A Novel Test Flow for One-Time-Programming Applications of NROM Technology

Mango C.-T. Chao, Ching-Yu Chin, Yao-Te Tsou, and Chi-Min Chang

***Abstract*—The NROM technology is an emerging non-volatile-memory technology providing high data density with low fabrication cost. In this paper, we propose a novel test flow for the one-time-programming (OTP) applications using the NROM bit cells. Unlike the conventional test flow, the proposed flow applies the repair analysis in its package test instead of in its wafer test, and hence creates a chance for reusing the bit cells originally identified as a defect to represent the value in the OTP application. Thus, the proposed test flow can reduce the number of bit cells to be repaired and further improve the yield. Also, we propose an efficient and effective estimation scheme to predict the probability of a part being successfully repaired before packaged. This estimation can be used to determine whether a part should be packaged, such that the total profit of the proposed test flow can be optimized. A series of experiments are conducted to demonstrate the effectiveness, efficiency, and feasibility of the proposed test flow.**

***Index Terms*—NROM, repair rate estimation, test flow.**

I. Introduction

THROUGH several technology evolutions, the floating-gate-based memory [1] has been the mainstream of the non-volatile-memory market in the past. However, its leadership is challenged by the emerging charge-trapping-based memories, such as MNOS [2], SONOS [3]–[5], and NROM [6]–[8], which can offer better value stability, simpler fabrication process, and less process adders to a logic process. Among those charge-trapping-based memories, the NROM technology provides two bits per device with a low fabrication cost and overcomes the reliability issue by using thicker oxide cladding layers. Those advantages make the NROM technology an attractive solution in current non-volatile-memory market.

MXIC [9] applies the NROM technology to manufacture stand-alone read-only-memory (ROM) applications, such as video/audio code, storage chip, game console chip, and system bias of 3C products. This NROM-based ROMs support the storage up to 2 Gb with a 100 ns random-access time and a 3.0–3.6 V supply voltage, which can match the applications of traditional mask ROMs. The NROM-based ROMs provided by MXIC not only inherit the advantage of high density from the NROM technology but also reduce its manufacturing cost by lowering the process requirement from multiple-time programmability to one-time programmability. Thus, the NROM-based ROMs can provide higher data density with economic price compared to the traditional mask ROMs. Also, NROM-based ROMs need not to know the customer's code when manufactured, and hence no change on its photomasks need to be made when the customer's code changes. This property can lower the cost for a small-volume order and allow the customers to revise the code in their next-version products without extra photomask charge. In addition, blank NROM-based ROMs can be manufactured in advance. Once an order is received, the code can be directly programmed to the blank NROM-based ROMs, which can significantly shorten the delivery time.

With all above advantages, however, the NROM-based ROMs require a more complicated test flow compared to the traditional ROM testing, in which the complete contents of all addresses are simply read out and compared [11]–[13]. The conventional test flow for NROM-based ROMs can be divided into two parts, the wafer test followed by the package test. In the wafer test, we test whether the default value of each word can be read out correctly. If any failed word exists, we attempt to repair those failed words with the spare rows and columns. The dies which cannot be successfully repaired are identified as failed dies and then discarded. The passing dies will then be packaged and wait for the code from customers. After the code is programmed into a packaged NROM-based ROM, the package test is applied to check whether the programmed code can be read out correctly at speed. The NROM-based ROMs passing the package test will be delivered to customers.

In our previous work [10], a novel test flow has been proposed for NROM-based ROMs. Its basic idea is to perform the repair process in the package test instead of in the wafer test. An NROM bit which fails to correctly represent the default value may be able to correctly represent its opposite value, which may probably be the value requested in the customer's code. In other words, the bit cell which is identified as a *physical-level defect* in the conventional wafer test may not be a *data-level defect* for customer's code. Thus, by repairing the data-level defects in the package test, the yield can be further improved. However, we cannot pass and package all the dies in the wafer test since each package costs. We can only pass the dies which have a decent chance to be successfully repaired in the package test. To estimate this successful-repaired probability in the wafer test is a challenge since the customer's code is not yet known at this stage. In [10], a mathematical model has been proposed to efficiently and effectively estimate the probability that the resulting data-level defects can be successfully repaired based on only the information of physical-level defects obtained from wafer test.

In this paper, we will first restate the new proposed test flow in [10]. Next, we will provide the detailed derivation of the mathe-

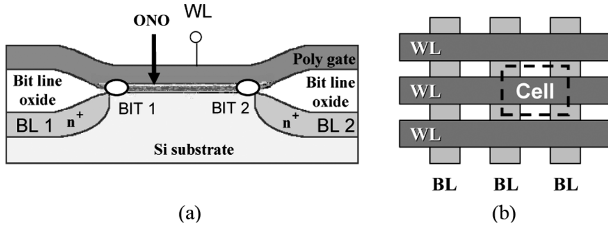
Manuscript received April 20, 2010; revised July 16, 2010; accepted September 21, 2010. Date of publication December 23, 2010; date of current version October 28, 2011.

M. C.-T. Chao, C.-Y. Chin, and C.-M. Chang are with the Electronics Engineering Department, National Chiao Tung University, Hsinchu, Taiwan 300 (e-mail: mango@faculty.nctu.edu.tw; lwaysrain.gr@gmail.com; cmc.ee95g@nctu.edu.tw).

Y.-T. Tsou is with Macronix International Company, Application-Specific Memory Division, Hsinchu, Taiwan, (e-mail: YTTsou@mxic.com.tw).

Digital Object Identifier 10.1109/TVLSI.2010.2087044

1063-8210/$26.00 © 2010 IEEE



(a) (b)

Fig. 1. Cross section view and top view of an NROM cell [6].

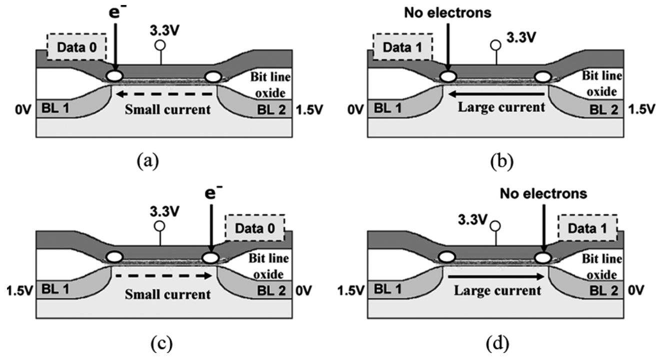


Fig. 2. Read mechanism for an NROM cell. (a) Read Bit 1 (data 0). (b) Read Bit 1 (data 1). (c) Read Bit 2 (data 0). (d) Read Bit 2 (data 1).

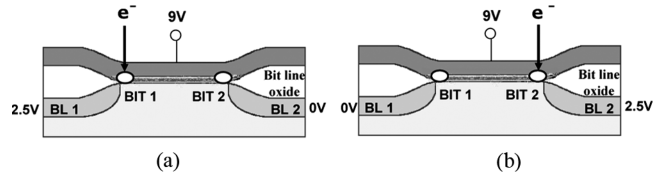


Fig. 3. Program mechanism for an NROM cell. (a) Program Bit 1 to 0. (b) Program Bit 2 to 0.

matical estimation to the above successful-repaired probability. We will further analyze the complexity and the optimality of this mathematical estimation and conduct additional experiments to validate its accuracy. In addition, we will discuss the impact of setting different threshold probabilities (p_th), used to determine whether a die should be packaged or discarded after wafer test. A novel mathematical model will also be developed to efficiently identify the optimal setting of this threshold probability such that the overall packaging cost and yield can be balanced. At last, the overhead of the test-application time, the extra ATE expense imposed by the proposed test flow, and the reliability issues will be further discussed. Overall, the experimental result, based on an industrial NROM-based ROM design, clearly shows the yield improvement of the proposed test flow as well as the high accuracy of the proposed mathematical models.

II. Background

A. *Characteristics of an NROM Cell*

Fig. 1 shows the cross-section view of an NROM cell, which is an n-channel MOSFET where the gate dielectric is replaced by an oxide-nitride-oxide (ONO) stack used for trapping electrons [6]. Each NROM cell stores two bits. The value of each bit is determined by whether enough of electrons are trapped at one end of the nitride layer adjacent to the n+ junctions, as indicated by $BIT1$ and $BIT2$ in Fig. 1(a). An NROM cell contains one word-line (WL) and two bit lines $(BL1,\ BL2)$. As shown in Fig. 1(a), the word-line is the polysilicon and the two bit lines are the two diffusions. Fig. 1(b) shows the top view of an NROM memory array.

To read the data of $BIT1$ in our NROM-based ROMs, we apply 3.3 V at WL, 0 V at $BL1$, and 1.5 V at $BL2$. If more electrons are trapped in $BIT1$, the sensed current from $BL2$ to $BL1$ becomes smaller, meaning that the threshold voltage of $BIT1$ becomes larger. A sensed current smaller than the reference current is defined as a value 0. A sensed current larger than the reference current is defined as a value 1. Similarly, to read the data stored in $BIT2$, we apply 3.3 V at WL, 1.5 V at $BL1$, and 0 V at $BL2$ and then sense the current from $BL1$ to $BL2$. Since the current directions for reading $BIT1$ and $BIT2$ are different, we need one current-sense amplifier at each bit line. The reference current ranges from 20 μA to 30 μA depending on products. Fig. 2 illustrates the read operations for $BIT1$ and $BIT2$ with value 0 and value 1, respectively.

Note that the electrons trapped at one end of the ONO stack will not affect the read mechanism for the other end. For the NROM devices introduced in [8], read is preformed after a polarity reversal, Reverse Read, interchanging the roles of the source and the drain, relative to programming, with a typical drain-source voltage of 1.6 V. This voltage punches through the potential barrier induced by charges trapped over the drain junction, thus the transistor current is only dictated by charges trapped over the source junction and two-bit separation in read is achieved. For more detailed device physics about NROM, please refer to [8].

When an NROM cell is fabricated, both $BIT1$ and $BIT2$ do not store electrons initially, representing a value 1. Electrons can be injected into the ONO stack through the channel hot electron injection (CHE) and trapped in the nitride layer of the ONO stack, representing a value 0. Fig. 3 illustrates the operations to write a value 0 to $BIT1$ and $BIT2$ in our NROM-based ROMs, respectively.

B. *Conventional Test Flow for NROM-Based ROMs*

A conventional test flow for NROM-based ROMs consists of two main procedures: a wafer test followed by a package test. The objective of the wafter test is to check whether each bit cell in an NROM-based ROM can function correctly. The objective of package test is to check whether the programmed code in an NROM-based ROM can be correctly read out. An NROM-based ROM passing the wafer test is then packaged and considered as a blank ROM. After the code is received from customers and programmed into blank ROMs, the package test is applied before the programmed ROMs are sent to customers. The repair mechanism is applied in the wafer test if defective words are identified.

Fig. 4 first lists the steps of both the conventional wafer test and package test for NROM-based ROMs. The detail of each step is discussed as follows.

1) DC Parametric Test: The DC parametric test here includes open/short test, leakage test, static current test, and operating current test. The open/short test checks the connectivity between pads in the wafer test (or pins in the packaged test). The leakage

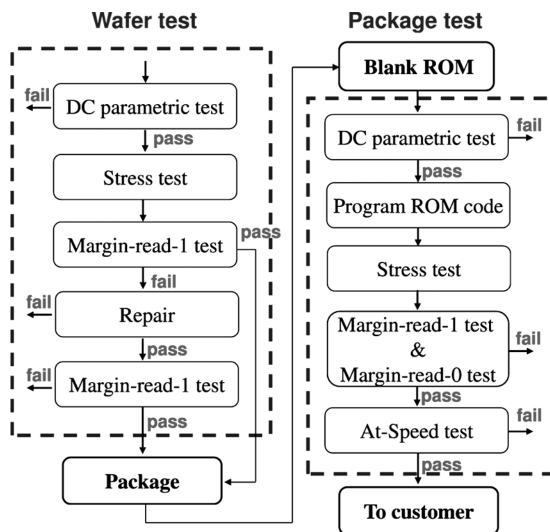


Fig. 4. Conventional test flow for NROM-based ROMs.

test checks whether an unacceptably large current exists on each pad or pin. The static current test and operating current test check the power consumption at standby mode and operating mode, respectively.

2) Stress Test: The stress test supplies a high voltage to each word-line, which can speed up the failure of unhealthy memory cells. Note that those induced defective cells are not identified in this stress test but will be identified in the later tests.

3) Margin-Read-1 & Margin-Read-0 Test: The margin-read-1 (or 0) test is to check whether an NROM bit can represent a "quality" value of 1 (or 0). Note that the value of an NROM bit is determined by the threshold voltage of the corresponding NROM MOSFET. A quality value of 1 (or 0) means that its threshold voltage is smaller (or larger) than the read voltage defined in the specification by an user-specific margin, such that its sensed current can be easily differentiated by the sense amplifier. Therefore, in the margin-read-1 test, we read value 1 by applying the smallest read voltage defined in the specification minus a voltage margin. In the margin-read-0 test, we read value 0 by applying the largest read voltage defined in the specification plus a voltage margin.

4) Repair: This repair step applies a repair algorithm to determine whether and how the defective cells can be replaced by the spare rows and columns. The redundancy design used in our NROM-based ROMs is a two-dimensional architecture containing both spare rows and spare columns. The repair algorithm used in our test flow is a repair-most algorithm based on a compressed defect map. Note that the address-folding technique used in our NROM-based ROMs is the adjacent folding [14], meaning that the bit-cells in a word are serially placed together in the layout. As a result, the column repair is performed in units of words (same as row repair), meaning that all bits in a word will be replaced simultaneously once a bit in that word fails. On the other hand, if the distributed folding [14] is used, all the ith bit-cells of different words at the same word-line are placed serially to one another in the layout, meaning that to repair the ith bit of a word with a spare column will also replace the ith bit-cell of the other words at the same word-line. But all other bits in the same remain the original ones.

5) At-Speed Test: At-speed test is to check whether the programmed code can be correctly read out at the defined frequency.

As Fig. 4 shows that the conventional wafer test first performs the DC parametric test, stress test, and the margin-read-1 test in order. Note that the default value of all bits in a newly manufactured NROM die is 1. Then we can obtain the defect map from the result of margin-read-1 test, based on that the repair analysis is applied. Last, another margin-read-1 test is applied on the repaired rows and columns.

In the conventional package test, the DC parametric test is first performed and then the customer's code is programmed into a package NROM-based ROM. After the stress test, we perform the margin-read-0 test and margin-read-1 tests to ensure that each programmed bit can represent a quality value 0 or value 1. Last, the at-speed test is applied.

Note that in the conventional wafer test, we do not check whether a bit cell can be programmed to a quality value 0. We only check whether a bit can represent a quality value 1 and only repair the bits which cannot represent a quality value 1, which is the initial value of an NROM bit. The main reasons of only checking value 1 are listed below.

First, the threshold voltage of a bit cell representing a value 1 is closer to the read voltage defined in the specification than that representing a value 0 (as will be shown in Fig. 6). Thus, it is more likely that a faulty bit cell fails to represent a value 1 rather than a value 0. Second, from the statistics of the collected manufacturing data, most bits failing to represent a quality value 0 result from some abnormal leakage or open/short defects on bit-lines or word-lines, not on the bit cells themselves. Those defects can already be detected during the margin-read-1 test or the DC parametric test. Third, a bit cell representing a quality value 1 can also represent a quality value 0 in our NROM process for most cases. Also, it requires significant amount of time to program all the bits to value 0 and apply another margin-read-0 test. In addition, if more write operations (program or erase) are applied to a bit cell, the probability of its next write operation being successful becomes smaller. Therefore, we attempt to avoid any unnecessary write operations during the wafer test.

According to the internal statistics of MXIC's mature product lines of NROM-based ROMs, the packaged parts passing the package test are usually more than 95%. Most yield loss in the conventional test flow results from the wafer test, which we would like to further reduce in this paper. In the conventional test flow, the repair mechanism is applied in the wafer test, which identifies the defective cells based on the margin-read-1 test. If the defective cells cannot be repaired by the spare rows and columns, the NROM die is then discarded. However, an NROM bit which cannot represent a quality value 1 may be able to represent a quality value 0, which may happen to be the value required in the customer's code. Since an NROM-based ROM is going to be programmed once, a *physical-level defect* identified during the wafer test may no longer be a *data-level defect* after the customer's code is programmed. Thus, if those physical-level defects identified in the conventional wafer test can be reused to represent a value 0 in the programmed code, we

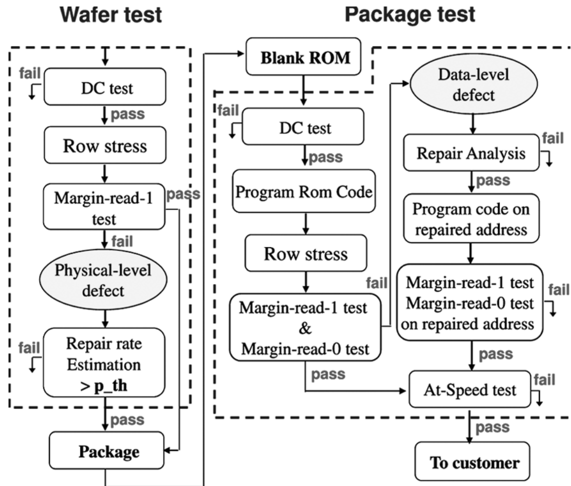


Fig. 5. Proposed test flow for NROM-based ROMs.

can reduce the number of defective cells which really need to be repaired, and in turn improve the yield of the NROM-based ROMs. Section III will introduce a novel test flow for NROM-based ROMs improved from this conventional test flow.

III. Proposed Test Flow for NROM-Based ROMs

A. Overall Flow

Fig. 5 shows the proposed test flow for NROM-based ROMs. The key idea of the proposed test flow is to apply the repair analysis after the code is programmed into an NROM-based ROM, such that the repair analysis can be performed based on data-level defects instead of physical-level defects. In the proposed package test, we program the customer's code to a blank ROM and then apply margin-read-0 and margin-read-1 tests to identify the data-level defects, i.e., the words which cannot successfully represent the corresponding value of the code. Based on those collected data-level defects, the repair analysis is then applied. After an NROM-based ROM is successfully repaired, we program the corresponding code to the replaced rows and columns and check their correctness with another margin-read-0 and margin-read-1 tests.

In the proposed wafer test, we only apply a margin-read-1 test to identify the physical-level defects, i.e., the words which cannot successfully represent the initial value of 1. Based on the collected physical-level defects, we apply a quick estimation scheme to predict the probability that the tested part can be successfully repaired after the code is programmed into it in the package test. If the predicted successful-repaired probability is higher than the user-specified threshold, denoted as p_th, then we package the part. Otherwise, we discard it.

Compared to the conventional test flow, the advantage of the proposed flow is that we could represent a value 0 in the customer's code by using the cells which cannot represent a quality value 1 in the wafer test. It can reduce the number of words required to be repaired and hence increase the yield. However, to make the proposed flow practical and effective, the following three premises need to sustain.

1) An NROM bit which cannot successfully represent a quality value 1 can indeed represent a quality value 0.
2) Most defects are random single defects and the number of defective bits in a word is small.
3) An effective and efficient estimation to the successful-repaired probability can be developed.

If the first premise does not hold, the number of data-level defects will be the same as the number of physical-level defects, meaning that our proposed test flow cannot gain any yield compared to the conventional one. If the second premise does not hold, it becomes more likely that a physical-level defective word still need to be replaced since the repair scheme is performed based on the unit of words. As long as one bit in a word cannot correctly represent the code's value, the whole word still need to be replaced. If the third premise does not hold, either too many non-repairable dies are packaged or too many repairable dies are discarded. Both cases may damage the overall profit of the proposed test flow.

In the following two subsections, we first conduct experiments on real NROM-based ROMs to validate the first and second premises, respectively. We will then detail our mathematical estimation used in the wafer test to predict the successful-repaired probability in the package test in Section IV. The accuracy of this estimation will be demonstrated by the experiments in Section V.

B. Representing Value 0 With Physical-Level Defective Bits

As discussed in Section II-A, the value of an NROM bit depends on the threshold voltage for the corresponding current direction. For value 1, its threshold voltage is low and hence the sensed current is large. For value 0, its threshold voltage is high and hence the sensed current is low. In the following experiment, we measure the threshold voltage of each NROM bit over different dies collected from different lots. We first apply different word-line (WL) voltages from 1.5 to 5.5 V using a sweeping step of 0.1 V, and collect their sensed logic values. The threshold voltage of an NROM bit is then obtained as the lowest word-line voltage that can sense a value 1.

The left-hand side of Fig. 6 first shows the probability distribution of the threshold voltages measured from the NROM bits with the initial value 1. Group 1 and Group 2 in Fig. 6 indicate the probabilities of NROM bits passing and failing the margin-read-1 test, respectively. We further program the NROM bits failing the margin-read-1 test to the value 0 and then measure their threshold voltages, whose distribution is indicated by Group 3. The specification for the read voltage is from 3.0 to 3.6 V. The cutting word-line voltages for margin-read-1 test and margin-read-0 test are 2.85 and 3.65 V, respectively. As Group 3 in Fig. 6 shows, all the NROM bits initially failing to represent a quality value 1 have a threshold voltage ranging from 4.8 to 5.4 V after programmed to value 0, which can all pass the margin-read-0 test.

Table I lists the mean and variance of the threshold voltages measured from the NROM bits passing the margin-read-1 test (Column 2) and failing the margin-read-1 test (Column 3) after they are all programmed to value 0. As the result shows, the value 0 represented by the NROM bits failing the margin-read-1 test actually have a higher mean of their threshold voltages than those passing the margin-read-1 test. Also, the variation of their threshold voltages is smaller. This result shows that the NROM

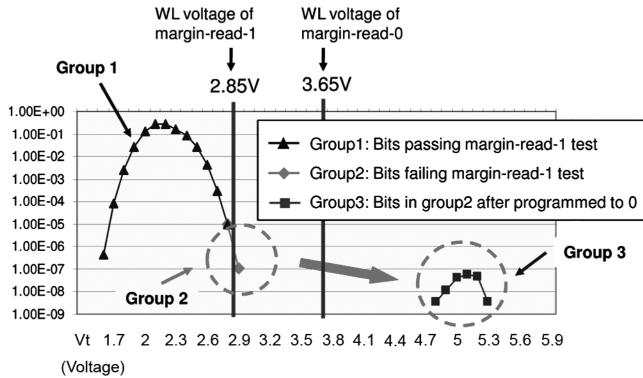


Fig. 6. V_t distribution before and after programmed a physical-level defective bit to value 0.

TABLE I
V_T COMPARISON AFTER THE NROM BITS ARE PROGRAMMED TO VALUE 0

	NROM bits passing margin-read-1 test	NROM bits failing margin-read-1 test
V_t mean	4.98	5.25
V_t variance	0.0297	0.0203

bits failing the margin-read-1 test can even represent a better value 0 than those passing the margin-read-1 test. Thus, a physical-level defective bit identified in the wafer test can indeed be used to represent a quality value 0, and hence may not be a data-level defective bit after the customer's code is programmed.

C. Statistics for Single Defects

When we replace a defective bit, the whole word containing that bit will be replaced by a spare row or column. Given n physical-level defective bits in a word, the probability that this word needs to be replaced for the customer's code is $(1-1/2^n)$, since there is a 50% chance that this bit in the customer's code just happens to be a value 0. If more physical-level defective bits exist in a word, it is less likely that all these defective bits in the customer's code happen to be a value 0. As a result, this physical-level defective word may still need to be replaced. In addition, it is extremely difficult for a physical-level column defect or row defect to be reused in the customer's code since matching all defective bits of a column or row defect with customer's code is very unlikely. Therefore, we hope that most physical-level defects in our NROM process could be random single defects, not clustered, row, or column defects, such that less data-level defective words would exist for the repair analysis.

Fig. 7 shows the probability distribution for the number of the physical-level defective bits in a physical-level defective word. This probability distribution is collected from the same sources as used in the experiments presented in Section III-B. As the result shows, more than half of physical-level defective words contain exact one defective bit. Majority of the defective words contain less or equal to 4 defective bits, and thus can be potentially reused in the programmed code. In average, this reuse probability for a defective word is more than 30% based on the distribution in Fig. 7, meaning that the number of defective

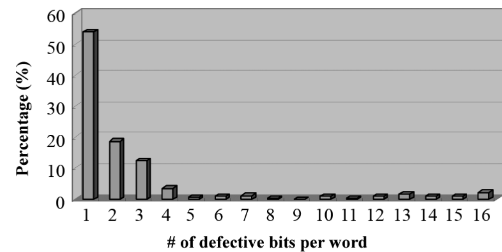


Fig. 7. Number of defective bits per word.

TABLE II
THE NUMBER OF WORDS FAILING THE MARGIN-READ-1 TEST ON A DEFECTIVE ROW

# of defective words in a defective row	1	2	3	4 to 4096
probability distribution	96.04%	1.46%	0.42%	2.08%

words to be repaired in the proposed test flow can be in average 30% less than that in the conventional flow.

Next, we further count the number of physical-level defective words in a row (for only the rows with at least one defective word). Its probability distribution is listed in Table II, showing that more than 96% of the defective rows contains exact one defective word. Therefore, 96% defective rows can be reused in the customer's code as long as their single defective word can be reused. A similar statistic can be obtained for the defective columns.

The results shown in Fig. 7 and Table II demonstrate that majority of the defects are random single defects. Thus, there exist a significant chance that a row (or column) required to be replaced in the conventional wafer test may not necessarily need to be replaced in our proposed test flow.

IV. MATHEMATICAL ESTIMATION TO SUCCESSFUL-REPAIR RATE

The most challenging task in the proposed test flow is to mathematically estimate the successful-repaired probability resulting from the repair analysis in the package test based on only the defect information collected in the wafer test. The accuracy and efficiency of this estimation scheme directly affect the cost of the proposed test flow. Also, its computation and space complexities determine the requirement of the ATE's computational capability. In the mathematical derivation shown in Section IV-A–IV-C, we first focus on the estimation of the successful-repaired probability targeting only the single defects. In Section IV-F, the estimation incorporating column and row defects can be obtained by slightly modifying the estimation considering only the single defects.

A. Derivation Overview for Successful-Repair Probability

After the margin-read-1 test in the wafer test, we can obtain the following three numbers about the physical-level defects from a tested part. These three numbers are the input parameters for estimating the probability of the part being successfully repaired in the package test.

- row_d: the number of *row defects*, which are the defect only repairable by a spare row.
- col_d: the number of *column defects*, which are the defect only repairable by a spare column.
- k: the number of physical-level defective words not locating on the row_d defective rows and col_d defective columns.

The estimation scheme also requires the following parameters, which can be computed before the wafer test is applied:

R total number of rows;

C total number of columns;

M total number of spare rows;

N total number of spare columns;

μ the average probability that a physical-level defective word is also a data-level defective word.

The following three random variable are defined for deriving the successful-repaired probability:

DD number of data-level defective words;

PD number of physical-level defective words;

DB number of physical-level defective bits in a defective word.

Since we first focus on the analysis considering only single random defective words, the successful-repaired probability based on k physical-level defective words in a tested part, denoted as $PSR(k)$, can be expressed by (1).

$$PSR(k) = \sum_{x=0}^{k} P\{DD = x | PD = k\} \times DSR(x), \quad (1)$$

where $P\{DD = x|PD = k\}$ is the probability that the number of data-level defective words existing after the code is programmed is equal to x given k data-level defective words in the part, and $DSR(x)$ is the probability that the part can be successfully repaired given x data-level defective words. Because the customer's code is not yet known in the wafer test, the number of the resulting data-level defective words may range from 0 to k. Thus, we need to sum the term $DSR(x) \times P\{DD = x|PD = k\}$ from $x = 0$ to $x = k$.

The following two subsections introduce how $P\{DD = x|PD = k\}$ and $DSR(x)$ can be calculated, respectively.

B. Computation of $P\{DD = x|PD = k\}$

In our margin-read-1 test, we read out one word at each read operation and check whether the data of that word equals to the expected initial value (value 1 for all bits). It implies that we can only know which word address fails the margin-read-1 test. The number of defective bits in a word cannot be obtained by the ATE operations of the current margin-read-1 test. However, the probability of a physical-level defective word remaining a data-level defective word depends on the number of defective bits in a word. One solution to estimate this probability is to design a new margin-read-1 test which is able to obtain the number of defective bits in a word, but this new margin-read-1 test requires significantly longer test application time to count the defective bits and may not be cost-effective. Thus, we decide to use the statistics of the defective bits per word collected in our past products to predict the $P\{DD = x|PD = k\}$.

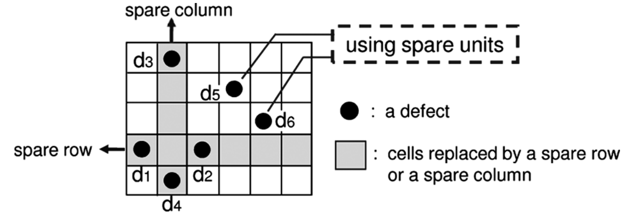


Fig. 8. An example of using spare units.

Even though the probability that a physical-level defective word remains a data-level defective word is different for each defective word, which violates the premise of using a binomial distribution, we found out that the following binomial distribution can be used to approximate the distribution of $P\{DD = x|PD = k\}$.

$$P\{DD = x|PD = k\} = C_x^k \times (1 - \mu)^{k-x} \times \mu^x, \quad (2)$$

where μ is the average probability that a physical-level defective word remains a data-level defective word and can be calculated based on the probability distribution of DB (the number of the defective bits per word) as follows.

$$\mu = \sum_{y=0}^{16} P\{DB = y\} \times (1 - 0.5^y), \quad (3)$$

where $1 - 0.5^y$ is the probability that a physical-level defective word remains a data-level defective word given y defective bits in a word. The distribution of $P\{DB = y\}$ is shown in Fig. 7.

C. Computation of $DSR(x)$

$DSR(x)$ is the probability that x random defective words can be successfully repaired by M spare rows and N spare columns in an $R \times C$ memory array. The past research works relied on the random simulations to estimate the $DSR(x)$ [15], [16], which is too time-consuming to be used during the wafer test. In our estimation scheme, we use a four-dimension probability array, $S^{(i)}[m][n][z]$, to estimate the $DSR(x)$. The array element $S^{(i)}[m][n][z]$ represents the probability that i defects can be repaired by using m spare rows, n spare columns, and z *spare units*. A spare unit here represents that a defect can be repaired by either a spare row or a spare column but which one of them is not determined yet. For example, the six defects shown in Fig. 8 are repaired by one spare row, one spare column, and two spare units ($i = 6$, $m = 1$, $n = 1$, and $z = 2$), where each of d_5 and d_6 can be repaired by a spare row or spare column.

Our estimation scheme uses an induction-based approach to calculate the elements of S when the number of defects is $i + 1$ ($S^{(i+1)}[m][n][z]$) based on the elements of S when the number of defects is i ($S^{(i)}[m][n][z]$). The computation of $S^{(i)}[m][n][z]$ starts from $i = 1$ and each iteration increases i by 1 until $i = x$. The ranges of m, n, and z are $0 \leq m \leq M$,

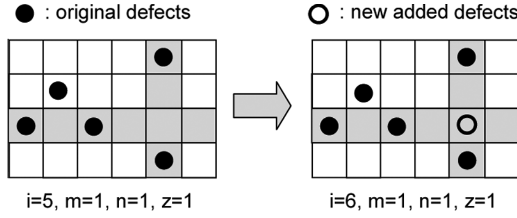


Fig. 9. An instance of the probability event associated with p_1.

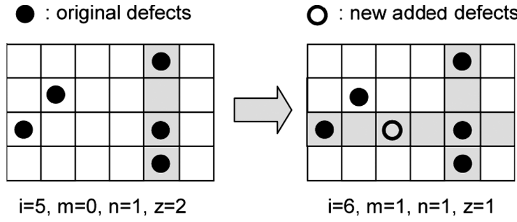


Fig. 10. An instance of the probability event associated with p_2.

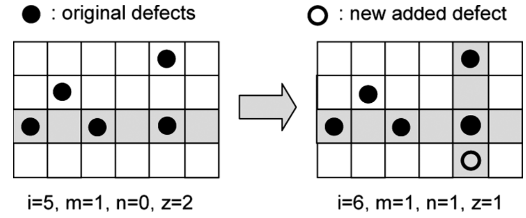


Fig. 11. An instance of the probability event associated with p_3.

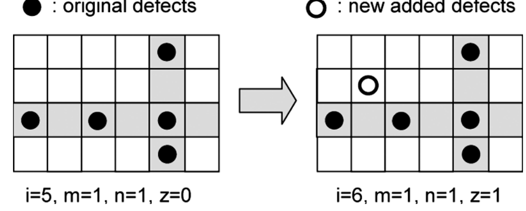


Fig. 12. An instance of the probability event associated with p_4.

$0 \le n \le N$, and $0 \le z \le M+N$, respectively. The induction equation is shown as follows:

$$\begin{aligned} &S^{(i+1)}[m][n][z] \\ &\quad = S^{(i)}[m][n][z] \times p_1(i,m,n,z) \\ &\qquad + S^{(i)}[m-1][n][z+1] \times p_2(i,m-1,n,z+1) \\ &\qquad + S^{(i)}[m][n-1][z+1] \times p_3(i,m,n-1,z+1) \\ &\qquad + S^{(i)}[m][n][z-1] \times p_4(i,m,n,z-1). \end{aligned} \tag{4}$$

The probability $S^{(i+1)}[m][n][z]$ is contributed from four probability events. The probabilities of these four events are denoted by p_1, p_2, p_3, and p_4, respectively. Each of p_1, p_2, p_3, and p_4 is a function of i, m, n, and z. The first event (associated with $p1$) occurs when the extra $(i+1)\mathrm{th}$ defect falls in a location covered by the m spare rows or the n spare columns repairing the original i defects. Hence no extra spare row, column, or unit needs to be used to repair this $(i+1)$ defect. Fig. 9 shows an instance of this probability event.

The second event (associated with $p2$) occurs when the extra $(i+1)\mathrm{th}$ defect falls in a location where we need to fix the usage of a spare unit as a spare row to repair this defect. Hence, the number of used spare rows m is increased by 1 and the number of spare units z is decreased by 1. Fig. 10 shows an instance of this probability event.

Third, the extra $(i+1)\mathrm{th}$ defect may fall in a location where we need to fix the usage of a spare unit as a spare column to repair this defect. Hence, the number of used spare column n is increased by 1 and the number of spare units z is decreased by 1. Fig. 11 shows an instance of this probability event.

Last, the extra $(i+1)\mathrm{th}$ defect may fall in a location where we can use another spare unit to repair this defect. Hence, the number of spare units z is increased by 1. Fig. 12 shows an instance of this probability event.

Equation (5) shows the initial condition of $S^{(i)}[m][n][z]$ when $i = 1$, where one single defect is always repaired by using a spare unit, which is the cheapest way of repairing a single defect. Thus, $S^{(1)}[0][0][1]$ is set to 1 and all other elements for $i = 1$ are set to 0.

$$\begin{aligned} &S^{(1)}[0][0][1] = 1 \\ &S^{(1)}[m][n][z] = 0, \qquad \text{otherwise}. \end{aligned} \tag{5}$$

Then, $DSR(x)$ can be obtained by summing the probability of each $S^{(x)}[m][n][z]$, whose m and n are not larger than M and N, respectively, and whose z is not larger than $M+N-m-n$. The calculation of $DSR(x)$ is listed in (6).

$$DSR(x) = \sum_{m=0}^{M} \sum_{n=0}^{N} \sum_{z=0}^{M+N-m-n} S^{(x)}[m][n][z]. \tag{6}$$

D. Computation of p_1, p_2, p_3, and p_4

In this subsection, we will show the detailed computation of p_1, p_2, p_3, and p_4, which are all functions of m, n, z, and i. In the following illustrations of these four probabilities, we consider the case of adding one extra defect into the situation where the original i defects are already repaired by m spare row, n spare columns, and z spare unit. In other words, we are computing the probability p_1 (i,m,n,z), p_2 (i,m,n,z), p_3 (i,m,n,z), and p_4 (i,m,n,z).

An event of p_1 (i,m,n,z) occurs when the extra $(i+1)\mathrm{th}$ defect falls in the area covered by the original m spare row and n spare columns, which is illustrated in Fig. 13. Equation (7) lists the computation of p_1 (i,m,n,z), where the denominator $(R \times C - i)$ represents the total number of cell locations that the extra defect may fall. For the numerator of p_1, the term $(R \times n + C \times m - m \times n)$ represents the locations repaired by the m spare rows and n spare columns. Then we need to exclude the number of defects falling already inside the repaired area, i.e., the term $(i - z)$. Note that, after adding the extra defect for the event of p_1 (i,m,n,z), the $i+1$ defects can still be repaired by m spare rows, n spare columns, and z spare units.

$$p_1 = \frac{R \times n + C \times m - m \times n - (i - z)}{R \times C - i}. \tag{7}$$

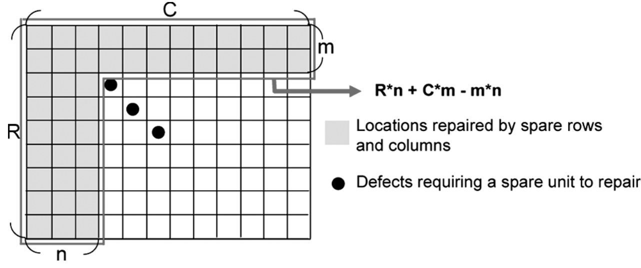


Fig. 13. Illustration for the computation of $p_1\ (i,m,n,z)$.

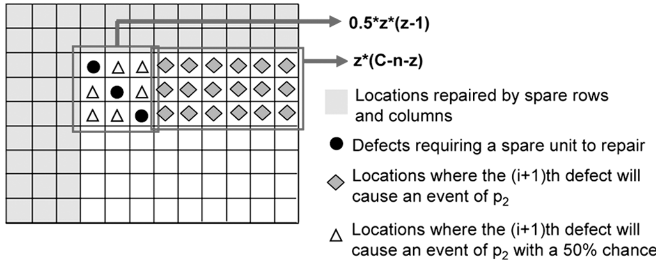


Fig. 14. Illustration for the computation of $p_2\ (i,m,n,z)$.

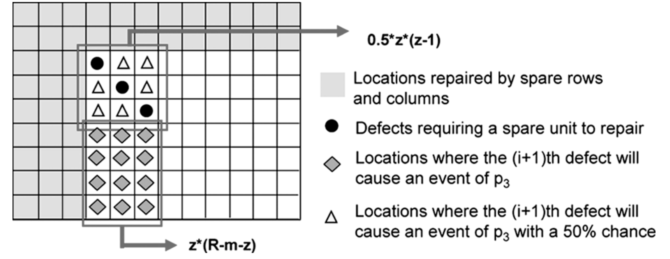


Fig. 15. Illustration for the computation of $p_3\ (i,m,n,z)$.

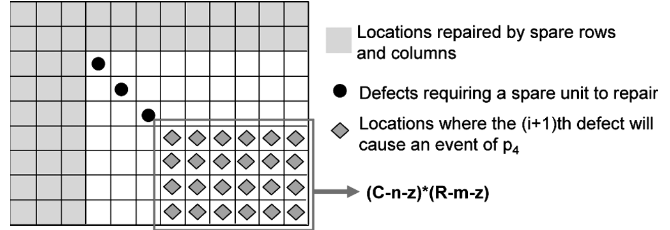


Fig. 16. Illustration for the computation of $p_4\ (i,m,n,z)$.

An event of $p_2\ (i,m,n,z)$ occurs when the extra $(i+1)\mathrm{th}$ defect falls in the area where we need to turn a spare unit into a spare row to repair the extra defect. Fig. 14 illustrates such area. Equation (8) lists the computation of $p_2\ (i,m,n,z)$. For the numerator of p_2, the term $(C-n-z)\times z$ represents the locations where the extra defect will for sure cause an event of p_2 (labeled as a diamond in Fig. 14). The term $z\times(z-1)$ represents the locations where the extra defect will cause an event of p_2 with a 50% chance (labeled as a triangle in Fig. 14) since we can turn a spare unit into either a spare row or spare column to repair the extra defect. Note that, after adding the extra defect for the event of $p_2\ (i,m,n,z)$, the $i+1$ defects are repaired by $m+1$ spare rows, n spare columns, and $z-1$ spare units.

$$p_2 = \frac{(C-n-z)\times z + 0.5\times z\times(z-1)}{R\times C-i}. \tag{8}$$

An event of $p_3\ (i,m,n,z)$ occurs when the extra $(i+1)\mathrm{th}$ defect falls in the area where we need to turn a spare unit into a spare column to repair the extra defect. Fig. 15 illustrates such area. Equation (9) lists the computation of $p_3\ (i,m,n,z)$. In a similar manner as p_2, the term $(R-m-z)\times z$ and the term $z\times(z-1)$ in (9) represent the locations where the extra defect will cause an event of p_3 for sure (labeled as a diamond in Fig. 15) and with a 50% chance (labeled as a triangle in Fig. 15), respectively. Note that, after adding the extra defect for the event of $p_3\ (i,m,n,z)$, the $i+1$ defects are repaired by m spare rows, $n+1$ spare columns, and $z-1$ spare units.

$$p_3 = \frac{(R-m-z)\times z + 0.5\times z\times(z-1)}{R\times C-i}. \tag{9}$$

An event of $p_4\ (i,m,n,z)$ occurs when the extra $(i+1)\mathrm{th}$ defect falls in the cell locations where we need an extra spare unit to repair the defect. Those cells are labeled as a diamond in Fig. 16. Equation (10) lists the computation of $p_4\ (i,m,n,z)$, in which the term $(R-m-z)\times(C-n-z)$ represents the number of those diamond cells in Fig. 16. Note that, after adding the extra defect for the event of $p_4\ (i,m,n,z)$, the $i+1$ defects are repaired by m spare rows, n spare columns, and $z+1$ spare units.

$$p_4 = \frac{(R-m-z)\times(C-n-z)}{R\times C-i}. \tag{10}$$

E. Optimality of Computed $DSR(x)$

The above estimation is actually a lower bound of the real $DSR(x)$ since we consider the successful-repaired combinations of using spare rows, spare columns, and spare units by adding one more defect to the defect map at a time. In reality, the repair analysis is applied after all x defects are added to the defect map. However, by using the concept of "spare units", our estimation scheme can cover a wide range of the successful-repaired combinations. Also, the applied repair algorithm in the repair analysis is only a heuristic and cannot cover all the combinations of using spare rows and spare columns as well. Thus, our estimation scheme can provide a close approximation to $DSR(x)$, which will be shown in our experiments in Section V-A.

F. Complexity of Computing $DSR(x)$

The complexity of computing $DSR(x)$ are determined by the size of the 4-dimensional array $S^{(x)}[m][n][z]$, which is $(M*N*(M+N)*x)$. To compute each array element of $S^{(x)}[m][n][z]$ (including the computation of p_1 to p_4 and (6)) requires 14 times of multiplication, 29 times of addition, and 4 times of division. Thus, the computation complexity and space complexity of computing $DSR(x)$ are both $O(M*N*(M+N)*x)$. Note that this complexity is proportional to the number of spare rows and spare columns instead of the size of the memory array $R\times C$, and hence can be properly controlled. Also, the value of $S^{(x)}[m][n][z]$ drops quickly when x exceeds a certain point. We can stop the computation of $S^{(x)}[m][n][z]$ when its value becomes negligible. The real computation effort will be not as high as shown in above. In addition, the array of $S^{(x)}[m][n][z]$

TABLE III
$DSR(x)$ ACCURACY GIVEN DIFFERENT NUMBERS OF SINGLE DEFECTS

# of single defects (x)	$DSR(x)$ in %		
	estimation (a)	simulation (b)	$\|a-b\|$
21	98.88	98.88	0
22	94.15	93.79	0.36
23	83.79	83.15	0.64
24	68.34	67.91	0.43
25	50.67	50.49	0.18
26	34.15	34.18	0.03
27	21.00	20.82	0.18
28	11.86	11.78	0.08
29	6.20	6.16	0.04
30	3.01	3.08	0.07
Average	–	–	0.20

TABLE IV
$DSR(x)$ ACCURACY GIVEN DIFFERENT NUMBERS OF SPARE COLUMNS

# of spare columns (N)	$DSR(x)$ in %		
	estimation (a)	simulation (b)	$\|a-b\|$
7	2.84	2.96	0.12
8	10.18	10.46	0.28
9	26.34	26.40	0.06
10	50.67	50.49	0.18
11	75.31	74.99	0.32
12	91.63	91.22	0.41
13	98.32	98.16	0.16
14	99.85	99.85	0
Average	–	–	0.19

TABLE V
$DSR(x)$ ACCURACY GIVEN DIFFERENT NUMBERS OF COLUMNS

# of columns (C)	$DSR(x)$ in %		
	estimation (a)	simulation (b)	$\|a-b\|$
50	86.87	86.18	0.69
100	50.67	50.49	0.18
150	35.32	34.66	0.06
200	28.13	28.24	0.11
400	18.74	18.72	0.02
Average	–	–	0.21

can be reused for estimating the successful-repaired probability of the next tested part. Therefore, the computation complexity of $DSR(x)$ is tractable for testing large industrial NROM-based ROMs.

G. Considering Row and Column Defects

As discussed in Section III-C, a physical-level row defect (or column defect) is hard to be reused since it is unlikely that the customer's code on all the defective bits in that row (or column) represents a value 0. Thus, once a physical-level row defect (or column defect) is identified, a spare row (or spare column) must be spent for that defect. In other words, the effective number of spare rows (or spare columns) used for repairing the random single defective words is $M - row_d\ (N - col_)d$, instead of the $M\ (N)$ as shown in the previous subsection. Therefore, when computing $PSR(x)$ (and $DSR(x)$), we only need to substitute the parameter M by $M - row_d$ and N by $N - col_d$ to consider the impact of row defects and column defects. The other computations remain the same. It also means that the function PSR should actually be denoted as $PSR(x, row_d, col_d)$, i.e., a function of the number of random single defects (x), the number of row defects (row_d), and the number of column defects (col_d). Note that in our estimation scheme, a row (or column) containing more than 4 physical-level defective words is defined as a row (or column) defect.

V. EXPERIMENTS AND DISCUSSION

A. Accuracy of $DSR(x)$

The most difficult and challenging task in our estimation scheme is the analysis of $DSR(x)$. We first validate the accuracy of $DSR(x)$ by comparing its results with the results obtained from a random simulation using 1-million samples. In each sample, we randomly select x defects on an $R \times C$ memory array and then apply the repair-most algorithm to check whether these x defects can be repaired by M spare rows and N spare columns.

Table III reports the results of $DSR(x)$ for different numbers of single defects x when $R = C = 100$ and $M = N = 10$. Column 2 and Column 3 of Table III list the value of $DSR(x)$ obtained by our estimation scheme and the random simulation, respectively. Column 4 lists the difference between the estimation scheme and the simulation. As the result shows, our estimation scheme can match the simulation result closely. The average difference is 0.20% and the largest difference is 0.64%.

Similar experiments are conducted for Tables IV and V to report the results of $DSR(x)$ for different numbers of spare columns (N) and different total numbers of columns (C), respectively. The other parameters are fixed as shown in the tables. The same trend can be observed as in Table III. The difference of the reported $DSR(x)$ between our estimation scheme and the random simulation is limited, only 0.19% for Table IV and 0.21% for Table V in average. All above results demonstrate the accuracy of $DSR(x)$ computed by our estimation scheme over different combinations of parameters.

B. Accuracy of $PSR(k)$

In this subsection, we compare the $PSR(k)$ computed by our estimation scheme with that obtained from random simulation based on an 8192x4096 memory array with 16 spare rows and 6 spare columns, which is the configuration used by a 512 Mb NROM-based ROM on a current production line. Each word contains 16 bits. Each spare row and spare column can repair 8 consecutive row addresses and 4 consecutive column addresses, respectively. The specifications of this NROM-based ROM are listed in Table VI.

In the random simulation, we sample the physical-level single defects, row defects, and column defects for a tested part based on the Poisson distributions with λ_{SD}, λ_{RD}, and λ_{CD}, respectively. The number of sampled defective bits in a physical-level defective word follows the probability distribution shown in Fig. 7. For each sampled part, we compute its corresponding $PSR(k)$ and package it if the computed $PSR(k)$ is larger than the given threshold, $p_th = 0.5$, in this experiment. Then we program a random customer's code to each packaged part, identify its data-level defects, and apply the repair algorithm to determine whether this part can pass the package test. We average

TABLE VI
CONFIGURATION OF NROMs IN THE EXPERIMENTS

Density	$32M \times 16 = 512M$ bits
Access time	100ns (10MH)
Operating current	50mA
Supply voltage	3.0-3.6V
Temperature	0-70 C
Input Hi/Lo	2.4V/0.4V
Output Hi/Lo	2.4V/0.4V
Package dimension	28.5 mm x 16.0 mm

TABLE VII
$PSR(k)$ ACCURACY GIVEN DIFFERENT λ_{CD} OF SINGLE-DEFECT DISTRIBUTION

parameters: R=8192, C=4096, M=16, N=6, λ_{RD}=6, λ_{CD}=2			
single-defect distribution (λ_{SD})	$PSR(k)$ in %		
	estimation (a)	simulation (b)	$\|a-b\|$
10	97.27	97.09	0.18
12	94.31	94.04	0.27
14	89.52	89.14	0.38
16	82.59	81.08	0.51
18	73.41	72.78	0.63
20	63.12	62.25	0.87
22	52.10	51.33	0.77
Average	–	–	0.51

TABLE VIII
$PSR(k)$ ACCURACY GIVEN DIFFERENT λ_{CD} OF COLUMN-DEFECT DISTRIBUTION

parameters: R=8192, C=4096, M=16, N=6, λ_{SD}=12, λ_{RD}=6			
column-defect distribution (λ_{CD})	$PSR(k)$ in %		
	estimation (a)	simulation (b)	$\|a-b\|$
1	96.85	96.75	0.10
2	94.31	94.04	0.27
3	89.19	88.93	0.26
4	80.53	80.25	0.28
5	68.29	68.32	0.03
6	54.30	54.37	0.07
Average	–	–	0.17

the $PSR(k)$ estimated for each packaged part and compare this average $PSR(k)$ with the percentage of the packaged parts actually passing the package test. The number of sampled parts is 100 k in the random simulation.

Table VII gives different numbers of λ_{SD} for the single defect distribution and reports the comparison of the average $PSR(k)$ between our estimation scheme and the simulation. The other parameters are fixed as listed in Table VII. As the result shows, the average $PSR(k)$ computed by our estimation scheme can closely match the simulation result for different given distributions of single defects. The average and maximum differences between the estimation results and the simulation results are 0.51% and 0.87%, respectively.

Similar to Tables VII–X compare the average $PSR(k)$ for different distributions of column defects, different numbers of spare columns, and different array sizes, respectively. Their average differences are 0.17%, 0.24%, and 0.37%, respectively. The results shown on Tables VII–X demonstrate that our proposed test flow can effectively predict the probability of a part being successfully repaired in the package test based on only the defect information collected from the wafer test.

TABLE IX
$PSR(k)$ ACCURACY GIVEN DIFFERENT SPARE COLUMNS

parameters: R=8192, C=4096, M=16, λ_{SD}=12, λ_{RD}=6, λ_{CD}=2			
# of spare column (N)	$PSR(k)$ in %		
	estimation (a)	simulation (b)	$\|a-b\|$
2	57.23	56.71	0.48
3	74.11	74.48	0.37
4	84.49	84.32	0.17
5	90.55	90.37	0.18
6	94.21	93.95	0.26
7	96.48	96.37	0.11
8	97.91	97.80	0.11
Average	–	–	0.24

TABLE X
$PSR(k)$ ACCURACY GIVEN DIFFERENT ARRAY SIZES

parameters: M=16, N=6, λ_{SD}=12, λ_{RD}=6, λ_{CD}=2			
array size (RxC)	$PSR(k)$ in %		
	estimation (a)	simulation (b)	$\|a-b\|$
8192×4096	94.21	93.95	0.26
8192×2048	94.51	94.22	0.29
8192×1024	94.89	94.31	0.58
4096×4096	94.47	94.18	0.29
2048×2048	97.89	97.53	0.36
1024×1024	98.07	97.64	0.43
Average	–	–	0.37

TABLE XI
$PSR(k)$ SIMULATION USING RANDOM CODES (A) VERSUS $PSR(k)$ SIMULATION USING A SINGLE CODE

parameters: R=8192, C=4096, M=16, N=6, λ_{RD}=6, λ_{CD}=2			
single-defect distribution (λ_{SD})	$PSR(k)$ in %		
	random-code simulation (a)	single-code simulation (b)	$\|a-b\|$
10	97.09	97.19	0.10
12	94.04	94.32	0.28
14	89.14	88.89	0.25
16	82.08	82.15	0.07
18	72.78	73.01	0.23
20	62.25	62.34	0.09
22	51.33	51.04	0.29
Average	–	–	0.19

From Tables VII–X, the $PSR(k)$ simulation sampled a different customer code for each time. However, when we receive a customer code, we will program the same customer code for multiple times. Table XI compares the $PSR(k)$ obtained by different random codes with that obtained by a single random code. The parameter setting used in Table XI is the same as Table VII. As the result shows, the $PSR(k)$ obtained by different random codes is also close to that obtained by a single random code. The average difference is 0.19%.

C. Proposed Flow vs. Conventional Flow

In Tables XII–XV, we perform the conventional test flow to those parts sampled in Tables VII–X, respectively. Then we compare the resulting yield of the conventional test flow with that of the proposed test flow. Column 2 of these four tables

TABLE XII
COMPARISON BETWEEN THE PROPOSED AND CONVENTIONAL TEST FLOWS GIVEN DIFFERENT SINGLE-DEFECT DISTRIBUTIONS

parameters: R=8192, C=4096, M=16, N=6, λ_{RD}=6, λ_{CD}=2				
single-defect distribution (λ_{SD})	proposed flow		conventional flow	yield difference $\|a-b\|$
	% of packaged parts	yield in % (a)	yield in % (b)	
10	97.97	96.78	86.42	10.36
12	95.37	93.95	72.85	21.10
14	91.08	87.08	57.09	29.99
16	84.70	78.91	40.44	38.47
18	75.19	68.01	26.55	41.46
20	64.24	56.12	16.03	40.09
22	52.36	44.27	8.83	35.44

TABLE XIII
COMPARISON BETWEEN THE PROPOSED AND CONVENTIONAL TEST FLOWS GIVEN DIFFERENT λ_{CD} OF COLUMN-DEFECT DISTRIBUTION

parameters: R=8192, C=4096, M=16, N=6, λ_{SD}=12, λ_{RD}=6				
column-defect distribution (λ_{CD})	proposed flow		conventional flow	yield difference $\|a-b\|$
	% of packaged parts	yield in % (a)	yield in % (b)	
1	97.77	96.20	80.23	15.97
2	95.37	93.95	72.85	21.10
3	90.56	87.27	64.56	22.71
4	81.70	77.93	54.50	23.43
5	69.27	65.56	43.33	22.23
6	54.88	51.40	32.83	18.57

TABLE XIV
COMPARISON BETWEEN THE PROPOSED AND CONVENTIONAL TEST FLOWS GIVEN DIFFERENT NUMBER OF SPARE COLUMNS

parameters: R=8192, C=4096, M=16, λ_{SD}=12, λ_{RD}=6, λ_{CD}=2				
# of spare columns (N)	proposed flow		conventional flow	yield difference $\|a-b\|$
	% of packaged parts	yield in % (a)	yield in % (b)	
2	58.09	54.09	31.71	22.38
3	75.47	71.11	44.34	26.77
4	85.97	81.88	56.21	25.67
5	91.92	88.72	65.02	23.70
6	95.37	93.95	72.85	21.10
7	97.40	95.71	80.65	15.06
8	98.59	97.46	85.37	12.09

lists the percentage of parts passing the wafer test in our proposed test flow. Column 3 lists the yield of the proposed test flow, which is the total percentage of parts passing both the proposed wafer test and package test. Column 4 lists the yield of the conventional test flow, which is actually the percentage of parts passing the conventional wafer test. Column 5 lists the yield difference between the two test flows. As the result shows, the proposed test flow can always generate a higher yield than the conventional test flow. It shows the advantage of postponing the repair analysis after the customer's code is programmed, such that the bits failing to represent a value 1 can be used to represent a value 0.

Note that in our simulation we omit the probability that a part passing the conventional wafer test may fail the conventional package test. Those failed parts may result from the defects generated in packaging or the timing-related defects captured during at-speed-testing. In the current product lines, this probability is lower than 5%. Most yield loss in the conventional test flow occurs in the wafter test. If adding this probability into consideration, the yield of both the conventional and the proposed test flows would be slightly lowered.

TABLE XV
COMPARISON BETWEEN THE PROPOSED AND CONVENTIONAL TEST FLOWS GIVEN DIFFERENT MEMORY SIZES

parameters: M=16, N=6, λ_{SD}=12, λ_{RD}=6, λ_{CD}=2				
array size (RxC)	proposed flow		conventional flow	yield difference $\|a-b\|$
	% of packaged parts	yield in % (a)	yield in % (b)	
8192 × 4096	95.37	93.95	72.85	21.10
8192 × 2048	96.30	94.28	73.51	20.77
8192 × 1024	97.13	94.80	75.99	18.81
4096 × 4096	96.81	94.69	74.29	20.40
2048 × 2048	98.69	97.53	87.59	9.94
1024 × 1024	98.75	97.64	89.41	8.23

As the result shows in the above tables, the yield of the proposed test flow varies with a lot parameters, such as the defect distributions, spare resources, and the array sizes. In order to design the most cost-effective repair architecture, designers need to evaluate the yield of each memory configuration and its corresponding overhead in advance. With the proposed estimation scheme, we can efficiently and effectively estimate the yield of different memory configurations, which allows us to avoid the time-consuming random simulations. Sampling 100 k parts in the above experiment takes more than one day to obtain the yield for one configuration. It shows another advantage of the proposed estimation scheme during the architecture design of NROM-based ROMs.

Since we assume that all packaged parts in the conventional test flow can pass its package test, the reported yields of the conventional test flow are actually equal to the percentages of parts being packaged. As the result shows, the proposed test flow needs to package more parts than the conventional test flow. While more extra packaged parts may more likely increase the overall yield for the proposed test flow, more packaged parts may be discarded as well. Thus, we need to set a proper value of the threshold probability, p_th, to optimize the overall profit of the proposed test flow.

D. Finding a Proper p_th

Table XVI lists the percentage of the packaged parts, the overall yield, and the percentage of the packaged parts being discarded for different values of p_th. When p_th decreases, the yield increases but the number of discarded packaged parts also increases. By substituting Table XVI's numbers into the cost/profit function of a given product line, we can find out the setting of p_th achieving maximal profit.

Equation (11) first lists a simplified computation of the profit of fabricating a die. In Equation (11), Y_w denotes the yield of parts passing the wafer test. $Y_{w\&p}$ denotes the yield of parts passing both the wafer test and the package test. C_{fab} and C_{pack} denote the cost of fabricating a die and packaging a die, respectively, and sp denotes the sale price of an NROM-based ROM. $Y_{w\&p} \times sp$ represents expectation of the income, i.e., the sale price of an NROM-based ROM times the probability that a fabricated die can be sold. C_{fab} and $Y_w \times C_{pack}$ represent the expectation of the cost of fabricating a die and packaging a die, respectively, where Y_w means the probability that a fabricated die will be packaged.

$$\text{Profit} = Y_{w\&p} \times sp - C_{fab} - Y_w \times C_{pack}. \tag{11}$$

TABLE XVI
COMPARISON BETWEEN USING DIFFERENT VALUES OF p_th

parameters: R=8192, C=4096, M=16, N=6, λ_{SD}=20, λ_{RD}=6, λ_{CD}=2			
p_th	% of packaged parts	overall yield in %	% of discarded packaged parts
0	100	67.93	32.07
0.1	88.15	67.35	20.80
0.2	82.95	66.68	16.27
0.3	78.59	65.49	13.10
0.4	73.90	63.95	9.95
0.5	69.74	61.93	7.81
0.6	65.48	59.63	5.85
0.7	59.46	55.73	3.73
0.8	53.66	51.41	2.25
0.9	45.89	44.88	1.01
1.0	13.33	13.33	0

TABLE XVII
PROFIT PER DIE COMPUTED BY MATHEMATICAL MODEL AND SIMULATION FOR DIFFERENT VALUES OF p_th

parameters: R=8192, C=4096, M=16, N=6, λ_{SD}=16, λ_{RD}=6, λ_{CD}=2			
p_th	Estimation(a)	Simulation(b)	$\lvert a - b\rvert$
0.1	2.412	2.371	0.041
0.15	2.451	2.411	0.040
0.2	2.480	2.441	0.039
0.25	2.503	2.466	0.037
0.3	2.517	2.482	0.035
0.35	2.526	2.493	0.033
0.4*	2.530	2.498	0.032
0.45	2.527	2.491	0.036
0.5	2.514	2.488	0.026
0.55	2.503	2.479	0.024
0.6	2.476	2.456	0.020
0.65	2.428	2.411	0.017
0.7	2.361	2.352	0.009
0.75	2.292	2.287	0.005
0.8	2.204	2.202	0.002
0.85	2.105	2.108	0.003
0.9	1.933	1.919	0.014
avg.			0.0243

In the above profit function, the value of Y_w and $Y_{w\&p}$ is actually determined by the specified value of p_th. Assume that the Poisson distribution of the number of single defects (denoted as SD), row defects (denoted as RD), and column defects (denoted as CD) can be extracted from the current manufacturing technology. Then the value of Y_w can be obtained by using (12), which sums the joint probabilities of SD, RD, and CD that result in a $PSR(i,j,k)$ larger than p_th, i.e., our criteria passing the wafer test. In our computation, we assume that the random variables SD, RD, and CD are all independent, and hence the joint probability of SD, RD, and CD can be quickly obtained by multiplying their probability mass functions. In addition, the increase of the index i (the number of single defects) in the summation stops when the resulting point probability is negligible.

$$Y_w = \sum_{i=0} \sum_{j=0}^{M} \sum_{k=0}^{N} P\{SD = i, RD = j, CD = k | PSR(i,j,k) > p_{th}\}. \quad (12)$$

Next, since $PSR(i,j,k)$ represents the probability that a part can pass the package test, the value of $Y_{w\&p}$ can be obtained by using (13), which multiplies the joint probability of SD, RD, and CD shown in (12) by its $PSR(i,j,k)$ to calculate the expectation of the part passing the package test.

$$Y_{w\&p} = \sum_{i=0} \sum_{j=0}^{M} \sum_{k=0}^{N} P\{SD = i, RD = j, CD = k | PSR(i,j,k) > p_th\}. \quad (13)$$

Table XVII compares the profit functions obtained by our mathematical model (12) and (13) with the results obtained by a 500 K-sample random simulation. In our profit function, we set $sp = 10$, $C_{fab} = 2$, and $C_{pack} = 4$. The lambda value of each defect distribution is also listed at the top of Table XVII. Note that this experimental setting is randomly chosen for demonstrating the process of finding a proper p_th and does not relate to any product line. As the result shows, the peak of the profit function (2.498) occurs when $p_th = 0.4$. Also, our mathematical model closely matches the simulation result with an average 0.0243 difference, which is just under one percent of the peak profit. In addition, using (12) and (13) to compute Y_w and $Y_{w\&p}$ is quite efficient since the computation of $PSR(i,j,k)$ is fast. With a modern PC, it takes less than one second to obtain all the profits shown in Table XVII with our mathematical model. However, it takes around 800 minutes to perform one 500 K-sample random simulation.

E. Computing $PSR(k)$ on ATE

The computation of $PSR(k)$ includes two steps: (1) computing the four-dimensional array $S^{(x)}[m][n][z]$, and (2) computing $PSR(k)$ based on (1). To compute the first step takes much longer than the second step. Take the above 512 MB NROM-based ROM (8192x4096x16) as an example. The runtime of computing $S^{(x)}[m][n][z]$ and the remaining (1) is 5.5 ms and 7.3 μs, respectively, on a PC with 2.4 GHz Intel Pentium Dual CPU and 3.5 GB DDR SDRAM. The memory requirement for $S^{(x)}[m][n][z]$ is about 844.8KB (100x16x6x22x4B).

However, the computational power of an ATE is not as strong as that of a modern PC. Take the wafer tester, Credence Kalos, as an example. This wafer tester can support 16 test sites and its test controller utilizes a Intel i960 Processer along with 64 MB DRAM. The runtime of computing the first and second steps on this tester is 12 s and 0.012 s, respectively. In fact, we can compute the four-dimension $S^{(x)}[m][n][z]$ with a large number of x for one time and repeatedly use the same $S^{(x)}[m][n][z]$ for each tested part, which can further reduce the test-application time. Note that the test application time of the conventional wafer test for a 512 MB NROM-based ROM is about several minutes. The test application time of the conventional package test is about twice more as that of the conventional wafer test. The runtime overhead of computing $PSR(k)$ on a wafer tester is hence relatively small.

F. Overhead of the Proposed Test Flow

Unlike the conventional package test, which only needs to program the customer's code and then read out the programmed code for comparison, the proposed package test needs to apply the repair analysis to each tested part and then program the customer's code to the spare rows and columns. Because the ATEs used in our conventional package test cannot support the parallel computation of repair analysis, these extra tasks of repair analysis may increase the test application time of the package test. In addition, the time-to-delivery for NROM-based ROMs is usually determined by the throughput of its package test, not its wafer test, since a blank packaged NROM-based ROM can be manufactured in advance and waits for the order from customers. Therefore, the test-time overhead in the proposed package test may directly affect the overall time-to-delivery.

Take the package tester AdvanTest T5377S, which has been used in the conventional package test for a NROM-based ROM product line, as an example. This package tester can test 64 DUTs in parallel but can only perform repair analysis in serial. The test application time for the conventional package test is around 90 seconds (without programming the code). Performing a single repair analysis on this package tester takes around 200 ms in average. The time for performing 64 repair analysis serially is around 12.8 seconds. To enable the capability of parallel repair analysis, we can add 8 Advantest FMRA (Failure Memory Repair Analyzer) boards onto the package testers, where each FMRA board can perform repair analysis for 8 DUTs simultaneously and costs around 90 K USD. In other words, if we want to remove the 12.8 sec test-time overhead from the proposed test flow, we need to spend another 720 K USD on enabling the parallel repair analysis. Note that a package tester, AdvanTest T5377S, costs around 1.25 M USD and a hander, AdvanTest M6542, costs around 320 K USD.

G. Reliability Issues of Using a Bad NROM Bit to Represent Value 0

For the experiment conducted for Fig. 6, we actually baked each chip at 80°C for 24 hours after we program all NROM bits to 0. Thus, the result shown in Fig. 6 shows certain degree of reliability for using the bad NROM bits to represent a good value 1. However, such an experiment is still far away from enough to prove the reliability since we only baked the chips but didn't operate them at the high temperature like general burn-in test. Also, even with burn-in test, the result can only reflect the infant-mortality rate of the chips but cannot reflect the product lifetime. To foresee the product's failure rate occurring after certain months or years of usage, we need to apply reliability test, such as THB (temperature, humidity, bias) test [17], HAST (highly-accelerated temperature and humidity stress) test [18], and HTOL (high temperature operating life) test [19], which are all very expensive in the current industry and time-consuming (may take even weeks such as THB test). So far we don't have any experimental result from a reliability test.

However, the reliability issue of using a bad NROM bit to represent value 0 is not like placing a wear-out defect in a chip, such as a thin wire which may turn into an open defect up after a certain period of time due to the electron migration. A bad NROM bit here is just a NROM bit whose V_{th} is too high to represent a good value 1, so that we only use a bad NROM bit represent value 0. Also, the reliability issue mentioned in [8] only occurs after multiple program and erase operations, which is not the case for the one-time-programming application as used in this paper. Furthermore, from the result shown in Table I, a bad NROM bit can actually be programmed to a higher V_{th} in average than a good NROM bit, meaning that a bad NROM bit can represent a better value 0 than a good NROM bit even though a bad NROM bit cannot represent a quality value 1. Based on this fact, the reliability of using a bad NROM bit to represent a value 0 should be higher than using a good NROM bit to represent a value 0 since it is more difficult to wear out a value 0 with a higher V_{th} and lower its V_{th} to a value 1. In addition, for a potential reliability fault induced by positive bias temperature instability (NROM device is a n-MOSFET structure), the V_{th} of an NROM bit will be further increased and even help the NROM bit to represent a better value 0. Thus, we should in fact worry more about the reliability of the good NROM bits used to represent value 1 rather than that of the bad NROM bits used to represent value 0, which is a problem irrelative to the use of our proposed test flow.

VI. Conclusion

In this paper, we first introduced the basic operations of an NROM cell and the conventional test flow of NROM-based ROMs. Secondly, we introduced the proposed test flow for NROM-based ROMs, which applies the repair analysis after the customer's code is programmed such that the repair analysis is performed based on the data-level defects instead of physical-level defects. The statistics collected from real NROM-based ROMs then demonstrated the feasibility of the proposed test flow. Next, we developed an estimation scheme to predict the probability that a packaged part can be successfully repaired in the package test based on only the defect information obtained in the wafer test. Also, a series of experiments based on different parameter combinations were conducted to validate the accuracy of the estimation scheme and the superiority of the proposed test flow over the conventional one. Last, we discussed the computational complexity of the estimation scheme and the overhead of the proposed test flow.

References

[1] D. Kahng and S. M. Sze, "A floating gate and its application to memory devices," *Bell Syst. Tech. J.*, vol. 46, p. 1288, 1967.

[2] D. Frohman–Bentchkowsky, "The metal-nitride-oxide-silicon (MNOS) transistor—characteristics and applications," *Proc. IEEE.*, vol. 58, no. 8, pp. 1207–1219, Aug. 1970.

[3] Y. L. Yang and M. H. White, "Charge retention of scaled SONOS nonvolatile memory devices at elevated temperatures," *Solid-State Electron.*, vol. 44, pp. 949–958, 2000.

[4] C. T. Swift, G. L. Chindalore, K. Harber, T. S. Harp, A. Hoefler, C. M. Hong, P. A. Ingersoll, C. B. Li, E. J. Prinz, and J. A. Yater, "An embedded 90 nm SONOS nonvolatile memory utilizing hot electron programming and uniform tunnel erase," *IEDM Tech. Dig.*, pp. 927–930, 2002.

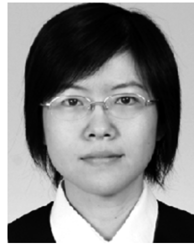
[5] S. Habermehl, R. D. Nasby, M. Rightley, and P. R. Mahl, "Endurance of SONOS NVM stacks prepared with nitrided Si(100)/SiO interfaces," in *IEEE Non-Volatile Semiconductor Memory Workshop*, Monterey, CA, 1998, vol. 66.

[6] B. Eitan, P. Pavan, I. Bloom, E. Aloni, A. Frommer, and D. Finzi, "NROM: A novel localized trapping 2 bit nonvolatile memory cell," in *IEEE Non-Volatile Semiconductor Memory Workshop*, Monterey, CA, 1998, vol. 66.

[7] M. Y. I. Liu, Y. W. Chang, N. K. Zous, I. Yang, T. C. Lu, T. Wang, W. T. J. Ku, and C. Y. Lu, “Temperature effect on read current in a two-bit nitride-based trapping storage flash eeprom cell,” *IEEE Electron Device Lett.*, vol. 25, pp. 495–497, 2004.
[8] A. Shappir, E. Lusky, G. Cohen, I. Bloom, M. Janai, and B. Eitan, “The two-bit NROM reliability,” in *Proc. IEEE.*, 2004, vol. 4.
[9] [Online]. Available: http://www.mxic.com.tw/
[10] C.-Y. Chin, Y.-T. Tsou, C.-M. M. Chang, and C.-T. Chao, “A novel test flow for one-time-programming applications of NROM technology,” in *IEEE Int. Test Conf.*, 2009.
[11] Y. Zorian and A. Ivanov, “An effective BIST scheme for ROM’s,” *IEEE Trans. Computers*, vol. 41, pp. 646–653, 1992.
[12] A. J. van de Goor, *Testing Semiconductor Memories, Theory and Practice*. Gouda, The Netherlands: ComTex, 1998.
[13] T. M. Schwair and H. C. Ritter, “Complete self-test architecture for a coprocessor,” in *IEEE Int.Test Conf.*, 1990, pp. 886–890.
[14] A. J. van de Goor and I. Schanstra, “Address and data scrambling: Causes and impact on memory tests,” in *Proc. 1st IEEE Int. Workshop on Electron. Design, Test, Appl. (DELTA 02)*, 2002, pp. 128–136.
[15] R.-F. Huang, C.-H. Chen, and C.-W. Wu, “Economic aspects of memory built-in self-repair,” *IEEE Des. Test Comput.*, vol. 24, no. 2, pp. 164–172, Mar./Apr. 2007.
[16] R.-F. Huang, J.-F.-Li, J.-C. Yeh, and C.-W. Wu, “Raisin: Redundancy analysis algorithm simulation,” *IEEE Des. Test Computers*, vol. 24, no. 3, pp. 386–396, May/Jun. 2007.
[17] Electronic Industries Assoc. and JEDEC Solid State Technol. Assoc., “Steady state temperature humidity bias life test,” EIA/JESD22-A101-B, 1997.
[18] Electronic Industries Assoc. and JEDEC Solid State Technol. Assoc., “Highly-accelerated temperature and humidity stress test,” EIA/JESD22-A110-B, 2008.
[19] JEDEC Solid State Technol. Assoc., “Temperature, bias, and operating life,” JESD22-A108C, 2005.

Mango C.-T. Chao received the B.S. and M.S. degrees from the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, in 1998 and 2000, respectively, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of California, Santa Barbara, in 2006.

Since 2006, he has been with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, as an assistant professor. His research interests include memory testing, on-chip test compression/decompression, WAT test-structure design, power-related testing, and physical design automation.



Ching-Yu Chin received the B.S. degree from the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, in 2008, and is currently working towards the Ph.D. degree in Institute of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan.

Her research interests include physical design automation and VLSI testing.

Yao-Te Tsou received the M.S. degree from the Department of Institute of Electrical Control Engineering, National Chiao Tung University, Hsinchu, Taiwan, in 2008.

He is currently with Macronix International Co., Ltd., Taiwan. His research interests include the design and testing of non-volatile memories.

Chi-Min Chang received the B.S. degree from the Department of Electrical Engineering at National Central University, Taoyuan, Taiwan, in 2006, and the M.S. degree from the Department of Electronics Engineering at National Chiao Tung University, Hsinchu, Taiwan, in 2008.

Since 2008, he has been with Taiwan Semiconductor Manufacturing Company, and has worked on test-structure design and WAT testing.