SUB operations, the input-output graphs are similar to one in ADC operation except that the value of C_0 is fixed at '0' or '1'. Thus, in cases of ADC, SBC, ADD and SUB operations, minimum test sets can be easily obtained by the use of the input-output graphs. The number of test vectors in every minimum test set is eight independently of the word size n.

For INC operation ($SC = SC_{A0}$), we illustrate the input-output graph in Fig. 4. If eight input combinations of (x_i, y_i, C_i) are applied to A_i, six '0's and two '1's are generated as values of C_{i+1} as seen from Table 2 and the equations described above. Therefore, even if eight test vectors are applied, they can not test two input combinations of A_{i+1}, and two other test vectors must be applied for testing A_{i+1} exhaustively. Thus, ten test vectors are necessary and sufficient for testing both A_i and A_{i+1}, exhaustively and simultaneously. Similarly, twelve test vectors are necessary and sufficient for testing three adjacent blocks A_i, A_{i+1} and A_{i+2}, exhaustively and simultaneously. Hence, the number of test vectors in a minimum test set for OU is $2n + 4$ $(= 2(n - 2) + 8)$. And, the input-output graph of DEC operation is similar to one in INC operation except that the numbers of '0's and '1's are in reverse. Therefore, a minimum test set also has $2n + 4$ test vectors.

From the discussions above, it is concluded that the total number of test vectors of a

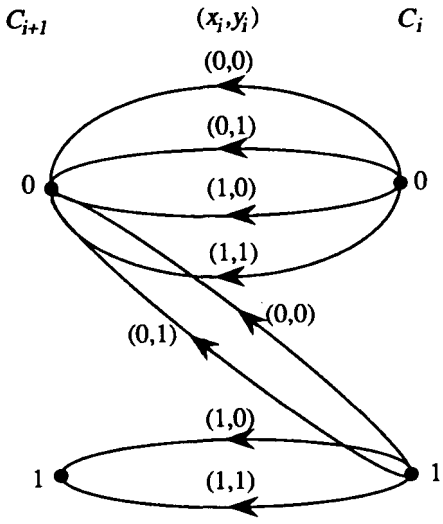

Fig. 4 Input-output graph ($SC = SC_{A0}$)

minimum test set for OU is $4n + 64$ ($= 4 \times 6 + 8 \times 4 + (2n + 4) \times 2$).

4.2 Test Vectors for LCG

In this subsection, we consider the number of test vectors of a minimum test set for LCG, where we assume $n = 2^m$ ($m \geq 2$) for practical use.

Suppose that every layer of LCG is separately tested. In Fig. 1 and Fig 2, all of the blocks in LCG have nine inputs, and generate $(0, 0)$, $(1, 1)$ and $(1, 0)$ as values of (P_i, G_i) $(0 \leq i \leq n-1)$ and $(P_{t\sim t+3}, G_{t\sim t+3})$ $(t = 0, 4, 8, 12)$. Thus, each block in LCG can be tested with 2×3^4 $(= 162)$ input combinations exhaustively. Hence, if $n = 4$, it is trivial that 162 test vectors can test $CG_{0\sim 3}$ exhaustively. If $n = 8$, $CG_{0\sim 3}$, $CG_{4\sim 7}$ and $CG_{0\sim 15}$ shown in Fig. 2 are used. If 162 input combinations are applied to $CG_{0\sim 3}$, 81 '0's and 81 '1's are generated as values of C_4. From the same reason described in subsection 4.1, 162 test vectors can test $CG_{0\sim 3}$ and $CG_{4\sim 7}$, exhaustively and simultaneously. Similarly, even if n increases, all blocks in the first layer of LCG can be tested with 162 test vectors, exhaustively and simultaneously. And from similar discussions above, it is also trivial that all blocks in each layer of LCG can be tested with 162 test vectors, exhaustively and simultaneously. In Table 4, we show the number of test vectors in a minimum test set.

In the discussions above, it is assumed that every layer of LCG is tested separately. Next, we attempt to overlap the test vectors of individual layer. For simplicity, we describe the case, $n = 16$. If 81 input combinations are applied to $CG_{0\sim 3}$ for either value of C_0, forty $(0, 0)$s, forty $(1, 1)$s and one $(1, 0)$ are generated as values of $(P_{0\sim 3}, G_{0\sim 3})$. And, each of the necessary numbers of $(0, 0)$, $(1, 1)$ and $(1, 0)$ is twenty-seven $(= 3^4/3)$. Therefore, if twenty-six $(= 27 - 1)$ $(1, 0)$s are appended, the first and second layers can be tested with $107(= 81 + 26)$ test vectors,

Table 4 Minimum number of test vectors (testing every layer)

# of bits (n)	First Layer	Second Layer	Third Layer
4	162	–	–
8	162	18	–
16	162	162	–
32	162	162	18
64	162	162	162

exhaustively an simultaneously. For a different value of n, the similar discussions above hold. In Table 5, we show the number of test vectors of a minimum test set generated under the condition that all layers are tested exhaustively and simultaneously.

Table 5 Minimum number of test vectors (testing all layers simultaneously)

# of bits (n)	Minimum Number of Test Vectors
4	162
8	166
16	214
32	218
64	266

4.3 Minimum Number of Test Vectors for ALU

In subsection 4.1 and 4.2, the numbers of test vectors in two minimum test sets for OU and LCG are shown under the condition that OU and LCG are tested separately. In this subsection, we discuss a practical test set for the ALU.

Let S_{OU} and S_{LCG} denote minimum test sets obtained in subsection 4.1 and 4.2, respectively. Then, the union S_T of S_{OU} and S_{LCG} is a complete test set for the ALU. It is not, however, a minimum test set for the ALU, because each of S_{OU} and S_{LCG} is independently derived. The number of test vectors of the minimum test set (denoted by N_m,) satisfies the following.

$$Max(|S_{OU}|, |S_{LCG}|) \leq N_m \leq |S_{OU}| + |S_{LCG}|, \tag{6}$$

where $|S_{OU}|$ and $|S_{LCG}|$ are the numbers of test vectors of S_{OU} and S_{LCG}, respectively. From the equation (6), the upper and lower bounds of N_m can be obtained as shown in Table 6. It is hard to obtain the minimum test set. And, it is easy to obtain the complete test set whose number of test vectors is equal to the upper bound. Furthermore, the test set corresponding to the upper bound may be considered to be sufficiently small for practical use. Thus, we can conclude that the complete test set can be used as a practical test set for the ALU.

Table 6 The lower and upper bound

# of bits (n)	Lower Bound	Upper Bound
4	162	162 + 56 + 24 = 242
8	166	166 + 72 + 24 = 262
16	214	214 + 104 + 24 = 342
32	218	218 + 168 + 24 = 410
64	296	266 + 296 + 24 = 586

5. CONCLUSIONS

In this paper, we considered a test set for the ALU, partioned into two groups of the operation units and look ahead carry generators. As the results, we obtained a minimum test set for each group. Since the number of test vectors of it is sufficiently small for practical use, the union of minimum test sets for two groups can be used as a practical and complete test set for the ALU.

Our future work is to deal with minimum test sets for ALUs with other high speed carry generators.

REFERENCES

(1) J. P. Roth : "Diagnosis of Automata Failures : a Calculus and a Method," IBM Journal of Research and Development, No. 10, pp. 278-291 (July, 1966).

(2) P. Goel : "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. Comput. Vol. **C-30**, No. 3, pp. 215-222 (March, 1981).

(3) J. Abraham and D. Gajski : "Design of Structures Defined by Simple Loops," IEEE Trans. Comput. Vol. **C-30**, No. 11, pp. 875-884 (November, 1981).

(4) B. Becker : "Efficient Testing of Optimal Time Adders," IEEE Trans. Comput. Vol. **C-30**, No. 9, pp. 1113-1121 (September, 1988).

(5) E. J. McClusky and S. B. Nesbet : "Design for Autonomous Test," IEEE Trans. Comput. Vol. **C-30**, No. 11, pp. 866-875 (November, 1981).

(6) E. J. McClusky : "Verification Testing – A Pseudoexhaustive Test Technique," IEEE Trans. Comput. Vol. **C-33**, No. 6, pp. 541-546 (June, 1984).

(7) T. Sridhar and J. P. Hayes : "A Functional Approach to Testing Bit-Sliced Microprocessors," IEEE Trans. Comput. **C-30**, No. 8, pp. 563-571 (August, 1981).

(8) J. A. Brzozowski and M. Yoeli : "Digital Networks," Prentice Hall (1976).

(9) H. Fujuiwara : "Logic Testing and Design for Testability," The MIT Press (1985).