Self-Repairable EPLDs II: Advanced Self-Repairing Methodology

Chong H. Lee, Marek A. Perkowski, Douglas V. Hall, and David S. Jun
Portland State University, Dept. of Electrical & Computer Engineering

Abstract

This paper describes an advanced self-testing, self-repair architecture and methodology for Lattice Logic Corp GAL (Generic Array Logic) devices used in high security and safety applications such as aerospace systems, military systems, or medical instruments. Described here is a new "column re-use" method that, if possible, exchanges a faulty GAL column with a column that needs the same programming as supplied by the faulty column and then reprograms the freed column to replace the faulty one. In contrast, our former self-repairing methodology, called "column replacement method with extra columns", described in [1], just discarded each faulty column and replaced it with an extra column. Our evaluation methodology shows that the lifetime of a GAL that uses the 'column re-use' method is longer than the lifetime of a GAL that uses just the 'column replacement' method or 'no self-repairing' method. Our results also give information on how many extra columns a GAL needs to reach a lifetime goal, in terms of simulation looping time, until the GAL is not useful any more. Our system is applicable to other devices such as FPGAs (Field Programmable Gate Arrays).

1 Introduction

Post-fabrication self-repair of digital circuits that have to work in adverse conditions such as increased cosmic radiation is not a new idea. Refer for instance, to the research of the "Hundred Year Spacecraft" in NASA [2]. However, such circuits have not been built in VLSI and so far not much has been published on them, except in the area of memories. Although the self-repair problem is already important for current technologies, it will become a must when the next scientific revolution of "molecular engineering" or "nanotechnology" creates molecular computers [3,4,5,6] that will be implanted in the human body. In high reliability applications such as these, a circuit should not only test itself but also repair itself as quickly as possible.

There has been very little research on diagnosis-based self-repair at the logic level and the few existing publications present various ideas for PLA (Programmable Logic Array) and PLA-based circuits [7,8,9,10,11,12]. However, to our knowledge, the circuits analyzed by other authors were not designed nor even simulated for reliability analysis.

We choose to work with a type of PLD called the Lattice GALs because these devices or similar devices are widely used, the technology used to achieve programmability in these devices is used in many other devices, and failure rate data was available for this technology.

For failure rate, we use two terms, Early Failure Rate (PPM; Parts Per Million devices or DPM; Defects Per Million devices from 0 year to 1 year) and the Long Term Failure Rate (FIT; Failures In Time from 0 year to 10 years). From data provided by National Semiconductor Corporation [1,13,14,15]; EEPROM failure rate, Early Failure Rate = 0.0489 %, Long Term Failure Rate = 5.07 %. The FIT value gives the maximum number of failures that will occur in one billion (10^9) device-hours, with 60% confidence. A billion device-hours is roughly equivalent to ten thousand devices operating for ten years as well as the PPM for one year [13,14,15]. The values of the PPM and FIT for EEPROM suggest that it is reasonable to invest in a self-repair mechanism for GALs which use the same technology.

In the next section of this paper we discuss the basic structure of a GAL and our fault model. In section 3, we discuss the architecture of our proposed self-test, self-repair system and its operation. In Section 4, we describe our evaluation methodology and show our simulation results. Finally in section 5, we give our conclusions and some suggestions for future work.

2 GAL architecture and fault modeling

A GAL has a fixed OR array (OLMC, Output Logic Macro Cell) and a programmable AND array. The re-programmable array is essentially a grid of conductors forming rows and columns with an electrically erasable CMOS (E^2CMOS) cell at each cross-point [1,16]. Each column is connected to the input of an AND gate, and each row is connected to an input variable or its complement. Any combination of input variables or complements can be applied to an AND gate to form any desired product term by programming each E^2CMOS cell to be either 'ON' or 'OFF'. A cell that is ON effectively connects its corresponding row and column, and a cell that is OFF disconnects the row and column. The cells can be electrically erased and reprogrammed. For further information on GALs, refer to [1,16]. Note that all the notations in this paper will be same as in [1].

0-7803-6657-3/01/$10.00 ©2001 IEEE

The most widely used fault model is the stuck-at fault model, which has been used for fault analysis and test generation in all types of logic circuits [17,18,19,20].

The cross-point stuck-at faults, which are located in E^2CMOS cells, are considered as our fault model because E^2CMOS cells of an AND array form a large percentage of GAL's area. As mentioned earlier, an E^2CMOS cells' array has the same structure and technology as used in EEPROMs. Each E^2CMOS cell (cross-point) of the programmable AND array of a GAL, which is located between a row (input line) and a column (product term), may be stuck to ON or OFF permanently, caused by an aging problem, radiation or by other facts referred in [1]. It is called a cross-point stuck-at-1 (simply, s-a-1) if the E^2CMOS cell of a particular cross-point, which should be programmed as OFF, is ON. If the E^2CMOS cell of a particular cross-point, which should be programmed as ON, is OFF, it is called a cross-point stuck-at-0 (simply, s-a-0).

The following assumptions will be used for the self-repair and evaluation methodologies. There are no faults in an initial state after the GAL manufacturing process. The primary input/output faults do not exist in a GAL, and also every AND gate input line does not have faults. For GALs, the cross-point stuck-at faults are considered much more probable than the stuck-at faults in wires. Thus, the cross-point faults (s-a-0, s-a-1) as defined above are only taken into account in this project. When an E^2CMOS cell is OFF, binary data 0 will be stored in memory. It can be called 'programmed as OFF.' This cell disconnects a primary input from an AND gate input. Binary data 1 will be on the AND gate input. When an E^2CMOS cell is ON, binary data 1 will be stored in memory. It can be called 'programmed as ON.' This cell connects a primary input from an AND gate input. To make it possible to replace a faulty column connected on the input of a particular AND gate, several extra AND gates are included in each OR gate (OLMC). When a fault is detected on the input(s) of a particular AND gate, the entire AND gate with its input columns is replaced with another AND column. The replacement may be an extra column that was included in the structure of the OR gate as described in [1], or it may be one that was acquired by an exchange and reuse of an existing column in the OR gate as described in the following section.

3 Test and repair architecture and methodology

3.1 Architecture

As shown on the right side of Figure 1 and described in [1], the digital system in our approach is a network of blocks realized as separate integrated circuits. Each block is realized as a self-repairable GAL. There are many GALs in a system, but for explanation purposes we concentrate on how to detect and repair faults in just one GAL. We assume that each block includes just one self-repairable GAL, and a global fault in the block is signaled with go/no-go signal when there are no more spare columns to be replaced. Note that rows will be represented as data inputs, and columns will be used as product terms on the rest of figures in this paper.

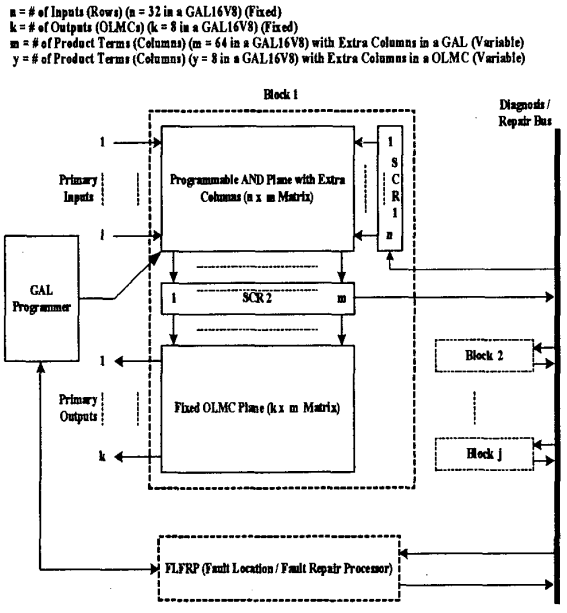


Figure 1. The design architecture for a self-repairable GAL

Each block is connected to a diagnosis/repair bus, which goes to the FLFRP (Fault Location/Fault Repair Processor). In the normal operation mode, the primary input data are fed into the GAL AND plane and the product terms of the AND plane are just transparent through SCR2 (Scan Register 2) into the OR. In the test mode, the FLFRP generates a test set through SCR1 (Scan Register 1) and this test vector scans each cross-point of the AND plane in the GAL. The test results are loaded into the SAP (State AND Plane). The SAP is compared with the MAP (Memory AND Plane) that has the original programmed data of the AND plane. If MAP and SAP data are different, then faults have occurred. Moreover, we know the exact locations of these faults. The structure of MAP and SAP arrays has been given in [1].

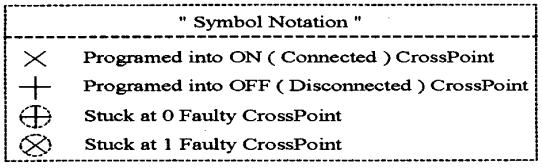
3.2 Test generation and fault diagnosis/fault location

We use a universal test set for detecting faults. Our test set is the well-known walking 0 approach. 'n' test vectors are required if there are 'n' inputs, because of only one '0' bit in a test vector. It will be shifted to right 'n-1' times.

Test vector set (Test pattern)

1	*2*	*3*	...	*n-1*	*n (inputs)*
0	*1*	*1*	...	*1*	*1*
1	*0*	*1*	...	*1*	*1*
1	*1*	*0*	...	*1*	*1*
.	.	.	...	.	.
1	*1*	*1*	...	*0*	*1*
1	*1*	*1*	...	*1*	*0*

The symbolic notation that will be used in the examples is the following:



A simple example with multiple faults in which an AND plane has 16 cross-points (4 inputs times 4 columns) is shown in Figure 2.

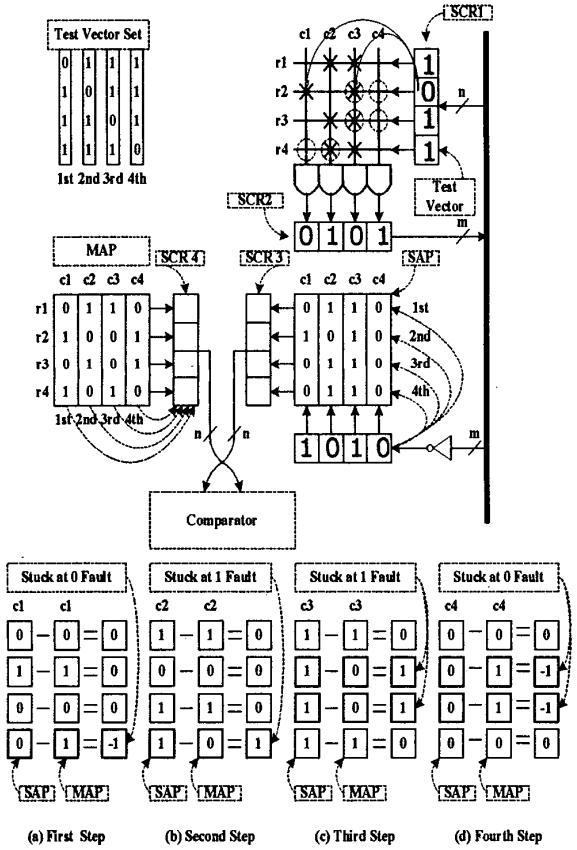


Figure 2. Fault diagnosis/location of an example with faults

There are no extra columns. This figure shows how to detect and locate faults. We will explain more detail about multiple faults in second row since others follow the same way. The original programmed data is in the MAP. For this row, there are stuck-at 1 and stuck-at 0 faults on the third and fourth columns, respectively. The test vector '1011' can detect and locate these faults. The second bit '0' of this test vector will make an output '0' on both the first and third AND gates, because the first cell is programmed as 'ON' and third cell is stuck-at 1. This makes a connection to the AND gate input from the second bit, '0' of the SCR1.

Actually, this second bit, '0' of the second test vector should make an output '0' on the fourth AND gate because the fourth cell was programmed as 'ON' as shown in the MAP. However, a stuck-at 0 fault has occurred and it makes a disconnection to the AND gate input from the second bit, '0' of the SCR1. (When an E^2CMOS cell is OFF, binary data 1 will be on the AND gate input as mentioned earlier.) Thus, second bit, '0' of the second test vector does not affect to the output of the AND gate. Finally, the SCR2 will have '0101' and the second row of SAP will have '1010' through the inverter. Now, it will be compared with data from the MAP. In Figure 2 (c), '1' from the SAP minus '0' from the MAP makes '1', which means that stuck-at 1 occurred in that E^2CMOS cell. If the result of the comparison operation is '-1', a stuck-at 0 fault has occurred in that E2CMOS cell, as shown in Figure 2 (d). The result '0' means no fault occurred, as shown in Figure 2 (a) and (b). Thus, all multiple stuck-at faults are detected and their exact locations are determined using the universal test vectors.

3.3 An advanced self-repairing methodology, "column re-use with extra columns"

An advanced self-repairing method (column re-use method) and NC (Next Column) register are introduced in this section.

For our column re-use method, we need an NC register to keep track of the status of each column in each OR gate in the GAL. As shown in Figure 3, each row represents an OR gate, and a column represents the input of an OR gate. There are '*y*' columns in the NC since each OR gate has '*y*' columns. There are 'k' ORs in this NC. In the GAL16V8, '*k*' will be 8 since it has 8 OLMCs. The intersection of a row and a column being '0' in the NC means that the corresponding column (AND gate) of that OR is being used. If it is '1', then that column is useful for reprogramming a new logic function, replacing, or re-using a faulty column. It can be also considered as an available extra column to repair faulty columns. The '-1' means that the column corresponding to that location of the AND plane can not be used to repair a faulty column or to reprogram a new logic

function. The '-2' means that if this column becomes faulty, it can only be replaced with an extra column, because it is assumed that the column was already repaired with the Column Re-Use method, and thus it can be only repaired with the Column Replacement method rather than with the Column Re-Use method. For more detailed information about the Column Replacement method, refer to [1].

Primary Outputs	c 1	c 2	----------	c y-1	c y
or1	0	0	----------	1	1
or2	-2	0	----------	0	1
or (k - 1)	-1	0	----------	1	1
or (k = 8)	1	1	----------	1	1

0: Column being used 1: AvailbleColumn to replace or re-use
-1: Unavailable Column to replace or re-use -2: Available Column to replace only
c: Column in a OR gate or: OR Group

Figure 3. Structure of the NC register

The column re-use method actually does not require extra columns, but it is called "column re-use with extra columns" (or simply column re-use) since it operates on an AND plane that has in addition some extra columns. For self-repairing of a faulty column, this method first attempts to exchange the faulty column with a column which is already in use in the OR gate and has the correct programming for the faulty column. The column acquired in this exchange can then be reprogrammed as needed for its new location. This test can be achieved by comparing the column in the SAP that needs to be exchanged with all column information in the MAP. For instance, if the data in c1 of the SAP is the same as the data in c4 of the MAP, then these two columns can be exchanged in this method. It means that original c1 data is stored in the c4 location of the MAP, and the original c4 data is stored in the c1 location of the MAP by reprogramming the original c1 data on the location c4 in the AND plane. After that, the corresponding cell (c1) in the NC register will be updated as '–2.' This is to mark that the column replacement method should be applied to that column (c1), instead of the column re-use method, if some faults are discovered in this column (c1) in a later test operation mode.

An example in which multiple faults have occurred in an AND plane is shown in Figure 4. There are 4 rows (input lines) and 8 columns (product terms) in an OR group of an AND plane. The originally programmed data (data of the MAP) is the same as in the previous example described in Section 3.2.

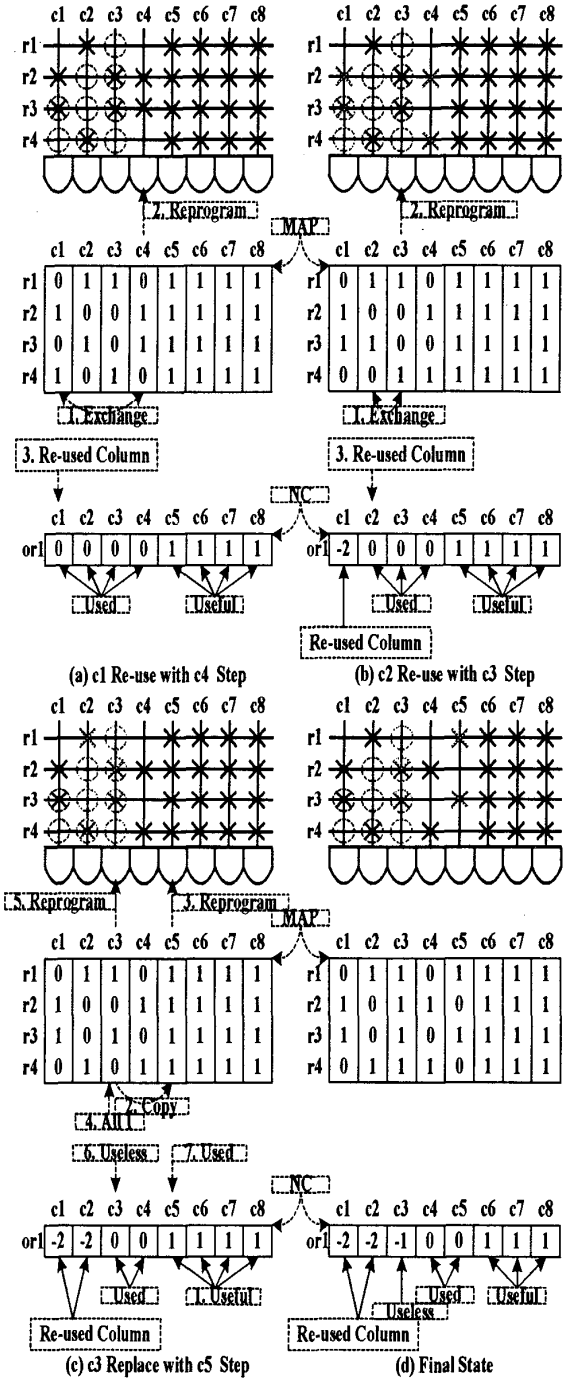


Figure 4. An example of column re-use method with multiple faults

All cross-points, which have not been programmed, are initialized as logic value '1'. There are 4 extra columns (c5, c6, c7, and c8) initialized as '1'. Thus, the MAP has logic value '1' on each location corresponding to those columns. The SAP was already obtained before applying the self-repairing method; thus it is not shown in this example. However, it can be described as just the actual state of this AND plane, as shown in Figure 4. In Figure 4 (a), the MAP shows the initial state that is the information programmed for c1, c2, c3, c4 and the initial state of extra columns for c5, c6, c7, and c8. The NC register has initial data which are '0' for c1, c2, c3, c4 since they are being used, and are '1' for c5, c6, c7, c8 since they are useful for replacing or re-using faulty columns.

There are multiple faults in an OR group which has 8 product terms, as shown in this figure. The numbered dotted boxes describe the sequence of the column re-use in the self-repairing method. Four steps are required to repair the faults in this example. In Figure 4 (a), c1 of the SAP stores '0110' since the intersection of c1 & r3, and c1 & r4 have s-a-1 and s-a-0 faults, respectively. It can be shown to be different from the c1 of MAP, '0101'. The c1 '0110' of the SAP finds which data is the same as in MAP. The c4 '0110' of MAP is found as the same data of the c1 '0110' of the SAP. Thus, they can be exchanged in MAP. The c4 '0101' exchanged data from c1 of MAP will be programmed in c4 of the AND plane. Now, the c1 of MAP has '0110' as the original data for c4. Although the personalities of these two columns are switched with each other, the function of this OR gate will not be changed. The c1 of the NC register will be updated as '-2' which shows that this column has been re-used, so that it must be only replaced with a new extra column if this column, c1, becomes faulty again. The same procedure will be done with the c2 '1001' of SAP in Figure 4 (b). In Figure 4 (c), the c3 '0110' of the SAP is the same as the c1 '0110' of the MAP, but they cannot be exchanged since the c1 of the NC register is marked as '-2'. Thus, the c3 '1010' of the MAP must be replaced with a new extra column, c5. The final state is shown in Figure 4 (d).

4 Evaluation methodology and simulation results

In this section, simulation results are analyzed and compared with column replacement method to evaluate the column re-use method and to prove that a self-repairable GAL using column re-use method will last longer in the field.

Assumptions for our evaluation methodology are the following. Each cross-point in the AND plane has the same probability of stuck-at faults (s-a-0 and s-a-1) to occur. The failure of one cross-point does not affect the likelihood that another cross-point will fail. Thus, the failure of one cross-point is assumed to be independent of the failure of another cross-point, likewise for each column and each OR gate. In GAL operation without self-repair, if at least one cross-point stuck-at fault appears in a column, then that column will be faulty. This means that if at least one used OR gate having a faulty column is faulty in a GAL, then that GAL is useless.

The PPM, 0.05% and FIT, 5.00% of EEPROM are adopted as the PPM and the FIT of a GAL in our simulation, referred to in section 1. In our simulation, the MAP is generated under 100% AND array utilization; thus all E^2CMOS cells are programmed using a random number generator. After creating the description of the MAP, the faults are simulated in an AND array by using a random number generator; the maximum number of cross-points that have faults at a time are limited to be less or equal to 5, 10, 20, 50, 100, 1000, and 2048. It is assumed that extra columns can also have faults. The extra columns will be added in multiples of 8 since there are 8 OLMCs in a GAL and each OLMC has the same number of extra columns. The results of the failure rate analysis are only valid under assumption that the assumed stuck-at fault model adequately represents all physical defects that can occur in a GAL device. Table 1 displays average looping time until a GAL is useless after running simulation in 100,000 times at each case.

These results show that with more extra columns, longer survival times in the field can be obtained. The results on 0.05% failure rate of a GAL are better than on 5% failure rate of a GAL. In 0 number of extra columns case, using re-use method (14591, 133) makes a self-repairable GAL last over 5 times longer than using replacement method (2708, 25) in [1] under both 0.05% and 5% failure rate. It means that re-use method makes a self-repairable GAL more reliable even without any spare columns.

# of Extra Columns	# of Faults Limit													
	≤5		≤10		≤20		≤50		≤100		≤1000		≤2048	
	Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	14591	133	9631	83	5760	53	3552	32	2786	25	2230	20	2206	20
8	35224	323	20309	185	11703	106	6087	56	3972	36	2322	21	2247	20
16	56254	512	31620	288	17554	161	8548	79	5312	49	2414	22	2284	21
24	76653	702	43053	395	23529	214	11009	101	6566	61	2491	23	2323	21
32	97962	899	54567	499	29686	271	13496	124	7878	73	2598	24	2362	22
40	118986	1096	66285	608	35884	328	16105	148	9219	85	2697	24	2398	22
48	146972	1297	78062	714	42158	386	18770	173	10564	98	2816	25	2448	22
56	163471	1500	90712	830	48607	443	21519	198	11941	110	2922	27	2493	23
64	186512	1707	103019	945	55274	506	24264	222	13356	123	3054	27	2562	23

Table 1. Average looping time of simulating the re-use methodology

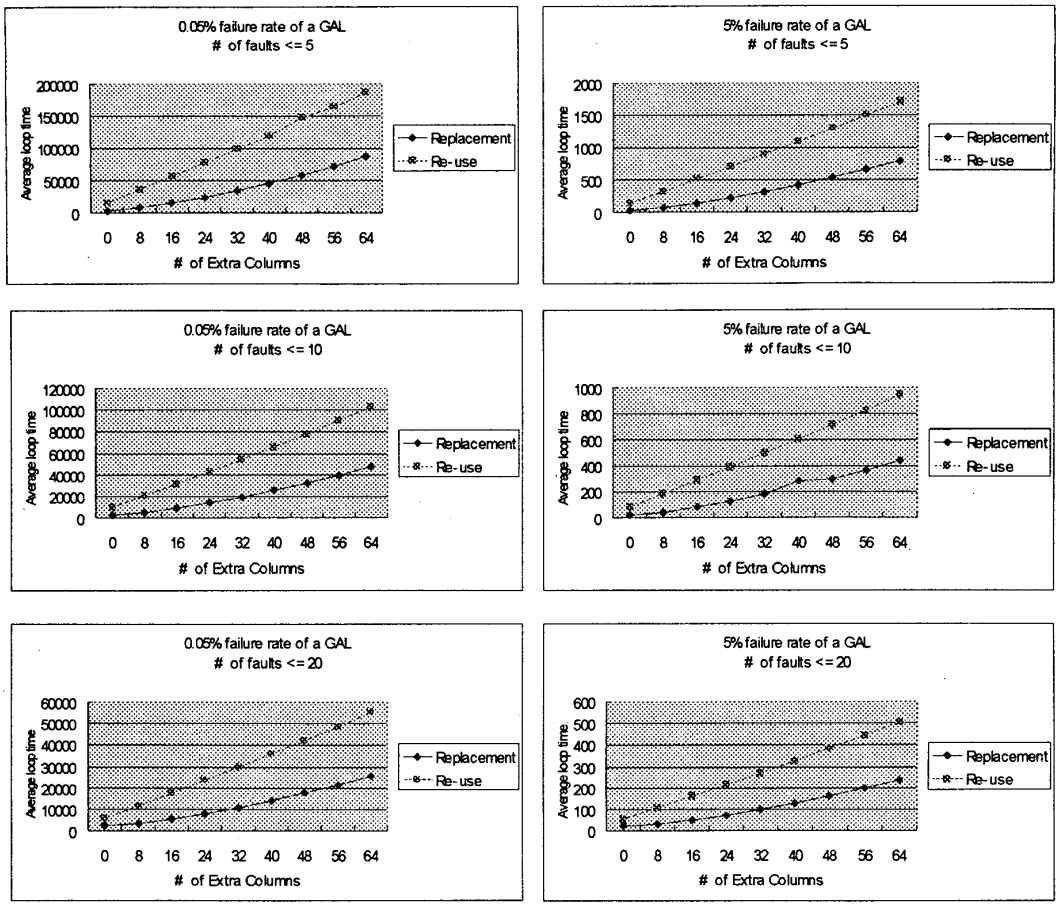


Figure 5. Comparison of average looping time for re-use method and replacement method in less or equal to 5, 10, 20 number of failure limits (a) Average looping time under 0.05% failure rate of a GAL, (b) Average looping time under 5.00% failure rate of a GAL

If the average looping time can be converted to the lifetime of a GAL, it would provide an indication how many extra columns a GAL should have in order to guarantee certain reliability and lifetime.

Finally, as shown in Figure 5, a self-repairable GAL using the re-use method will last longer than one using the replacement method or no self-repair method in the field. Figure 5 only shows three cases that the number of faults limit is less or equal to 5, 10, and 20 under both 0.05% and 5% failure rate since for other cases that the number of faults limit is less or equal to 50, 100, 1000 and 2048, under both, failure of a GAL will be very rare. The simulation results for the re-use method are better than the results for the replacement method in all cases. (Please refer to [1] for the detailed simulation results for the replacement method.) Figure 5 (a) is different scale from Figure 5 (b) vertically.

Now, we consider the performance defined as the ratio of looping time divided by the ratio of area overhead at each case. In the measurement of area overhead, the LSI Logic Corporation's 0.5-micron LCA/LEA500K array-based products were used for synthesis [21].

All area measurements are expressed in cell units, excluding the interconnection wires. This measurement is the same as done in [1]. We refer to [1] for the area overheads and ratios.

Table 2 shows the performance of a self-repairable GAL, as a function of the number of extra columns. This table represents the ratio of efficiency versus cost factor, so a larger number represents better performance. This result shows that all cases when the number of fault limits is in columns "less than equal" 50, 100, 1000, and 2048 are not practical. The case of adding 64 extra columns for the assumption of the number of faults, less than or equal to 5 is the most efficient from Table 2, both

under the 0.05% failure rate (5.63) and under the 5% failure rate of a GAL (5.65).

# of Extra Columns	# of Faults Limit													
	≤5		≤10		≤20		≤50		≤100		≤1000		≤2048	
	Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL		Failure rate of a GAL	
	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%	0.05%	5%
0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
8	1.75	1.76	1.53	1.62	1.47	1.45	1.24	1.27	1.04	1.04	0.75	0.76	0.74	0.72
16	2.57	2.57	2.19	2.31	2.03	2.03	1.61	1.65	1.27	1.31	0.72	0.73	0.69	0.70
24	3.22	3.24	2.74	2.92	2.50	2.48	1.90	1.94	1.45	1.50	0.69	0.71	0.64	0.64
32	3.81	3.84	3.22	3.41	2.93	2.90	2.16	2.20	1.61	1.66	0.66	0.68	0.61	0.63
40	4.31	4.36	3.64	3.88	3.30	3.28	2.40	2.45	1.75	1.80	0.64	0.63	0.58	0.58
48	4.99	4.83	4.01	4.26	3.62	3.60	2.61	2.68	1.88	1.94	0.62	0.62	0.55	0.54
56	5.21	5.25	4.38	4.65	3.93	3.89	2.82	2.88	2.00	2.05	0.61	0.63	0.53	0.53
64	5.63	5.65	4.71	5.02	4.23	4.21	3.01	3.06	2.11	2.17	0.60	0.59	0.51	0.51

Table 2. Performance of a self-repairable GAL

5 Conclusions and future works

A method for self-testing and self-repairing digital circuits has been presented. A GAL was used as an example here, but the method is applicable to other devices such as FPGAs. The basic principle of our approach is that a fault processor periodically tests the AND plane of the GAL and diagnoses any faults in the AND plane. If a faulty column is found, an attempt is first made to exchange that column with another column that needs the programming found in the faulty column. The column gained in this exchange is then reprogrammed to replace the faulty column. If column re-use is not possible, the faulty column must be replaced with an extra column built in the OR gate. When the test/repair cycle is complete, the circuit returns to normal operation. Assuming a stuck-at fault model, faulty columns can be diagnosed and located with a universal test vector set. Our method allows the repair all multiple faults up to the point where no more columns are available for re-use or replacement. Our simulation results show that a self-repairable GAL using the re-use method will last longer in the field. Data presented also shows how many extra columns a GAL should have in order to achieve a desired reliability and device lifetime. With this data and estimates of the required areas, it is possible to estimate an ideal point, where the maximum reliability can be achieved with the minimum cost.

This paper is an initial attempt to present some of the possible approaches to design for self-repair. Although we present here only SOP (Sum of Product) type EPLDs, the method is applicable to other devices such as FPGAs. The development and comparison of various self-repairable programmable logic structures is our long-term research objective.

References

[1] C. H. Lee, M. A. Perkowski, D. V. Hall, and D. S. Jun, "Self-Repairable EPLDs: Design, Self-Repair, and Evaluation Methodology," *The Second NASA/DoD Workshop on Evolvable Hardware,* pp.183-193, July 2000.

[2] A. Avizienis, "Hundred Year Spacecraft," *The First NASA/DoD Workshop on Evolvable Hardware,* pp.233-239, July 1999.

[3] E. Drexler, "Engines of Creation," *Anchor Books,* 1986.

[4] G. Tempesti, D. Mange, and A. Stauffer, "A Self-Repairing FPGA Inspired By Biology," *The Third IEEE International On-Line Testing Workshop,* pp.191-195, 1997.

[5] D. Mange, M. Goeke, D. Madon, A Stauffer, G. Tempesti, and S. Durand, "Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Array with Self-Repair and Self-Reproducing Properties," *In Towards Evolvable Hardware*, Springer-Verlag, pp.197-220, 1996.

[6] D. Mange and A. Stauffer, "Introduction to Embryonics: Towards New Self-Repairing and Self-Reproducing Hardware Based on Biological-like Properties," *Artificial Life and Virtual Reality*, John Wiley, pp.61-72, 1994.

[7] D. L. Ostapko and S. J. Hong, "Fault Analysis and Test Generation for Programmable Logic Array (PLA)," *IEEE Trans. on Computers, Vol. C-28, No. 9*, pp.617-627, September 1979.

[8] K. S. Ramanatha and N. N. Biswas, "An On-Line Algorithm for the Location of Cross Point Faults in Programmable Logic Arrays," *IEEE Trans. on Computers, Vol. C-32, No. 5*, pp.438-444, May 1983.

[9] J. E. Smith, "Detection of Faults in Programmable Logic Arrays," *IEEE Trans. on Computers, Vol. C-28, No. 11*, pp.845-853, November 1979.

[10] H. Fujiwara and K. Kinoshita, "A Design of Programmable Logic Array with Universal Tests," *IEEE Trans. on Computers, Vol. C-30, No. 11*, pp.823-829, November 1981.

[11] W. Daehn and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays,"

IEEE Trans. on Computers, Vol. C-30, No. 11, pp.829-833, November 1981.

[12] R. Treuer, H. Fujiwara, and V. K. Agarwal, " Implementing a Built-In Self-Test PLA Design," *IEEE Design and Test*, pp.37-48, April 1985.

[13] K. Seshan, T. J. Maloney, and K. J. Wu, "The Quality and Reliability of Intel's Quarter Micron Process," *Intel Technology 3rd quarter Journal, http://www.intel.co.uk/technology/itj/q31998/articles/*, July 1998.

[14] National Semiconductor Corporation, "Quality Network – Failure Rates of Major Processes at National Semiconductor, National Semiconductor Failure Rate Trends, and National Semiconductor Reliability," *http://207.82.57.10/quality/pages*, May 1999.

[15] D. Sellers, "Quality and Reliability," *Quality and Reliability Hand Book – Space Electronics Inc., http://www.spaceelectronics.com/Spaceprod/reliability/qr.html*, June 1999.

[16] Lattice Semiconductor Corporation, "Lattice Semiconductor Data Book 1996," *Lattice Semiconductor Corporation*, pp.365-392, 1996.

[17] J. P. Hayes, "Fault Modeling," *IEEE Design & Test of Computers*, pp. 88-95, April 1985.

[18] W. Maly, "Realistic Fault Modeling for VLSI Testing," *Proc. of the 24th ACM/IEEE Design Automation Conference*, pp. 173-180, 1987.

[19] R. Nair, S. M. Thatte, and J. A. Abraham, "Efficient Algorithms for Testing Semiconductor Random-Access Memories," *IEEE Trans. on Computers, Vol. C-27, No. 6*, pp. 572-576, June 1978.

[20] F.G. Cockerill, "Quality Control for Production Testing," *1982 International Test Conference*, pp. 308-314, November 1982.

[21] LSI Logic Corporation, "LCA/LEA500K Array-Based Products Databook," *Document DB04-000002-03, Fourth Edition*, May 1997.