SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

October 2016, Vol. 59 102401:1–102401:11
doi: 10.1007/s11432-015-5437-0

A snake addressing scheme for phase change memory testing

Xiaole CUI[1]*, Zuolin CHENG[1], Chunglen LEE[1], Xinnan LIN[1], Yiqun WEI[1], Xiaogang CHEN[2] & Zhitang SONG[2]

[1]*Key Laboratory of Integrated Microsystems, Peking University Shenzhen Graduate School, Shenzhen* 518055*, China;*
[2]*State Key Laboratory of Functional Materials for Informatics, Shanghai Institute of Micro-System and Information Technology, Chinese Academy of Sciences, Shanghai* 200050*, China*

Received July 17, 2015; accepted August 10, 2015; published online February 1, 2016

Abstract Phase change memory (PCM) is one of the most promising candidates for next generation nonvolatile memory. However, PCM suffers from a variety of faults due to its special device structure and operation mechanism. A snake addressing scheme is introduced into the test algorithms of PCM to reduce the test time and excite proximity disturb faults more effectively. The March test algorithm with the proposed snake addressing scheme is less complex than most traditional test algorithms. In addition to conventional faults, it is capable of covering disturb and parasitic faults. Moreover, when incorporated with the sneak path testing method, it is able to test the read fault, read recovery fault, incomplete program fault 0, and false write fault.

Keywords phase change memory, fault model, snake addressing mode, March test algorithm, sneak path test method

Citation Cui X L, Cheng Z L, Lee C L, et al. A snake addressing scheme for phase change memory testing. Sci China Inf Sci, 2016, 59(10): 102401, doi: 10.1007/s11432-015-5437-0

1 Introduction

Phase change memory (PCM) is a variable resistance device that consists of a top electrode, phase change material layer, heater, and bottom electrode [1,2]. It utilizes two stable states of chalcogenide material, such as $Ge_2Sb_2Te_5$ (GST), to store information. The PCM cell changes into the amorphous state, i.e., the RESET state or logical '0' state, if the phase change material is heated above its melting temperature and cooled rapidly, which leads to a high resistance state in the memory device. The amorphous state is achieved by applying a narrow, high amplitude electrical pulse to the heater of the PCM cell. While the PCM cell changes into a low resistance crystalline state, i.e., the SET state or logical '1' state, the phase change material is heated to a temperature above its crystalline temperature but lower than its melting point. This is implemented by applying a moderate amplitude electrical pulse for a period of time to the heater of the PCM cell. The state of the PCM cell is read out by applying a small current to its electrode. Because of its short access time, high memory density, low operation power, high endurance,

* Corresponding author (email: cuixl@pkusz.edu.cn)

good data retention, and technology compatibility with the CMOS process, PCM is regarded as one of the most promising candidates for next generation nonvolatile memory [3].

However, PCM suffers from a variety of faults due to its special material, device structure, and operation mechanism. The presence of contaminants or other impurities in the active region of the phase change material may result in parasitic parallel conductive paths in the PCM cell. These resistive or capacitive parasitic effects lead to an incomplete program fault 0 (IPF0), weak transition fault 0 (WTF0), write 1 destructive fault (WDF1), or weak write 1 destructive fault (WWDF1), depending on the value of the parasitic resistance and capacitance [4–6]. A read recovery disturb (RRD) fault, which is caused by the non-equilibrium state in the newly programmed cell, may occur when a cell is read immediately after it has been programmed to the RESET state [7]. The regular read current is small and not able to generate a temperature over the crystalline temperature of the phase change material. However, the current flowing through the phase change material region may actually increase if defects present, which will result in a read (RD) fault that destroys the information stored in the PCM cell [8,9]. Since the SET state requires a lower temperature than the RESET state, only the crystalline state is taken into consideration when discussing a RD fault. A false write (FWR) fault may occur, which changes the addressed cell from the SET state to the RESET state when the read operation is performed if the write current circuitry of the PCM cell is falsely activated, especially in a radiation environment [10]. A stuck-at SET (SS) fault is caused by the inter diffusion between the phase change material of PCM and its adjacent material, or circuit errors, such as a short connection between the bit line and ground (GND) [9,11]. A stuck-at RESET (SR) fault occurs if the contact region between the top electrode and phase change material is worn out [12,13] or if circuit errors are present in the PCM array, such as bridging between the word line and GND [11]. Since the PCM cell changes its state according to the Joule heat generated by the input current pulse, the thermal crosstalk between the neighboring cells may lead to a proximity disturb fault (PDF). A comprehensive summary of the PCM fault models can be found in [4,11].

To cover the faults of PCM, new PCM test algorithms had been designed and discussed. Mohammad et al. proposed a March-like algorithm, which has a complexity of $11mn$, where m and n are the number of rows and columns, respectively, to test stuck-at faults and disturb faults of PCM [11]. Furthermore, they improved the test algorithm to reduce the complexity of the algorithm to $8mn$ [4]. Because of the linear addressing mode, the algorithms restrain their capability to excite PDFs because they are unable to utilize all four neighboring cells to disturb the victim cell. Zhang et al. [14] applied the modified March algorithm to test program interference faults, RD faults, and write '1/0' faults and observed that the write '1' fault is the dominant type of fault that linearly increases with the number of program cycles. Kannan et al. [15] applied a sneak path method to test nonvolatile memories with a bidirectional selector in each cell. Their algorithm has a complexity of only $3.8mn$ when it is applied to PCM because the sneak path test method tests the PCM array by a unit of the Region of Detection (RoD), which is a group of memory cells that are able to be tested simultaneously rather than individually.

In this paper, new test algorithms are proposed to overcome the disadvantages of the above algorithms. These algorithms adopt a new addressing scheme called snake addressing to more effectively excite a PDF. For the victim cell, the new addressing scheme enables the write operations of its four adjacent cells before the read operation of the cell to be invoked, making a more stringent test condition for the faults incurred by the thermal crosstalk between cells. Furthermore, it incorporates a March-like test algorithm with the snake addressing scheme to reduce the test complexity to $7mn$, and the algorithm has the capability of testing all fault types in [4] and the faults incurred by the parasitic effects. Moreover, it applies the snake addressing scheme to the sneak path test method to extend the test capability of the test algorithm to cover an RD fault, read recovery fault, IPF0, and FWR fault.

The remainder of this paper is organized as follows. First, the snake addressing scheme is defined and demonstrated in Section 2. Then, the proposed algorithms with and without the consideration of the sneak path test method are presented and discussed in detail in Section 3. Finally, conclusion is drawn in Section 4.

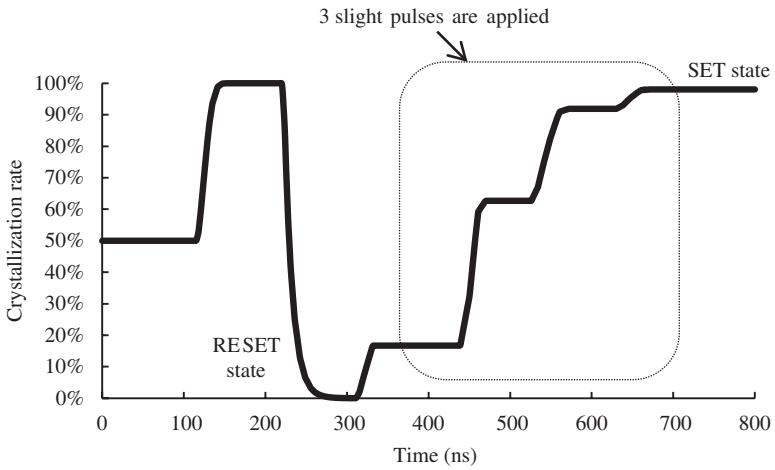

Figure 1 Simulation of the disturb accumulation phenomenon with an initial RESET state of the victim cell.

2 The snake addressing mode

2.1 Disturb accumulation phenomenon

A special disturb accumulation phenomenon is simulated for the thermal crosstalk fault as follows. Three consecutive slight pulses that have lower magnitude than those of the normal programming pulses are applied sequentially to a victim cell with an initial RESET state, imitating the thermal crosstalk caused by write '0' operations at its neighboring cells; the simulation results are shown in Figure 1. In this figure, the resistance of the PCM cell is presented by the crystallization rate of its phase change material. The simulation is carried out with a PCM compact Simulation Program with Integrated Circuit Emphasis (SPICE) model [16].

Observe that a single slight pulse dose not lead to the crystallization state of the victim cell; however, several consecutive light pulses are able to eventually accumulate enough energy to change the cell to the crystallization state, leading to a state transition of the victim cell from the RESET to SET state. This implies that the more write '0' operations at its neighboring cells, the more effective it is at exciting the PDF of the victim cell; that is, the traditional test condition of the PDF [4,11], which only excites two neighboring cells of the victim cell, requires further improvements to enhance its effectiveness. In fact, the amorphous state of the phase change material of the PCM cell is less stable than the crystalline state and consequentially, more vulnerable to proximity disturbs. Hence, for the PDF, the disturbance of the RESET state in the victim cell is the dominated case. Thus, the only case that must be considered is the unintentional loss of information in the victim cell with the RESET state, which occurs when the neighboring cells are programmed to the RESET state. The PDF is modeled as $\langle$x, w0; $0/1_{\mathrm{m}}/\text{-}\rangle$, where 1_{m} is the marginal value of the quasi-SET state incurred by the PDF. For the activation of PDFs, a tradeoff is necessary between an improvement in the test effectiveness for the PDF and an acceptable complexity of the test algorithm. Hence, the test condition of the PDF was redefined as follows: each victim cell is read after all of its four neighboring cells have performed the write '0' operation at least once before any other operations at the victim cell.

Furthermore, observe that after every slight pulse, the crystallization state of the phase change material of the victim cell stays at its current state and does not return to its original state, unless a new programming pulse is applied to refresh the current state. This implies that the effects of crosstalk are accumulated on the victim cell, even when write '0' operations are applied to the neighboring cells in a non-continuous manner. These characteristics differ from those of coupling faults or disturb faults of electron based memory devices such as Flash or dynamic random-access memory (DRAM), in which the negligible effects of disturbance or coupling of the victim cell may disappear gradually because of leakage.

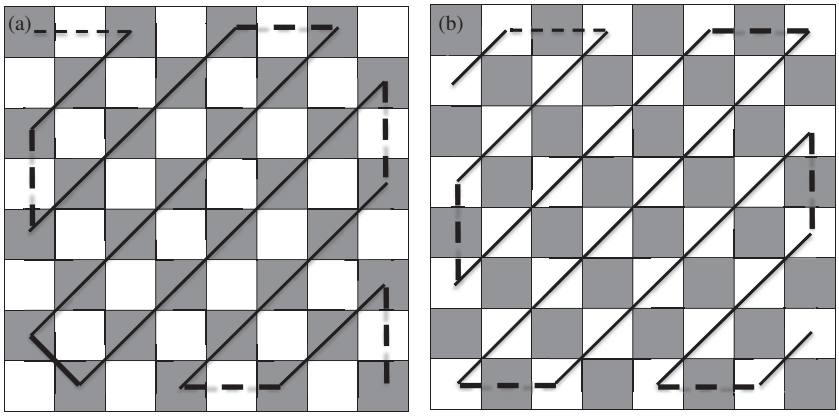

Figure 2 Snake addressing mode (a) for black cells and (b) white cells.

2.2 The snake addressing mode

Based on the observations above, a new addressing mode, namely the snake addressing mode, which addresses the memory cells during testing, is proposed to utilize all four neighboring cells of the victim cell. Similar to the checkerboard algorithm, the snake addressing mode divides the memory cells into two groups, which are shown as black cells and white cells in Figure 2. For each black cell, all four neighboring cells are white, and vice versa for each white cell. Figure 2(a) and (b) illustrates the snake addressing mode for the operations of black cells and white cells, respectively, where the solid lines denote the operations to black/white cells, and the dashed lines denote skipping the corresponding white/black cells. Here, the symbols ⬊ and ⬁, which are variations of the notation used by the traditional March algorithm, are used to denote snake addressing for the black cells and white cells, respectively.

To test the PDFs of the PCM array, a four-step march-like test algorithm is designed as follows:

$$\text{March-PDF} = \{\text{M1:}\Updownarrow(\text{w0});\ \text{M2:}⬁\ (\text{w0});\ \text{M3:}⬊\ (\text{r0, w0});\ \text{M4:}⬁\ (\text{r0})\}. \tag{1}$$

The algorithm follows the extended notation of the March algorithm, where $\Updownarrow$ denotes the arbitrary addressing sequence in a linearly increasing ($\Uparrow$) or decreasing ($\Downarrow$) order, wx denotes the operation that writes a value of x to the memory cells, and rx denotes a read operation expecting a value of x. The algorithm initializes all cells into the RESET state. It then performs write '0' operations for each white cell and black cell with the snake addressing mode in Step 2 and Step 3, and excites the PDFs with the RESET operations at all neighboring cells. For the boundary cells and corner cells, three and two neighboring cells are accessed, respectively. For the other cells in the array, four neighboring cells are programmed to the RESET state. Finally, the state of each victim cell is read expecting a value of '0' in Step 3 and Step 4 for the black cells and white cells, respectively. This algorithm completely satisfies the test condition of the PDF.

3 Proposed test algorithms

3.1 March test algorithm incorporated with snake addressing

3.1.1 *The proposed March test algorithm*

The PCM fault models and associated test conditions are summarized in Table 1. In Table 1, the faults are modeled with traditional notation $\langle S/F/R\rangle$ [17], where S$\in$\{w0, w1, r0, r1, 1, 0\} is the fault sensitizing condition, F$\in$ $\{0, 0_{\rm m}, 1_{\rm m}, 1\}$ is the faulty cell value, R$\in$\{0, 1, -\} is the output of the read operation, $0_{\rm m}$ and $1_{\rm m}$ are the marginal values of the PCM cell incurred by special faults, and '-' is used to denote if a write operation is performed. For the fault model and test condition of the PDF, the operations before and after ';' are performed at aggressor cells and the victim cell, respectively.

Table 1 Fault models and test conditions of PCM

Failure name	Fault model	Test condition
Stuck at SET fault (SS)	$\langle\forall/1\rangle$	w0/r0
Stuck at RESET fault (SR)	$\langle\forall/0\rangle$	w1/r1
Incomplete program fault 0 (IPF0)	$\langle x$,w0/1/-$\rangle$	w0/r0, r0
Weak transition fault 0 (WTF0)	$\langle x$,w0/$1_{\rm m}$ / - $\rangle$	w0/r0
Write 1 destructive fault (WDF1)	$\langle x$, w1/0/- $\rangle$	w1/r1
Weak write 1 destructive fault (WWDF1)	$\langle x$,w1/$1_{\rm m}$/-$\rangle$	w1/$r1_{\rm m}$
Proximity disturb fault (PDF)	$\langle x$, w0; 0/$1_{\rm m}$/-$\rangle$	x, $w0_{\rm neighbors}$; r0
Read recovery disturb fault (RRD)	$\langle$1,w0, r0/0/$1_{\rm m}\rangle$	x, w0, r0(following w0)
Read fault (RD)	$\langle$0, r0/$1_{\rm m}$/0$\rangle$	0, r0, r0
False write fault (FWR)	$\langle$1, r1/0/1$\rangle$	1, r1, r1

```
Define addressing path A: //
i = 1; j = –1;
while ( i <= 2*max(n,m) )
{ if ( (i+j) % 4 == 2)
   { i++; j–; }
else                              // (i+j) % 4 ==0
   { i—; j++;}
if (i ==–1)                       // out of array
   { i = 0; j++;}
if (j ==–1)                       // out of array
   {j =0; i++;}
if((i > (n–1)) ||( j>(m–1)))   //out of array
   { continue; }
}
```

```
Define addressing path B: //
i = 2; j = –1;
while ( i <= 2*max(n,m) )
{ if ((i+j) % 4 == 3)
   { i++; j—;}
else                              // (i+j) % 4 ==1
   { i—; j++;}
if (i ==–1)                       // out of array
   { i = 0; j++;}
if (j ==–1)                       // out of array
   { j=0; i++;}
if((i >(n–1))||( j>( m–1))) // out of array
   { continue;}
}
```

```
March-SA Algorithm:
{ Step 1: for addressing path A:
     {write 0 to C_{i,j};}
  Step 2: for addressing path B:
     {write 0 to C_{i,j}; read C_{i,j};}
  Step 3: for addressing path A:
     {read C_{i,j}; write 0 to C_{i,j}; read C_{i,j};}
  Step 4:
     for (i=0; i<n; i++)
        for (j=0; j<m; j++)
           {read C_{i,j}; write 1 to C_{i,j}; read C_{i,j}; }
  Step 5:
     for (i=0; i<n; i++)
        for (j=0; j<m; j++)
           {read C_{i,j};}
}
```

Figure 3 Pseudo code of the March-SA algorithm.

A March test algorithm is designed incorporating with the snake addressing scheme to test all of the faults listed in Table 1. The proposed March test algorithm with the snake addressing, namely, the March-SA algorithm, is given by

$$\text{March-SA} = \{\text{M1:}\searrow\ (\text{w0});\ \text{M2:}\nwarrow\ (\text{w0, r0});\ \text{M3:}\searrow\ (\text{r0, w0, r0});\ \text{M4:}\updownarrow\ (\text{r0, w1, r1}_{\rm m});\ \text{M5:}\updownarrow\ (\text{r1})\}, \quad (2)$$

where the $1_{\rm m}$ state is a weak 1 state, the corresponding resistance of the cell is usually larger than that of state 1. For example, the typical resistive values of state 0, $1_{\rm m}$, and 1 are approximately 270 k, 159–34 k, and 4.5 k ohms, respectively, with the PCM SPICE model in [16]. The pseudo code of the March-SA algorithm is illustrated in Figure 3.

Assume that the address range of the memory cells in an array is from (0, 0) to $(n-1, m-1)$. The notation $\searrow$ and $\nwarrow$ in the March-SA algorithm is defined as addressing path A and addressing path B, respectively, in the pseudo code of Figure 3. For the addressing paths $\searrow$ and $\nwarrow$ in Figure 2, the values of the black/white cells in the same quasi-diagonal line are equal to $i+j$, where (i, j) is the location of a cell in the memory array. Furthermore, the direction of the quasi-diagonal lines in the addressing path is determined by the result of $(i+j)$ mod 4. For $\searrow$, if the result of $(i+j)$ mod 4 is 2, the corresponding quasi-diagonal line ramps down, and if the result is 0, it ramps up; otherwise, the corresponding cells are not in the addressing path. For $\nwarrow$, if the result of $(i+j)$ mod 4 is 3 or 1, the corresponding quasi-diagonal line ramps down or up, respectively; otherwise, the corresponding cells are not in the addressing path.

Using the proposed March-SA algorithm, the fault coverage for all faults listed in Table 1 is analyzed as follows:

SR fault: The w1 operation in Step 4 excites the SR faults for all cells. The faults are then tested by the r1 operation in Step 5.

SS fault: The w0 operations in Step 1 and Step 2 reset all memory cells. The SS faults are then tested

Table 2 Fault coverage capability of the proposed March-SA algorithm

Fault	Testing algorithm			Fault	Testing algorithm		
	Initial state	Excitation	Test		Initial state	Excitation	Test
SR	–	M4:w1	M5:r1	WWDF1	–	M4:w1	M4:$r1_m$
SS	–	M1:w0 M2:w0	M2:r0 M3:r0	PDF	M1:w0 M2:w0	M2:w0 M3:w0	M3:$r0_{1st}$ M4:r0
IPF0	–	M2:w0 M3:w0	M2:r0;M4:r0 M3:r0;M4:r0	RRD	–	M2:w0 M3:w0	M2:r0 M3:$r0_{2nd}$
WTF0	–	M1:w0 M2:w0	M2:r0 M3:r0	RD	M1:w0 M2:w0	M2:r0 M3:r0	M4:$r0_m$
WDF1	–	M4:w1	M5:r1	FWR	M4:w1	M4:$r1_m$	M5:r1

by r0 operations in Step 2 and Step 3.

IPF0: The w0 operations in Step 2 and Step 3 excite the IPF0s for the white and black cells, respectively. The IPF0s for the white cells are subsequently tested by r0 in Step 2 and r0 in Step 4, and the IPF0s for the black cells are tested by the second r0 in Step 3 and r0 in Step 4.

WTF0: The test condition is the same as that of an SS fault, i.e., it is tested thoroughly by the algorithm.

WDF1: The faults are excited by w1 operations in Step 4 and are then tested by r1 in Step 5.

WWDF1: A WWDF1 is excited and tested by w1 and $r1_m$ in Step 4, respectively.

PDF: All black cells are initialized to the RESET state in Step 1, and their PDFs are excited in Step 2 by writing '0' to their neighbors (white cells). Then, the PDFs of the black cells are tested by the first r0 operation in Step 3. Similarly, the white cells are initialized to the RESET state in Step 2, their neighboring cells (black cells) are written to '0' in Step 3, and the PDFs are tested by the r0 operation in Step 4.

RRD fault: The RRD faults of white cells are excited by the w0 operation in Step 2. Then, the faults are tested by the r0 operation in Step 2, which follows the w0 operation. For the black cells, the w0 operation in Step 3 excites the RRD faults, and the second r0 operation in Step 3, followed by the w0 operation, tests the fault.

RD fault: The RD faults of black and white cells are initialized to zero by the w0 operation in Step 1 and Step 2, respectively. The faults for the white cells are excited and tested by r0 in Step 2 and r0 in Step 4, while the faults for the black cells are excited and tested by r0 in Step 3 and r0 in Step 4, respectively.

FWR fault: The w1 operation in Step 4 initializes all cells to the SET state. Then, the FWR faults are excited by $r1_m$ in Step 4 and are tested by the r1 operation in Step 5. Both $r1_m$ and r1 operations apply the same current pulse to the electrode of the PCM cells; the only difference between them is that they have different reference voltages. For this reason, $r1_m$ in Step 4, which is used to excite FWR faults, is substituted for the r1 operation in the test condition.

The fault coverage for each fault is summarized in Table 2.

3.1.2 *Comparison and discussion*

Almost all of the reported traditional PCM test algorithms are March test algorithms with a traditional linear addressing scheme [4–6]. The proposed snake addressing mode is compared with linear addressing in Figure 4.

Assume that the black cell $C_{i,j}$ is the victim cell; it has four neighboring white cells labeled by $C_{i-1,j}$, $C_{i,j-1}$, $C_{i+1,j}$, and $C_{i,j+1}$. For linear addressing, only $C_{i,j+1}$ and $C_{i+1,j}$ are accessed before the victim cell is accessed along the addressing path as shown in Figure 4(a). Similarly, for the reverse addressing path, only $C_{i,j-1}$ and $C_{i-1,j}$ are accessed before addressing $C_{i,j}$. Hence, for linear addressing, the white and black cells are accessed alternately, and only two neighboring cells are accessed. In contrast, for snake addressing, as illustrated in Figure 4(b), all of the white and black cells are accessed in one diagonal addressing path, and all four neighboring cells are accessed before any operation at the victim

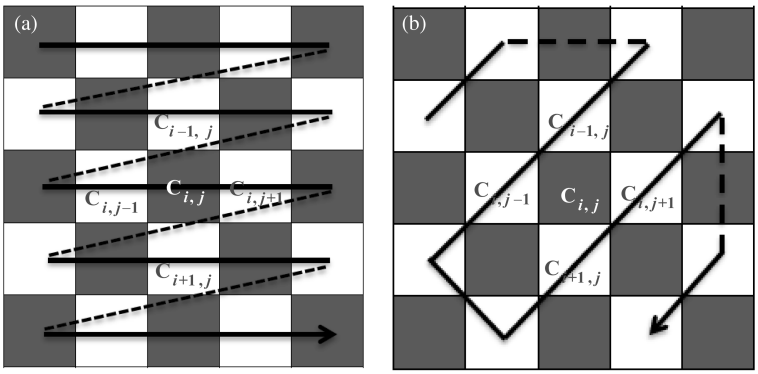

Figure 4 Effect of the addressing mode on PDFs for (a) linear addressing (b) and snake addressing of white cells.

Table 3 Comparison of the proposed March test algorithm with those in [4] and [11]

Algorithm	SAF	IPF0	WTF0	WDF1	WWDF1	PDF	RRD	RD	FWR	Complexity
March-PC [11]	Y	Y	Y	Y	Y	Y_2	Y	Y	Y	$11mn$
March-PCM [4]	Y	Y	Y	Y	Y	Y_2	Y	Y	Y	$8mn$
March-SA	Y	Y	Y	Y	Y	Y_4	Y	Y	Y	$7mn$

cell. To test the PDF, snake addressing has double the number of RESET operations of neighboring cells compared to linear addressing. This excites the PDF of a victim cell more effectively because of the disturb accumulation.

In terms of algorithm complexity, the proposed algorithm has an obvious advantage because only half of the memory cells are accessed by snake addressing. For a PCM array with m rows and n columns, the complexity of the five steps of the proposed algorithm is $mn/2$, $2mn/2$, $3mn/2$, $3mn$, and mn, respectively, which results in an overall complexity of $7mn$ for the proposed algorithm; this is less than the complexity of other published PCM test algorithms [4,11]. Table 3 compiles the fault coverage capability and complexity of our algorithm and those in [4,11], where Y indicates that the algorithm is capable of testing the corresponding fault, Y_2 indicates that the algorithm only accesses two neighboring cells, and Y_4 indicates that it accesses four neighbors to test the PDF. Clearly, all algorithms cover all the listed fault types, but the proposed algorithm is able to test the PDF more effectively with a lower complexity.

3.2 Enhancement of the sneak path test algorithm

3.2.1 *Analysis of the sneak path test algorithm*

In 2013, Kannan et al. determined that utilizing sneak paths inherent in the PCM or resistive random access memory (RRAM) array leads to a dramatic reduction in the testing time [15]. For a cross-shaped RoD of PCM, only the center cell needs to be tested. With this method, Ref. [15] proposed a sneak-path PCM testing algorithm as follows:

$$\text{Sneak-path-PCM} = \{\text{M1:}\Uparrow\text{(w0); M2:}\uparrow\text{(r0); M3:}\Uparrow\text{(w1); M4:}\uparrow\text{(r1); M5:}\downarrow\text{(r1); M6:}\Downarrow\text{(w0); M7:}\downarrow\text{(r0)}\}. \quad (3)$$

In this algorithm, $\uparrow$ and $\downarrow$ denote the addressing modes following the sneak path testing method with an increasing address order and decreasing address order, respectively. If this sneak path addressing mode is used, only one fifth of the cells are accessed, for there are five cells in one RoD, except for RoDs on the boundary or corner positions. Hence, for an array with m rows and n columns, the complexity of this algorithm is only $3.8mn$ [15].

However, we found that not all types of faults in Table 1 are covered by this original sneak path PCM test method. The analysis of the faults that are not tested effectively by this sneak path PCM test algorithm is as follows:

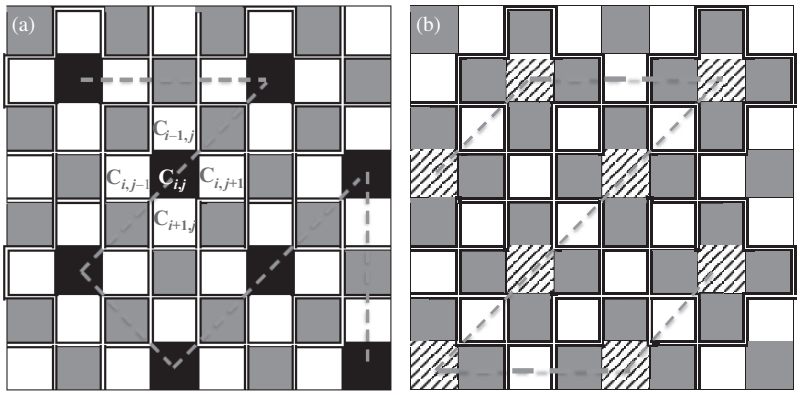

Figure 5 March-SASP model (a) for black center cells and (b) for white center cells.

PDF: PDFs need two phases to cover all of the PDFs in the memory array. In phase 1, the w0 operation in Step 1 resets all of the memory cells, initializes the victim cell, and excites its PDF by accessing two of its neighboring cells. Then, r0 in Step 2 tests the PDFs in sneak path mode. Similarly, in phase 2, the w0 operation in Step 6 resets the other two neighboring cells, and r0 in Step 7 tests the PDF. Although this test algorithm is able to test PDFs, it only programs two of the neighboring cells of the victim cell with the RESET state in one test phase.

IPF0: This type of fault is not tested by this algorithm because no two r0 operations exist before a write operation in the algorithm.

RRD fault: The testing of RRD faults requires them to be immediately read after a w0 operation. However, the sneak path testing algorithm separates all of the read and write operations into different steps; thus, it is not able to meet the test condition of the RRD fault.

RD fault: Only one fifth of the cells are addressed to read in sneak path testing, while the excitation condition of an RD fault requires a read operation for each cell; thus, this algorithm is not able to cover all RD faults.

FWR fault: This type of fault is not tested by this algorithm because the r1 operation with sneak path addressing in Step 4 is not able to support the excitation requirement.

3.2.2 *Enhanced sneak path test algorithm*

The snake addressing scheme is applied to the sneak path test method to enhance its testing capability. The proposed sneak path test algorithm incorporated with the snake addressing mode, namely, the March-SASP algorithm, is as follows:

$$\begin{aligned}&\text{March-SASP}\\&\quad=\{\text{M1:}\updownarrow(\text{w1,r1}_\text{m});\ \text{M2:}\updownarrow(\text{r1});\ \text{M3:}\searrow(\text{w0,r0});\ \text{M4:}\searrow(\text{w0,r0});\ \text{M5:}\searrow(\text{r0});\ \text{M6:}\searrow(\text{w0});\ \text{M7:}\searrow(\text{r0})\ \},\end{aligned}\quad(4)$$

where $\updownarrow$ means the address path is arbitrarily selected from the sneak path addressing path, $\uparrow$ or $\downarrow$. New addressing schemes $\searrow$ and $\searrow$ are introduced in Step 5 and Step 7 of the algorithm, respectively, where the white cells and black cells are accessed using the sneak path test method incorporated with the snake addressing. Figure 5 illustrates the addressing scheme operations of $\searrow$ and $\searrow$. For every four white cells $C_{i,j-1}$, $C_{i-1,j}$, $C_{i+1,j}$, and $C_{i,j+1}$ that share the same center cell $C_{i,j}$ (black cell), only one measurement of the current on the bit line for the addressed cell $C_{i,j}$ is needed to detect the existence of faults in these five cells; hence, for a memory array with n rows and m columns, the number of operations needed to test white/black cells in the array with the March-SASP algorithm is roughly $mn/2/4 = 0.125mn$. The total complexity of the March-SASP algorithm is $2mn + 0.2mn + 2 \times 0.5mn + 2 \times 0.5mn + 0.125mn + 0.5mn + 0.125mn = 4.95mn$.

The pseudo code of the March-SASP algorithm is illustrated in Figure 6.

To be precise, the cells in addressing paths $\searrow$ and $\searrow$ are subsets of the cells in addressing paths $\searrow$ and $\searrow$, respectively. In the pseudo code in Figure 6, $\searrow$ and $\searrow$ are defined as addressing path C and addressing

```
Define addressing path C: //
i = 3;j = –1;
while ( i <=2*max(n,m) )
{ if ( (i+j) % 8 ==6)
   { i =i+2; j =j–2; }
else                                  // (i+j) % 8 ==2
   { i =i–2; j =j+2;}
if (i ==–1)                           // out of RoDs
   { i =1; j =j+2;}
if (j ==–1)                           // out of RoDs
   {j =1;i =i+2;}
if((i > (n–1))|| (j > (m–1)))  // out of RoDs
   {continue;}
}
```

```
Define addressing path D: //
i = 5; j =–2;
while( i <=2*max(n,m) )
{ if ((i+j) % 8 ==7)
   { i =i+2; j =j–2; }
else                                  // (i+j) % 8 ==3
   { i =i–2; j =j+2;}
if (i ==–1)                           // out of RoDs
   { i =1; j =j +2;}
if (j ==–2)                           // out of RoDs
   {j=0; i =i+2;}
if((i >(n–1))||(j >( m–1)))  // out of RoDs
   { continue; }
}
```

```
March-SASP Algorithm:
{Step 1:
   for ( i=0; i<n; i++)
      for( j=0; j<m; j++)
      { write 1 to C_{i,j}; read C_{i,j};}
 Step 2:
   for ( i=0; i<n; i++)
      for( j=0; j<m; j++)
      { read C_{i,j};}
 Step 3: for addressing path B:
   { write 0 to C_{i,j}; read C_{i,j};}
 Step 4: for addressing path A:
   { write 0 to C_{i,j}; read C_{i,j};}
Step 5: for addressing path D:
   { read C_{i,j};}
Step 6: for addressing path B:
   { write 0 to C_{i,j};}
Step 7: for addressing path C:
   { read C_{i,j};}
}
```

Figure 6 Pseudo code of the March-SASP algorithm.

path D, respectively. Similar to ↘ and ↖, the direction of the quasi-diagonal lines in addressing paths ↘ and ↘ are determined by the result of $(i + j)$ mod 8. For ↘, if the result of $(i + j)$ mod 8 is 6, the corresponding quasi-diagonal line ramps down, and if the result is 2, it ramps up; otherwise, the corresponding cells are not in the addressing path. For ↘, if the result of $(i + j)$ mod 8 is 7 or 3, the corresponding quasi-diagonal line ramps down or up respectively; otherwise, the corresponding cells are not in the addressing path.

The fault coverage capability of the proposed March-SASP algorithm is analyzed as follows:

SR fault: The SR fault in every cell is excited by the w1 operation in Step 1, and it is tested by the r1 operation in Step 2 with the sneak path addressing mode.

SS fault: The SS faults in the white and black cells are excited and tested by the w0 and r0 operations in Step 3 and Step 4, respectively, with the snake addressing mode.

IPF0: The w0 operations in Step 3 and Step 4 excite the IPF0s for the white and black cells, respectively. The IPF0s for the white cells are tested by the r0 operation in Step 3 and r0 in Step 5. The IPF0s for the black cells are tested by the r0 operation in Step 4 and r0 in Step 7.

WTF0: A WTF0 is covered by the algorithm because its test condition is the same as that of an SS fault.

WDF1: These faults are excited by the w1 operation in Step 1 and are then tested by r1 in Step 2.

WWDF1: WWDF1s are excited and tested by the w1 and $\mathrm{r1_m}$ operations in Step 1, respectively.

PDF: All white cells are initialized to the RESET state in Step 3, and their PDFs are excited in Step 4 by writing '0' to their four neighbors (black cells). Then, the PDFs of the white cells are tested by the r0 operation in Step 5 with the March-SASP scheme. Similarly, the black cells are initialized to the RESET state in Step 4, their four neighboring cells (white cells) are written to '0' in Step 6, and the PDFs are tested by the r0 operation in Step 7.

RRD fault: The RRD faults of white cells are excited by w0 operations in Step 3. Then, the faults are tested by the r0 operation following the w0 operation in Step 3. Similarly, for the black cells, the w0 operation in Step 4 excites the RRD faults, and the following r0 operation in Step 4 tests the RRD fault.

RD fault: White and black cells are initialized to state '0' by the operations of w0 in Step 3 and Step 4, respectively. The faults for the white cells are excited and tested by the r0 operation in Step 3 and r0 in Step 5, while the faults for the black cells are excited and tested by the r0 operation in Step 4 and r0 in Step 7.

FWR fault: The w1 operation in Step 1 initializes all cells to the SET state. Then, the FWR faults are excited by $\mathrm{r1_m}$ in Step 1 and tested by the r1 operation in Step 2 with the sneak path addressing mode.

Table 4 Fault coverage capability of the March-SASP algorithm

Fault	Testing algorithm			Fault	Testing algorithm		
	Initial state	Excitation	Test		Initial state	Excitation	Test
SR	–	M1:w1	M2:r1	WWDF1	–	M1:w1	M1:r1$_{\mathrm{m}}$
SS	–	M3:w0 M4:w0	M3:r0 M4:r0	PDF	M3:w0 M4:w0	M4:w0 M6:w0	M5:r0 M7:r0
IPF0	–	M3:w0 M4:w0	M3:r0;M5:r0 M4:r0;M7:r0	RRD	–	M3:w0 M4:w0	M3:r0 M4:r0
WTF0	–	M3:w0 M4:w0	M3:r0 M4:r0	RD	M3:w0 M4:w0	M3:r0 M4:r0	M5:r0 M7:r0
WDF1	–	M1:w1	M2:r1	FWR	M1:w1	M1:r1$_{\mathrm{m}}$	M2:r1

Table 5 Comparison between the proposed sneak path test algorithm and those in [4], [11] and [15]

Algorithm	SAF	IPF0	WTF0	WDF1	WWDF1	PDF	RRD	RD	FWR	Complexity
March-PC [11]	Y	Y	Y	Y	Y	Y_2	Y	Y	Y	$11mn$
March-PCM [4]	Y	Y	Y	Y	Y	Y_2	Y	Y	Y	$8mn$
March-SA	Y	Y	Y	Y	Y	Y_4	Y	Y	Y	$7mn$
Sneak-path [15]	Y	N	Y	Y	Y	Y_2	N	N	N	$3.8mn$
March-SASP	Y	Y	Y	Y	Y	Y_4	Y	Y	Y	$4.95mn$

The analysis of the fault coverage capability of the March-SASP test algorithm is summarized in Table 4.

3.2.3 *Comparison*

Table 5 compares the March-SASP algorithm with other test algorithms of PCM based on fault coverage and complexity. Notice from this table that although the complexity of the March-SASP algorithm is slightly higher than that of the original sneak path algorithm ($4.95mn$ vs. $3.80mn$), this algorithm extends the fault coverage capability to all of the fault types in Table 1. Compared to other March-like test algorithms such as [4] and [11], which have the same fault coverage capability as March-SASP, March-SASP has a much lower complexity. Furthermore, the March-SASP algorithm is able to excite the PDF more effectively since all of the neighboring cells are programmed to the RESET state before the victim cell is read.

4 Conclusion

We proposed a new snake addressing scheme, which accesses the memory array in a diagonal path, to be applied to PCM testing. This scheme effectively excites PDFs because it utilizes access to the four neighboring cells to heat up the victim cell, and thus accumulating disturb effects of the thermal crosstalk from the four neighboring cells. A March test algorithm with snake addressing, namely, the March-SA algorithm, was designed to cover all of the traditional PCM faults and parasitic faults discovered in recent years. It has a complexity of $7mn$, which is 12.5% less than the March-PCM test algorithm. Furthermore, when it was incorporated with the sneak path test method, it extended the fault coverage capability of an RD fault, read recovery fault, IPF0, and FWR fault, which the original sneak path test method failed to cover. By incorporating the sneak path test method, the test complexity increased by approximately 30%. However, when compared to March-SA algorithms, its test complexity decreased by 29.3%.

Acknowledgements This work was supported by National Basic Research Program of China (973) (Grant Nos. 2015CB057201, 2013CBA01903) and R&D project of the Shenzhen Government, China (Grant Nos. JCYJ20140417144423194, JCYJ20140417144423198).

Conflict of interest The authors declare that they have no conflict of interest.

References

1 Pirovano A, Lacaita A L, Benvenuti A, et al. Electronics switching in phase-change memories. IEEE Trans Electron Dev, 2004, 51: 452–459

2 Kang S, Cho W Y, Cho B H, et al. A 0.1-μm 1.8V 256-Mb phase-change random access memory (PRAM) with 66-MHz synchronous burst-read operation. IEEE J Solid-State Circ, 2007, 42: 210–218

3 Prall K, Ramaswanmy N, Kinney W, et al. An update on emerging memory: progress to 2Xnm. In: Proceedings of IEEE International Memory Workshop, Milan, 2012. 1–5

4 Mohammad M G. Fault model and test procedure for phase change memory. IET Comput Digital Tech, 2011, 5: 263–270

5 Pan X J, Cui X L, Zha J, et al. Modeling and test for parasitic resistance and capacitance defects in PCM. In: Proceedings of 12th Annual Non-Volatile Memory Technology Symposium (NVMTS), Singapore, 2012. 73–76

6 Zhang X, Wei Y, Lin X, et al. Critical parasitic capacitance in nano-scale phase-change memory cell. In: Proceedings of IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), Chengdu, 2014. 18–20

7 Ielmini D, Lacaita A L, Mantegazza D. Recovery and drift dynamics of resistance and threshold voltages in phase-change memories. IEEE Trans Electron Dev, 2007, 54: 308–315

8 Osada K, Kawahara T, Takemura R, et al. Phase change RAM operated with 1.5V CMOS as low cost embedded memory. In: Proceedings of IEEE International Conference on Custom Integrated Circuits, San Jose, 2005. 431–434

9 Pirovano A, Redaelli A, Pellizzer F, et al. Reliability study of phase-change nonvolatile memories. IEEE Trans Dev Mater Reliab, 2004, 4: 422–427

10 Maimon J D, Hunt K K, Burcin L, et al. Chalcogenide memory array: characterization and radiation effects. IEEE Trans Nucl Sci, 2003, 50: 1878–1884

11 Mohammad M G, Terkawi L, Albasman M. Phase change memory faults. In: Proceedings of 19th International Conference on VLSI Design. Held jointly with 5th International Conference on Embedded Systems and Design, Hyderabad, 2006. 6–6

12 Wong H S P, Raoux S, Kim S B, et al. Phase change memory. Proc IEEE, 2010, 98: 2201–2227

13 Lacaita A L. Phase change memories: state-of-the-art, challenges and perspectives. Solid-State Electron, 2006, 50: 24–31

14 Zhang Z, Xiao W, Park N, et al. Memory module-level testing and error behaviors for phase change memory. In: Proceedings of IEEE 30th International Conference on Computer Design (ICCD), Montreal, 2012. 358–363

15 Kannan S, Rajendran J, Karri R, et al. Sneak path testing of crossbar-based non-volatile random access memories. IEEE Trans Nanotechnol, 2013, 12: 413–426

16 Wei Y, Lin X, Jia Y, et al. A SPICE model for phase-change memory (PCM) cell based on analytical conductivity model. J Semiconduct, 2012, 33: 1–5

17 van de Goor A J, Al-Ars Z. Functional memory faults: a formal notation and a taxonomy. In: Proceedings of IEEE 18th International VLSI Test Symposium (VTS), Montreal, 2000. 281–289