Simulation and Development of Short Transparent Tests for RAM

S. Demidenko*©, A. van de Goor**, S. Henderson*, P. Knoppers**
*Institute of Information Sciences and Technology, Massey University
Riddett Complex, Palmerston North, Private Bag 11222, New Zealand
** Department of Electrical Engineering, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Abstract

Short transparent memory test algorithms for semiconductor memories are presented along with an evaluation of the space on silicon required for their built-in self-test implementation. A modified version of the memory test simulation package MAP+ has been developed and employed for test algorithms generation and simulation.

1. Introduction

The majority of the complex VLSI/ULSI/WSI circuits and systems contain built-in *Random -Access Memories* (RAM) with significant capacities [1-5]. Such memories are among the most important parts of the complex *Integrated Circuits* (IC). Besides, semiconductor memories are the most wide spread type of the completed ICs produced by the industry. This leads to the increasing importance of fault-free operation of the memory components. In achieving it, comprehensive and reliable electronic testing (first of all, the functional testing [6,7]) has to be performed.

An important issue in RAM testing is the ability to provide it on-line or quasi on-line (without interruption) or with minimal interruption of normal operation of the whole system). This implies that the contents of memory-under-test should not be affected as a result of the testing.

The traditional approach to satisfy this requirement consists of saving the contents of the entire memory-under-test in some intermediate data storage prior to the testing. After testing is completed, the data are transferred back to the memory. However, such an approach not always can be implemented to test memories of an embedded type or the memories where high availability is imperative. Spare data storage of the same capacity as the main memory is often not available. Besides, a time delay due to the save-restore procedure can be too long to be acceptable. Moreover, testing of embedded memories is further complicated by the limited access to it from outside of the system.

An efficient way to solve the problem is the use of transparent tests [8-12], which preserve the initial contents of the memory.

Among the different types of the algorithms used in RAM testing, *March* algorithms have proved to be among the most time/cost effective [6,7]. Design and evaluation of Transparent March algorithms aimed at the *Built-in Self-Test* (BIST) implementation for some specific types of faults at bit-oriented memories are discussed in this paper along with the modified version of the memory simulation package.

2. Symbols and notation

The following symbols and notation are used:

0(1)	denotes that a cell is in a logic state zero (one)
a	denotes that a cell is in logic state *a*; i.e., either *0* or *1*, $a \in \{0,1\}$
$\uparrow(\downarrow)$	denotes write *1(0)* operation to a cell containing *0(1)*
$\updownarrow$	denotes write $\overline{a}$ operation to a cell containing *a*
w_a	denotes write *a* ($a \in \{0,1\}$) operation to a memory cell
r_a	denotes read operation from a memory cell; where *a* ($a \in \{0,1\}$) is expected
$\vee(\wedge)$	denotes a *lower (higher)* address of one cell in comparison with another
$\Uparrow$	denotes the addressing order *up*
$\Downarrow$	denotes the addressing order *down*
$\Updownarrow$	denotes the addressing order *don't care*
A(B)	denotes a March test phase, where *A* denotes an addressing order ($A \in \{\Uparrow, \Downarrow, \Updownarrow\}$), while *B* denotes the set of operation applied to the cell ($B \in \{r_a, w_a, r_{\overline{a}}, w_{\overline{a}}\}$).

1081-7735/01 $10.00 © 2001 IEEE

3. Fault models

The following fault models [6,13] are used in this paper.

Address decoder faults (AF). Four different subtypes of AF can be distinguished. Each of them can not occur alone but only in combination with at least one other subtype. They are:

AF(1) – no cells can be accessed with some address;
AF(2) – there is no address with which a particular cell can be accessed;
AF(3) – multiple cells are accessed simultaneously with some address;
AF(4) – a certain cell can be accessed with multiple addresses.

Stuck-at fault (SAF): the logic value of a stuck-at cell is always *0* (SAF0) or always *1* (SAF1).

Transition fault (TF): a cell fails to undergo *0->1* (TF$\uparrow$), or *1->0* (TF$\downarrow$) transition.

Coupling fault (CF) involves two (or more) cells: a transition write operation to one cell (cell *j*) changes the contents of another cell (cell *i*). Cell *j* is said to be the *coupling cell*, whereas cell *i* is said to be the *coupled cell*. A CF that involves two cells is called a *2-coupling fault (2-CF)*. Such a fault is a special case of the more general *k-coupling fault (k-CF)*. The *k*-CF fault is related to the same two cells as the *2*-CF while, in addition, the fault only takes place when another *k-2* cells are in a certain state. Several types of CFs can be distinguished.

Inversion coupling fault (CFin) is defined as follows: an $\uparrow$ or a $\downarrow$ transition in the coupling cell causes an inversion in the coupled cell. There are two CFin subtypes: $<\uparrow;\updownarrow>$ and $<\downarrow;\updownarrow>$.

Idempotent coupling fault (CFid) is caused by an $\uparrow$ or a $\downarrow$ write operation in the coupling cell forcing a certain value (*0* or *1*) in the coupled cell. There are four CFid subtypes: $<\uparrow; 0>$, $<\downarrow; 0>$, $<\uparrow; 1>$, $<\downarrow; 1>$. The coupled cell can have a *lower address* than the coupling cell (it is indicated as $\vee$), or of a *higher address* ($\wedge$). Consequently, the complete notation for the coupling fault is as follows: $\Phi <S;F>$, where Φ - coupling direction ($\wedge,\vee$), S - the fault sensitisation operation ($\uparrow,\downarrow$), and F - the value which is set into the coupled cell ($0,1,\updownarrow$).

Read Fault (RF): is caused by reading the contents of a particular cell. Two *RF* fault models are discussed here: *Read Disturb Fault (RDF)* and *Deceptive Read Disturb Fault (DRDF)* [14,15].

Read Disturb Fault (RDF): the contents of a memory cell is inverted during a read operation; i.e., r_a is resulted in $r_{\overline{a}}$, and the contents of the cell is changed to $\overline{a}$. There are two subtypes of this fault: when an initial state of the memory cell was 0 and became 1 after the read operation (RDF$\uparrow$), and when the initial state was 1 and became 0 as a result of reading (RDF$\downarrow$).

Deceptive Read Disturb Fault (DRDF): the contents of the memory cell is inverted as a *result* of read operation, i.e., r_a is resulted in r_a, and the contents of the cell is changed to $\overline{a}$. As the faulty transition happens *after* the reading, it is not detected by the initial r_a operation, as it produces the correct result *a*. Once again, there are two fault model subtypes there: DRDF$\uparrow$ and RDF$\downarrow$.

4. Transparent March tests

Traditionally [6,7], March tests are characterised as follows. They consist of a sequence of *March Phases* (or *March Elements*). Each phase is a set of operations that are applied to a memory cell. After the phase application is completed for a cell, the algorithm proceeds to the next cell where the procedure is repeated, and so on.

The way the process moves from one cell to the next is determined by the address order which can be increasing or decreasing. The operations applied to a cell are w_0, w_1, r_0, and r_1.

March tests are based on functional fault models [6,7]. For this reason they provide generally the best *fault coverage/test time* ratio among memory test algorithms. For example, the MATS++ test [6] includes just three phases: M_0, M_1 and M_2. The first phase is memory initialisation (writing a background), while the other two phases are sets of read and write operations aimed at detecting faults of the targeted types.

$$\{\underset{M_0}{\Updownarrow(w_0)};\ \underset{M_1}{\Uparrow(r_0, w_1)};\ \underset{M_2}{\Downarrow(r_1, w_0, r_0)}\}$$

The test is only of 6N complexity. At the same time it detects all AFs, SAFs and unlinked TFs, as well as some CFs.

Transparent March algorithms are a very attractive solution to testing when preserving the memory-under-test's contents is imperative. The condition is that at the end of the test session data in the memory should be same as it was prior to testing. It has been shown in the literature [9-11] that it is possible to convert any traditional March test into its transparent version. In general, the conversion procedure includes the following steps:

- Remove the initialisation phase (writing background) from the test
- Change all r_0 (r_1) operations to the r_a ($r_{\overline{a}}$) operations
- Change all w_0 (w_1) operations to the w_a ($w_{\overline{a}}$) operations.

It is important to mention that the conversion into the transparent type does not lead to any decrease in fault coverage of the algorithm.

An example below presents the result of the MATS++ test algorithm converted into its transparent

version. Removing initialisation phase, and substituting initial read and write operations with new ones results in the following two phase transparent March test:

$$\{\Uparrow(r_a, w_{\bar{a}}); \Downarrow(r_{\bar{a}}, w_a, r_a)\}$$

M_0 M_1

Transparent tests are particularly suitable for Built-In Self-Test (BIST) implementation. BIST can be employed for both manufacturing and in-service testing. In the latter case, the testing can be done periodically between the cycles of memory system operations, while a small area of the memory (e.g., 3x3 or 4x4 bit – sliding window) is tested at every particular cycle. This would also reduce the hardware overhead required to calculate and to store the data related to the fault-free and actual operation of the memory, as well as to compare the data after the testing cycle is completed.

5. Minimal complexity transparent tests with Read Fault Coverage

Periodic transparent testing allows utilisation of simple March algorithms for testing RAMs where the contents are randomly changed during system operation. For example, the minimal complexity March 3N test [12] $\{\Uparrow(r_a, w_{\bar{a}}); \Updownarrow(r_{\bar{a}})\}$ provides asymptotically maximal (i.e., 100%) fault coverage with respect to AFs, SAFs, TFs, and covers the majority of CFs and some PSFs when the memory contents are randomly changing, and when the number of the test executions is increasing unlimitedly. The test consists of two phases: the stuck-at faults and address decoder faults are sensitised in the first phase, while they are detected in the second phase of the test. The same is true for the transition faults (if the faulty cell has an initial state *0* (*1*) for TF↑ (TF↓).

A single execution of the March 3N test is sufficient to detect all stuck-at faults and address decoder faults, while double execution of the algorithm detects transition faults. The coupling fault coverage is increased with the number of the test runs for different memory contents (backgrounds). It can be shown [8] that the ratio between the number of coupling faults detected by the single execution of the March 3N algorithm and the total number of possible coupling faults is $T_{x_1}/T = 1/3$. Total fault coverage for an arbitrary number n of the test executions is $T_{x_n}/T \approx (4(1-0.75^n)+8(1-0.5^n))/12$.

Obviously, the effectiveness of any RAM test greatly depends on the type of errors occurring in the memory. Traditionally, transparent tests have been developed assuming that read operations do not cause errors into the memory-under-test. Unfortunately, this assumption is not true in many cases of real memories [14]. For this reason an additional type of fault models (*Read Faults*) has been included into consideration.

March 3N test can be modified to increase its efficiency in terms of read fault coverage. Faults of the *RDF* subtype can be detected by performing a read operation immediately after a write operation while comparing the written-in and read-out data. Besides this, a read operation immediately after a write operation would also cover errors related to the SAF fault type. The read operation $r_{\bar{a}}$ throughout the memory is the last phase of the March 3N algorithm. For a given cell this operation does not happen immediately in time after the write operation (it starts only after all the memory cells have completed the first phase). However, it is still a correct detection step as there are not any other manipulations of any memory cell contents between the write and read. Thus it can be seen that RDFs are covered by the March 3N algorithm.

In order to detect faults of the DRDF subtype, two sequential read operations immediately following a write operation are required (this is due to the fact that cell contents is 'flipped' *because of* and *after* the read operation). If the error occurs, the results of the two read operations will differ, while the value of the first read operation is equal to that first written to the cell.

The corresponding modification of March 3N results in a new algorithm of 4N complexity – let us call it March R: $\{\Uparrow(r_a, w_{\bar{a}}); \Updownarrow(r_{\bar{a}}, r_{\bar{a}})\}$. The proposed algorithm provides the same fault coverage as March 3N. In addition it covers half of RDFs (either RDF↓ or RDF↑) and DRDFs (either DRDF↓ or DRDF↑). The algorithm can be further enhanced to cover all RDFs/DRDFs by adding one more read operation (5N complexity) - March RR: $\{\Uparrow(r_a, r_a, w_{\bar{a}}); \Updownarrow(r_{\bar{a}}, r_{\bar{a}})\}$.

It can be seen that the above March 3N algorithm and its modifications invert the memory contents. This is not acceptable in many cases, because it would require an even number of passes of the test, causing possibly too much delay for the application. To solve the problem it can be recommended to incorporate into the memory special hardware that provides additional inversion during reading and writing [17].

Alternatively, the following basic transparent March algorithm of complexity 4N can be employed: $\{\Uparrow(r_a, w_{\bar{a}}, w_a); \Updownarrow(r_a)\}$. This algorithm provides substantially higher coverage than March 3N for the same number of the test runs, in particular in terms of coupling faults. All address decoder and inversion coupling faults are detected by a single test execution. Besides, it can be shown that a single execution detects half of the idempotent CFs as well.

The ratio between the number of the CFs detected by a single execution of the test and the overall number of coupling faults is $T_{x_1}/T = 8/12 = 2/3$, while for n runs it is $T_{x_n}/T \approx (4+8(1-0.5^n))/12$.

There is one drawback of the discussed March 4N test compared with the March 3N. March 4N does not detect all SAFs and some of TFs. For a case where such faults are expected, another basic transparent test algorithm (March 5N) can be employed: $\{\Uparrow(r_a, w_{\bar{a}}, r_{\bar{a}}, w_a); \Updownarrow(r_a)\}$. This test has all advantages of March 4N, while in addition it can detect all SAFs and TFs during its first phase.

The presented March algorithms 4N and 5N can be enhanced to cover RDFs and DRDFs in the same way as it has been presented above for March 3N transparent test.

6. BIST generator implementation

In order to evaluate the feasibility of implementing transparent March testing in a BIST environment, an automatic test pattern generator for the transparent March 5N algorithm was designed for a static RAM fragment (sliding window) of 4x4-bit size. Atmel-ES2 CMOS 0.7μm 2-metal 1-poly technology was used. The assessment of the feasibility of implementing this design comes about by calculating the percentage of silicon area the generator occupies in relation to the memory that it is designed to test. Fig. 1 shows comparison between silicon areas occupied by March 5N BIST generator and the matrix of memory cells.

However the above data have to be adjusted by taking into account an array efficiency parameter. Array efficiency is the percentage that a memory array occupies on an entire silicon space of a memory chip [17]. The parameter has different values depending of memory capacity, implementation technology, and architecture. Fig. 2 presents adjusted data where typical values [17] of the array efficiency were used.

It can be seen that for a small-capacity memory, the test circuitry occupies a substantial part of the silicon space. At the same time the portion is decreasing as the memory size increases. This supports the fact that BIST is feasible for larger memories. It must be stated that the above evaluation did not include the area that is required for the BIST analyser circuitry. It is expected this circuitry will consume approximately the same silicon area as the test generator for 4x4 bit memory (depending on the type of the analyser implemented, i.e., bit-by-bit comparison, signature compression, or other), thus giving the following results (Table 1).

Memory Capacity	Space on Silicon	
1K bit	42	%
32K bit	0.6	%
1M bit	0.02	%
1G bit	0.00003	%

Table 1. BIST circuitry area for 5N transparent March testing (% of a chip silicon area)

It is rather natural and obvious that the feasibility of implementing BIST increases with the size of the memory. This is due to the fact that the area of silicon required for BIST circuitry increases logarithmically whereas the area required for the memory itself increases linearly. In fact, the IC DRAM size considered by industry for one of the first reported BIST implementations was 256M bit [18].

7. Simulation package MAP+

Originally developed at the Delft University of Technology, the Memory Animation Package MAP [6] has been employed for a number of years as a simulation tool for the evaluation of new and known test algorithms in presence of different faults (such as stuck-at, transition, coupling, address decoder, neighbourhood pattern sensitive, etc.). Unfortunately, the package does not support the RDF and DRDF fault models. In addition, the simulator is not capable of working with transparent tests (it does not have background initialisation for regular or random patterns, and does not support multiple executions of a single test for different backgrounds).

Addressing the necessity of obtaining and making it capable of evaluating new transparent tests, a modified version of the simulation system - MAP+ has been developed. The modified package provides the same user environment as its parent version (Fig.3). At the same time, it offers several new enhancement options for memory testing simulation (Fig. 4). In addition to all options of the 'old' system, it supports the following new functions:

- handling faults of read type;
- memory initialisation, i.e., writing specified initial backgrounds (deterministic or random) manually or automatically;
- automatic new background initialisation and execution of a transparent March algorithm, while recording failed cells after each run;
- limited option of new March test generation for given test faults.

The new menu for the Test Memory option offers additional items on writing a user-defined initial background into the memory-under-test prior to the test execution. It can be performed either automatically in between test cycles, or manually. The manual memory initialisation function has the structure presented in Fig. 5.

The initialisation is performed incrementally, one cell at time. This function is useful for the detection of some particular fault configurations. However such an approach can be impractical for any large size memories, or in cases where the same test is performed over a number of different backgrounds. Counter-based and Random Initialisation options of MAP+ allow for automatic background writing (Fig. 6). A special function *write_def_background* is employed. It can be called by the *write_and_test* function that implements the user option of writing all possible backgrounds to a memory while repeatedly performing the

same test (or a number of tests) for each background. The background writing function uses right-left conversion technique for integer to binary conversion [17].

Another improvement that has been introduced into MAP+ is a new function of automatic March test generation for given type of faults. At this stage it has limited fault type coverage and is based on combining the test phases that initialise and detect corresponding faults [6] (Table 2).

Fault	Initialisation and detection procedure
SAF	From each cell, a 0 and a 1 must be read
TF	Each cell must undergo a ↑ transition (state of cell goes from 0 to 1), and a ↓ transition (state of cell goes from 1 to 0), and be read after each transition before undergoing any further transitions.
CFin	For all cells which are coupled cells, each cell must be read after a series of possible CFins may have occurred (by writing into the coupling cells), with the condition that the number of transitions in the coupled cell is odd (i.e. the CFins do not mask each other).
CFid	For all cells which are coupled cells, each cell should be read after a series of possible CFids may have occurred (by writing into the coupling cells), in such a way that the sensitised CFids do not mask each other
NPSF	Active NPSF: Each base cells must be read in state 0 and in state 1, for all possible changes in the deleted neighbourhood pattern. Passive NPSF: Each base cell must be written and read in state 0 and in state 1, for all permutations of the deleted neighbourhood pattern. Static NPSF: Each base cell must be read in state 0 and in state1, for all permutations of the deleted neighbourhood pattern.
RF	RDF: Each cell must be read from immediately following a write operation. DRDF: Each cell must be read from twice in succession following a write operation.

Table 2. Fault diagnostic phases

It is planned to use some additional techniques to achieve maximal fault types coverage and minimisation of the generated algorithm thus making the package more efficient. For example, a state-machine based approach[19] and the approach based on the use of Markov chain modelling in algorithm generation [20] are considered for implementation. Several other modifications are going to be incorporated into the package MAP+ (one of them is the sliding window framework, proposed by one of the authors - S.D.© together with his graduate student N. Lord [21]). The package will be demonstrated to the time of the symposium.

8. Conclusion

The development and evaluation of transparent March memory tests enhanced to cover read faults were discussed in this paper. The tests are of minimal complexity while their fault coverage is asymptotically rising along with the number of test executions over different backgrounds. The algorithms are particularly suitable for application in an on-line BIST environment for memories of sufficiently large capacities. This has been proved by the experimental results on design of BIST test generator for microelectronics, implementation in CMOS technology. At the same time minimal complexity transparent March algorithms also can be used in traditional off-line external memory testing.

In order to support design and evaluation of new March algorithms of the transparent type, an enhanced memory test simulation system MAP+ has been developed. We believe the system is the only one of its type supporting transparent testing among the software simulators available on the market. The system allows performing arbitrary background initialisation, provides multiple test execution for different backgrounds, offers a fail data collection option, and supports some new fault models.

References

1. L. Geppert and T. Perry, Transmeta's Magic Show, IEEE Spectrum, May 2000, p.26-33.
2. http://developer.intel.com/design/product.htm
3. S. Gary et al., PowerPC 603™: A Microprocessor for Portable Computers, IEEE Design & Test of Computers, no 4, 1997, pp. 16-17.
4. Bhavsar and J. Edmondson, Alpha 21164 Testability Strategy, IEEE Design & Test of Computers, No 4, 1997, pp. 25-33.
5. Y. Zorian et al., Testing Embedded-Core-Based System Chips, IEEE Computer, Vol. 32, no 6, 1999, pp. 52-60.
6. Ad. van de Goor, Testing Semiconductor Memories, Theory and Practice, CamTex Publishing, Gouda, The Netherlands, 640 p.
7. B. Cockburn, Tutorial on Semiconductor Memory Testing, JETTA, vol. 5, 1994, pp. 321-336.
8. S. Demidenko, et al, Transparent Tests for Semiconductor Memory, 7th International Symposium on IC technology, Systems and Applications, Singapore, 1997, pp. 192-195.
9. M. Nikolaidis, Transparent BIST for RAMs, ITC, Baltimore, 1992, pp. 598-607.
10. V. Yarmolik and M. Karpovski, Transparent Memory Testing for Pattern-Sensitive Faults, ITC, Washington, 1994, pp. 860-869.
11. M. Karpovski and V. Yarmolik, Transparent Memory BIST, IEEE International Workshop MTDT, 1994, pp. 106-111.
12. S. Demidenko, et al. March 3N and March 4N Memory Tests, IES Journal on Electronics, Vol. 39, No1, 1999, 9p.
13. Ad. van de Goor and Y. Zorian, Effective March Algorithms for Testing Single-Order Addressed Memories, JETTA, vol. 5, 1994, pp. 337-345.
14. Ad. van de Goor, Private correspondence with DRAM manufacturing companies
15. S. Hamdioui and Ad. van de Goor, March Tests for Word-Oriented Two-Port Memories, 8th Asian Test Symposium, 1999, pp. 53-60.
16. B. Prince, Semiconductor Memories. A Handbook of Design, Manufacture, and Application. Wiley, 1996.

17. S. Henderson, Transparent Built-In Self Testing of Semiconductor Memories, Massey University, College of Sciences, Final Year Project (S. Demidenko – supervisor), 2000, 64 p.
18. F. Hii, T. Powell, D. Cline, A Built In Self test Scheme for 256Meg SDRAM, IEEE International Workshop on Memory Technology, Design and Testing, Singapore, August 13-14, 1996, pp. 15-21.
19. B. Smith, Automatic Verification and Generation of March Tests, Delft University of Technology, The Netherlands, Department of Electrical Engineering, Section Computer Architecture & Digital Systems, MSc Thesis, 2000, 77 p.
20. D. Niggemeyer and E. Rudnick, Automatic Generation of Diagnostic March Tests, VTS'2001, USA, 6p.
21. N. Lord, Window-Based Simulation System for Memory Testing, Massey University, College of Sciences, Final Year Project (S. Demidenko – supervisor) - to be completed in November 2001

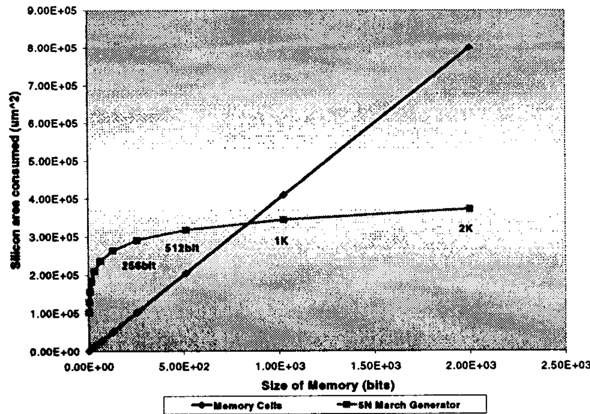

Fig.1. March 5N generator Vs memory cells area.

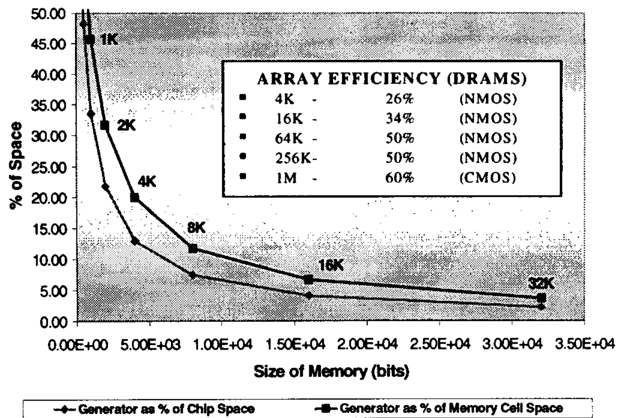

Fig.2. Test generator Vs DRAM chip space

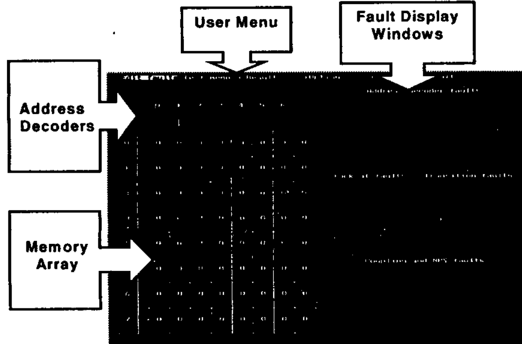

Fig.3. MAP+ user environment

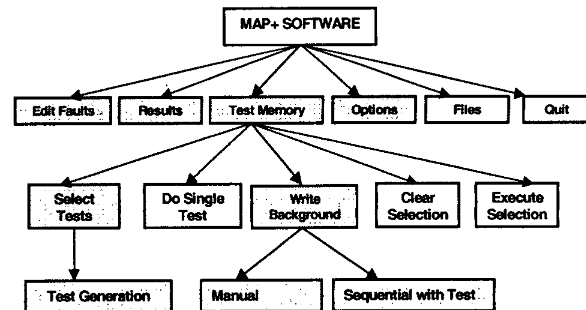

Fig.4. Menu structure of MAP+

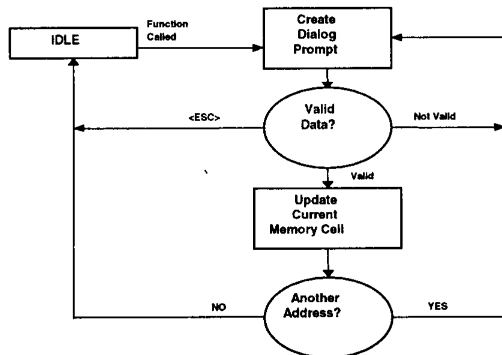

Fig.5. Manual initialisation function

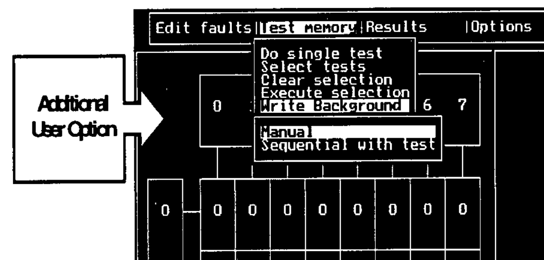

Fig.6. User-defined pre-test background