ELSEVIER

INTEGRATION, the VLSI journal 26 (1998) 117–140

INTEGRATION
the VLSI journal

FTROM: A Silicon Compiler for Fault-tolerant ROMs[1]

Anurag Gupta[a,*], Kanad Chakraborty[b], Pinaki Mazumder[c]

[a] *Intel Corporation, 2200 Mission College Blvd., Santa Clara, CA 95052–8119, USA*
[b] *IBM Microelectronics, E. Fishkill, 1580 Route 52, Hopewell Junction, NY 12533–6531, USA*
[c] *Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109–2122, USA*

Abstract

This paper describes a new CAD tool, **FTROM** – **F**ault-**T**olerant **ROM** compiler, which synthesizes layout geometries of fault-tolerant ROM modules with flexible, user-specified geometry and CMOS design-rule parameters. This physical design tool produces high-quality built-in self-testable (BIST) and fault-tolerant ROM layouts and uses a novel, minimum-delay overhead approach for fault-tolerance. A tool like FTROMeliminates the high cost of external testing of embedded ROM macros with I/O pins that are difficult to control and observe.

Keywords: Built-in self-test (BIST); Built-in self-repair (BISR); Fault-tolerance; Read only memories; Silicon compilers

1. Introduction

In the eighties and nineties, VLSI physical design and testing have met new challenges and witnessed remarkable growth. As the chip size has grown phenomenally to more than a million-transistors per chip, the complexity of physical design has increased, and more and more sub-circuits have become inaccessible for testing due to the diminishing pin-to-device count. Built-in self-test (BIST) was introduced for various structured logic and memory arrays, to allow comprehensive testing for functional, electrical and parametric faults. BIST circuits ensure that bad chips will be automatically spotted and eliminated on the production line, thus ensuring quality control. As effective channel lengths of transistors shrink below 0.2 μ, even minor process variations

* Corrresponding author.

[1] Extended version of a paper presented at the IEEE Symposium on Defect and Fault Tolerance in VLSI Systems (Austin, Nov. 1998). This research was partially supported by the Army Research Office under its MURI programs at the University of Michigan and also by the National Science Foundation.

0167-9260/98/$ – see front matter
PII: S0167-9260(98)00024-8

across the surface of a silicon wafer may cause layout defects. Even though such defects are not always associated with immediate functional failure, they may lower the device reliability during the operational lifetime of the chip. To improve both the manufacturing yield and reliability of VLSI chips, BIST needs to be used in conjunction with fault-tolerant design techniques such as error-control coding (ECC). Furthermore, BIST and on-line error correction are of paramount importance in designing embedded ROM macros used as microcode lookup tables in high-performance microprocessors, ASICs used in outer space, and in signal-processing applications. In embedded and mission-critical (space and other) applications, external field testing and error-correction are prohibitively expensive or infeasible.

Silicon compilation of CMOS layouts is a useful technique for reducing the design cycle time and thereby increasing the CAD designer's efficiency and lowering the time to market. However, silicon compilation has so far been used almost exclusively for structured custom design; in other words, for layouts that can be easily generated by replicating a basic unit; for example, an iterative logic array (ILA) [1], a read-only memory (ROM) array [2,3]. In this work, a novel silicon compilation technique that combines structured custom design with custom place-and-route for efficiently interconnecting the sub-modules of the global fault-tolerant memory macromodule (or macrocell) has been proposed for BIST/BISR[2] ROM design. A similar approach for BISR RAM design has achieved excellent results [4]. The proposed technique has been found to generate very high-quality layouts (almost similar to full-custom ones) with modest area overheads for BIST and error-correction. Furthermore, it employs an innovative error-correction circuit technique that causes the correct contents to be retrieved with a negligible access time penalty on the ROM.

The paper is organized as follows: in Sections 2 and 3, we introduce the concept of BIST and its application to ROMs; next, we describe the silicon compiler environment (Section 4); fault-modeling, self-testing and self-repair schemes used in **FTROM** (Sections 5 and 6); circuit implementation (Section 7); **FTROM** user interface (Section 8) and the layout generation process (Section 9). The sample layouts generated by **FTROM** and an analysis of the performance metrics (area, delay and yield) are presented in Section 10.

2. Introduction to BIST concepts

A number of algorithmic methods for test generation and techniques for enabling testing (commonly known as *design for testability* (DFT)) have been proposed for combinational and sequential circuits. These methods are primarily employed during external testing. However, testing of complex and gargantuan VLSI chips like multi-mega-bit memories is extremely time consuming, where test time becomes a critical factor in the overall production cost.

Built-in self test (BIST) is a design technique in which a part of the circuit is used to test the circuit itself. This approach allows all the functions comprising a test to be performed autonomously within a chip, with no external control or test data (often stored as part of the circuit itself). A DFT approach, on the other hand, allows part of the test algorithms to be executed off-chip.

[2] In this paper, we use the terms Built-in self-repair (BISR) and 'fault-tolerance' interchangeably.

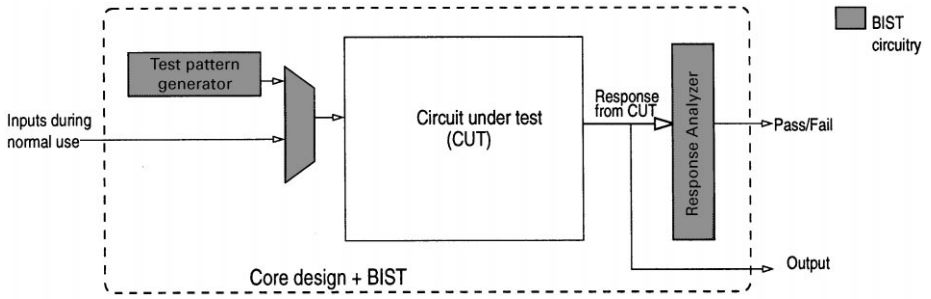


Fig. 1. Structure of a BIST design.

BIST provides a number of advantages over conventional test approaches – namely, the chip does not need to spend much time on a tester; secondly, this technique can be used for embedded designs where I/O pins are often inaccessible. Fig. 1 shows the structure of a typical BIST design.

However, there are several design considerations that need to be examined to prevent the reduction in test cost from being neutralized. These are:

(1) Silicon area overhead – The silicon required to incorporate the test circuitry is a component in the overall real estate requirements of the design. This results in an increase in die size, often associated with decreased yield.
(2) Performance of the overall design – Addition of the BIST circuitry should be done so as to cause minimal impact on the overall speed of the BIST'ed design. The BIST circuitry should be placed away from the critical path within the chip.
(3) Extra pins or time-multiplexed pins – Extra pins or multiplexed pins are often necessary to make the device switch from *operating mode* to *test mode*, because extra control signals are required to achieve this. The use of extra pins should be minimized because pins are an important cost item and have packaging consequences. There is a performance penalty for multiplexed pins due to the multiplexer delay.

A key advantage of BIST is that it can be used for *concurrent* testing, where the tests are performed while the chip is in normal use. If the test mechanism is powerful enough, faults occurring during normal use can be detected and even corrected, thereby increasing the reliability. This form of testing usually relies on some form of information redundancy used in the form of a parity checking or an error-correcting code (ECC). The interested reader is referred to [5] for a detailed survey of BIST techniques employed for VLSI designs.

With increasing complexity of designs, fault-location or diagnosis is gaining considerable importance. Determining the location of the fault is essential if the design has to be repaired *in situ* to make it operational. Furthermore, since defects are caused by imperfections in the design process, an accurate defect diagnosis would provide useful feedback that can be used to improve the mask-alignment and other fabrication processes. The silicon compilation technique proposed in this paper incorporates both fault diagnosis and repair schemes.

3. Review of BIST schemes for ROMs

This section provides a brief summary of the schemes proposed for ROM BIST. Since ROMs often constitute critical part of complex circuits (like microcode in microprocessors), a good BIST scheme with a high fault coverage is desirable. Most BIST techniques are based on compaction of output data (using signature analysis, parity trees, etc.) which is often associated with loss of information. Such a loss might result in the possibility of error escape which effectively causes the signature of the faulty circuit to be identical to the "golden" signature (corresponding to the fault-free circuit). Assume that the ROM under test has $N = 2^m$ n-bit words. The different compaction schemes employed are

(1) Parity based compaction – In this technique [6], each word of the ROM is associated with an extra parity bit. Depending on the type of parity-checking being used, the number of ones could be either odd or even. The BIST circuit consists of a XOR gate and a latch. This scheme can be easily employed for concurrent testing and requires minimal area overhead. However, the principal drawback lies in its high *deception volume*[3] [10], which can be shown to be $2^{n-1} - 1$ for each word.
(2) Count based compaction – This scheme, also known as checksum checking, uses checksums formed by the modulo-k arithmetic of the N data words, where k is arbitrary. Its use for ROM BIST requires high cost in terms of area overhead. For a single precision checksum, i.e., $k = 2^n$, a set of n full adders and n-bit long registers are required. Also the size of the deception volume depends on k and McCluskey et al. [7] show that if $k < 2^n$ and all errors are possible, then the deception volume is independent of k and is equal to $2^{N \times n - n} - 1$.
(3) Polynomial division based compaction – In this technique, the bit stream from the ROM under test is fed to a polynomial divider implemented by a multiple feedback shift register (MISR). The residue that remains in the register at the end of the division is the signature of the circuit and is compared with the golden residue. Savir [5] show that the deception volume of signature analysis is $2^{N \times n - n}$ and is independent of the divisor polynomial. The low area overhead required for the MISR makes this scheme particularly attractive for ROM BIST and a number of approaches aiming at minimizing the probability of error aliasing using signature analysis have been proposed [8,10].

In this work, we have used the BIST scheme proposed by Zorian et al. [10] primarily because of its high fault coverage and reasonable area overhead. Also, the same hardware can be used for ROM repair.

4. Overview of *FTROM*

Fig. 2 gives an overview of the process employed by **FTROM** to create layouts and simulation models for built-in self-testable and fault tolerant ROMs. A library of leaf cells (inverter, nor, or,

[3] Number of words mapping to the same signature as that of the fault-free word.

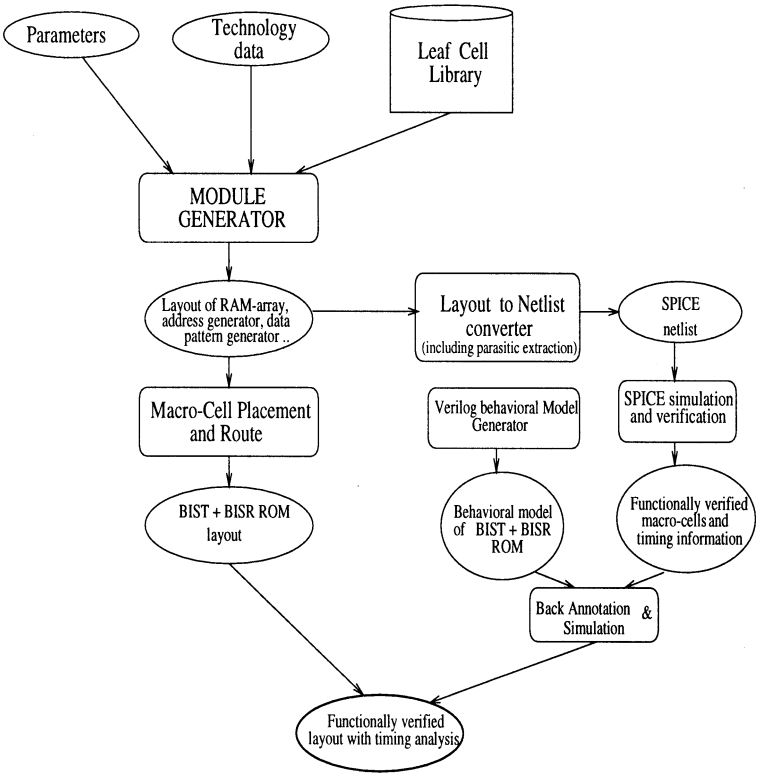


Fig. 2. Overview if **FTROM**.

xor, dff, etc.) is created using a set of user-specified geometry parameters (like buffer size) and technology data (which specifies the CMOSprocess being used and the corresponding design rule constraints). The user can specify constraints on cell height and input/output pin positions. By default, the program generates the leaf-cells of minimum area (minimum height and width) with the power supply lines running in metal 1 at the top and bottom of the cell. This enables the generation of layout of macrocells in a standard-cell layout type of environment where cells can abut; thereby minimizing the number of interconnects. Consider the generation of the layout for an OR gate which is obtained by abutting a NOR and an INV cell. If we consider the leaf cells at level 0 in the hierarchy, OR layout is at level 1. This can be obtained by instantiating a NOR and an INV cell from the leaf cell library. Typically, $height_{\mathrm{NOR}} > height_{\mathrm{INV}}$ (since number of transistors in a NOR is 4 and 2 inINV). If the minimum-sized cells abut, we obtain an OR cell with non-uniform height and need extra interconnect for power supply. To avoid this, we use an INV cell with height specified as $height_{\mathrm{NOR}}$ [or in general, if k cells $c_1, c_2, \ldots, c_k$ have to abut,instantiate each of the cells with height equal to ($height_{c_1}$, $height_{c_2}$, …, $height_{c_k}$)].

FTROM is based on a layout description language, **CDS** (Compiler Development System), a Cascade Design Automation product. The user can specify a particular CMOS process from the set of available 3-metal Cascade Design Automation processes like CDA.5μ3m1p, CDA.7μ3m1p,

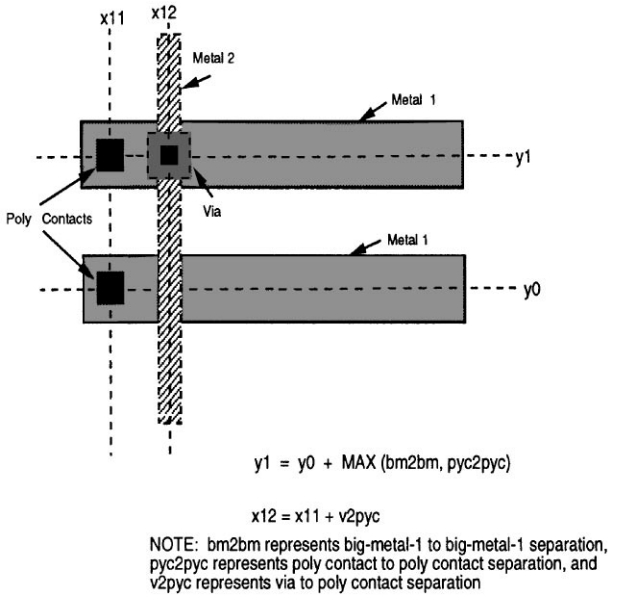


Fig. 3. Concept of landmarks.

etc. The process specific design rules are represented (encoded) by a set of equations, the variables of which are initialized at run-time. The constraints are implemented by using the concept of *landmarks* – horizontal and vertical lines drawn across the layout (typically through the centers of contacts) and the relationship among these specified by various variables (Fig. 3). The layout plane is transformed into a grid by using a set of X and Y landmarks which define the constraints in the verticaland the horizontal direction respectively.

The rectilinear polygons for different layers in the layout are described with reference to the landmark grid. This concept is illustrated by the following example of a simple inverter. A sample layout of an inverter cell is shown in Fig. 4 and the corresponding layout description (using landmarks) is shown in Fig. 5.

This library is used to create a set of macrocells in a hierarchical (bottom-up) fashion. Placement and routing of the macrocells is done so as to reduce the interconnect delays and minimize the overall hardware overhead. The functional correctness of the layouts generated by **FTROM** is verified by simulating the transistor netlist obtained by converting the layout to SPICE format. SPICE simulation also provides timing information of the various modules. However, practical considerations limit the feasibility of this approach for simulation of the complete BISR ROM module for ROMs of practical sizes. Behavioral simulation is a practical approach for such large circuits. Since the functionality of the individual macrocells is known, the correctness of the behavioral model can be easily tested. By back-annotating the SPICE-obtained delays for the individual modules into the circuit netlist,we obtainan approximate estimate of the delays associated with the entire layouts generated by **FTROM**.

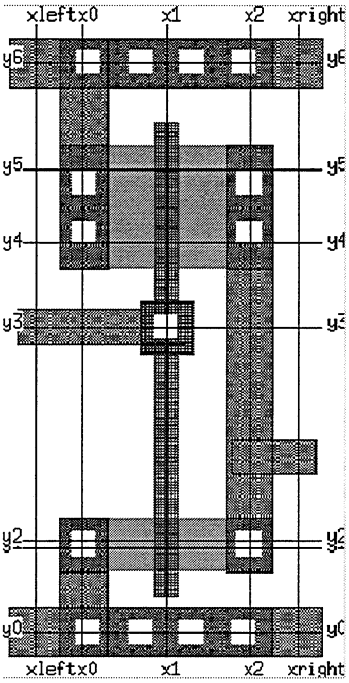


Fig. 4. Layout of the inverter cell (generated by leaf-cell library generator).

5. ROM organization and Fault Modeling

A ROM is a two level AND–OR structure which is usually implemented as a NOR–NOR array. Fig. 6 shows the block diagram of a ROM employing column-multiplexing. The ROM has m inputs (address lines) and n outputs (data lines). The AND array consists of a m-to-2^m decoder (outputs generally known as *Word lines*) and the OR-array stores the n-bitdata corresponding to each of the 2^m *word lines*. The memory cells (in the OR-array) are laid out in an array described by the user-specified *bits per word* (*bpw*) and *bits per column* (*bpc*). There are *bpw* (word size) number of multi-bit columns in a column-multiplexed ROM. The number of rows is total number of words divided by *bits per column* (*bpc*) ($2^m/bpc$). In column-multiplexed ROMs, a single word stores *bpc* *bpw*-bit wide words. Thus the first word occupies bit 0 of each of the *bpw* multi-bit columns in row 0, the second word occupies bit 1 in each of these columns, and so on (Fig. 6). A word is addressed by simultaneously selecting a row and the bit position occupied by that word in the multi-bit columns. By ensuring that *bpc* is a power of 2, it can be easily shown that k lower bits of the address, where $k = \log_2 bpc$ correspond to the column number and the remaining bits specify the row to be selected. Consider the example in Fig. 6 ($words = 64$, $bpw = 5$, $bpc = 4$). The word at address 000101 can therefore be obtained by selecting row 1and selecting column 1.

As shown in [9], most of the single switch-level faults affect the ROM outputs as the single address errors. Karpovsky et al. [9] also show that for the typical sizes of microcode ROMs (e.g. $9K \times 38$ ROM in the HP 9000 CPU chip) and signal processor program ROMs, the percentage of

```
inv_geom(geocellptr,buf_size,output,height)
dmCELL *geocellptr;
int    buf_size; /* specifies the BUFFER SIZE */
int    output;   /* for the location of OUT pin in the cell */
int    height;   /* height/pitch of the cell specified by the user */
{


        nsize = get_n(1,buf_size);
        psize = get_p(1,buf_size);

        portpitch = MAX(v2v,bm2bm);  /* portpitch is assigned the MAX of via-2-via and big metal1 to metal1 */
        halfpitch = portpitch/2;

        *geocellptr = geoSysId(dmCREATE,DIGITAL,"inv","basecell","BUFFSIZE = %d", buf_size);

        define(*geocellptr);

        x0     = 0;
        xleft  = MIN(x0-sma2p,x0-halfpitch);
        x1     = x0+MAX(ac2gt2,bm2bm);
        x2     = MAX(x1+ac2gt2,MAX(x0+bm2bm,x1+bm2bm));
        xright = MAX(x2+sma2p,x2+halfpitch);

        y0 = 0;
        y1 = y0+MAX(nac2pwc,ngtx+ac2ngl);
        y2 = y1+nsize-2*hac;
        y3 = y2+MAX(bm2bm,ac2pyc);
        y3 = MAX(y3,output);         /* To position the OUT pin correctly */
        y4 = y3+MAX(bm2bm,ac2pyc);
        y4 = MAX(y4,y2+pwoac+pw2pac);
        y5 = y4+psize-2*hac;
        y6 = y5+MAX(nac2pwc,pgtx+ac2pgl);

        y6 = MAX(y6,height);         /* for pitch matching */

        /** add the layers **/

        layer("metal");
        wire(bm,x0,y6); y(y4);
        wire(bm,x0,y2); y(y0);
        .
        .

        /** add the PORTS **/

        tport("input",xleft,y3,"metal",NONE,bm,0,"signalports.inputports.Wports");
        .
        .

        enddef();
}
```

Fig. 5. Layout description file for the inverter cell.

the single address faults for $(m,n) = (9,60)$, $(10,20)$, $(13,40)$ is 97%, 98% and 99% respectively. Therefore, a built-in self-repairable scheme which identifies and corrects single address errors is highly desirable.

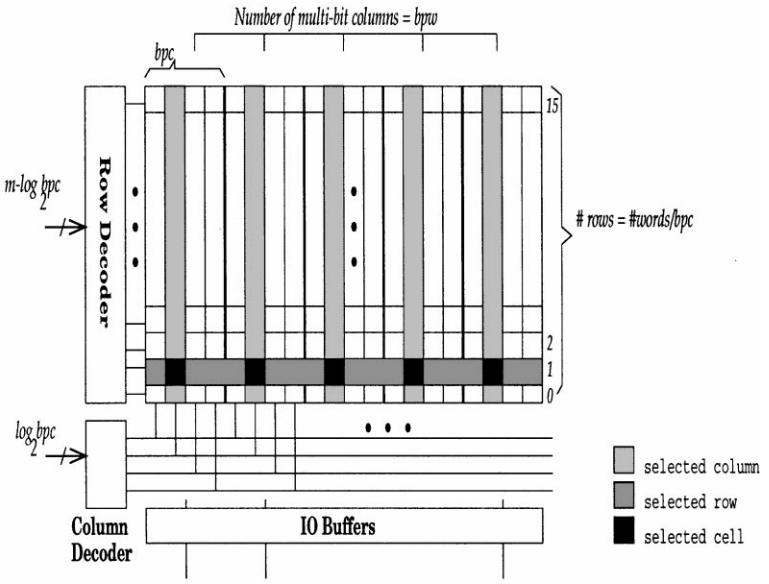


Fig. 6. Block diagram of ROM employing column-multiplexing.

6. Built in self-test and built-in self-repair circuitry

As explained in Section 5, we consider only single-address faults in the ROM array. However, since the ROM employs column-multiplexing, single address error[4] is equivalent to error in *bpc* words of *bpw* bits each. Using the scheme proposed in [9], we would require MISR, TFF register and registers of $bpc \times bpw$ bits which would cause an enormous area overhead. To reduce the area overhead and increase the fault coverage, we assume single address errors are possible in each of the multi-bit columns. The $2^m \times n$ ROM can be viewed as *bpc* ROMs of size $2^{m-\log_2 bpc} \times n$ each. Hence we can use the diagnostic procedure outlined in [9] or each of the multi-bit columns (smaller ROMs). Also we do not require that all the faults be in the same physical row.

Our approach involves two passes. In the first pass, the memory array is tested for single address faults in each of the *bpc* multi-bit columns (Section 6.2) and the faulty addresses and the correct contents are stored in a translation lookaside buffer (TLB). In the second pass, the array is re-tested along with the *corrected* contents of the faulty locations (Section 6.1). Any fault detected in the second pass produces a `repair_unsuccessful` status signal, which implies either too many faults in the memory array, lack of spares or faults probably outside the assumed fault universe. This signal is of extreme importance in an embedded application as it signals the external circuitry of the correctness of the ROM's contents;else, it would be assumed to be correct and hence faulty behavior of the entire circuit.

[4] Error in a single row.

6.1. Built-in self-test scheme

This section presents a brief overview of EEODM [10], the BIST scheme used for re-testing of the ROM along with the corrected contents (stored in TLB after the repair phase). EEODM is based on exhaustive testing for address generation and uses an enhanced form of ODM (Output Data Modification) for compaction of the ROM contents. ODM is a compaction technique which aims at reducing the deception volume by exploiting the non-uniformity of the distribution of the deception volume of the count-based compaction techniques like ones counting. The n-bit test output stream q is modified to have its weight moved away from the median $n/2$. Ideally, the modified stream is the same as the expected stream from the CUT; so that EXORing the two yields a stream of zeros in a fault-free case and a non-zero otherwise.

However, since the data in the ROM to be compacted is a matrix and not a serial bit stream, a MISR is used to serve as a space compactor (parallel to serial compactor). The serial output (quotient) of the stream produced by the MISR is modified using ODM. MISRs often introduce two types of errors – *error cancellation* (during space compaction when the equivalent serial stream does not contain any errors) and *error masking* (occurs during time compaction i.e. when the equivalent serial stream is a multiple of the divisor polynomial implemented by the MISR). EEODM further minimizes the probability of error cancellation by using bidirectional polynomial division. This is motivated by observing that the patterns of errors that may result in error cancellation all form diagonals in the output data matrix. By changing the direction of the shift of the MISR, most patterns cancelled in one direction would not be cancelled in the opposite direction.

Thus, the BIST scheme requires two test phases. In the first phase, the complete contents of the ROM as well as the additional modifier stream are read and compacted using a $n + 1$ bit MISR that has one direction of shift, say right to left. The final contents of the MISR are used to initialize the second phase during which the contents of the ROM (excluding the modifier column) are read and ODM is applied to the quotient bit generated by the n-bit MISR shifting in the opposite direction. A finalMISR residue of zero and zero-count in the EXORing of the quotient and the modifier stream indicates fault-free circuit. Fig. 7 shows the hardware requirements of EEODM.

6.2. Built-in self-repair scheme

In the following, we give a brief description of the built-in self-diagnostic scheme. The reader is referred to [9] for details. Fig. 8 shows the block diagram of the built-in self-repairable ROM. Suppose we are testing bit i in each of the multi-bit columns. The **aux_counter** ($\log_2 bpc$ bit counter corresponding to column address) is initialized to i. The T flip-flop register (TFF reg.) and the MISR are initialized to the signatures corresponding to column i.

In the repair mode, the **main_counter** (m-$\log_2 bpc$ bit counter corresponding to the row address) generates all the $2^{m\text{-}\log 2\ bpc}$ input patterns. For each input applied, the contents of the corresponding address are parallely loaded into the TFF reg. and simultaneously into the MISR which is shifted to the next state. If the contents of the TFF reg.are zero at the end of this process, it implies that the

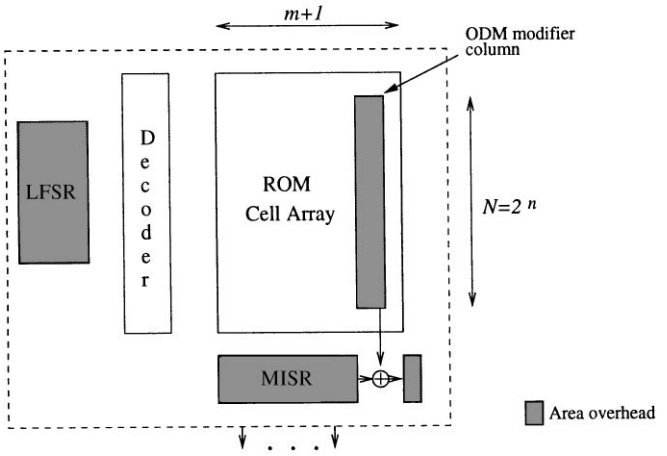


Fig. 7. EEODM for ROMs [10].

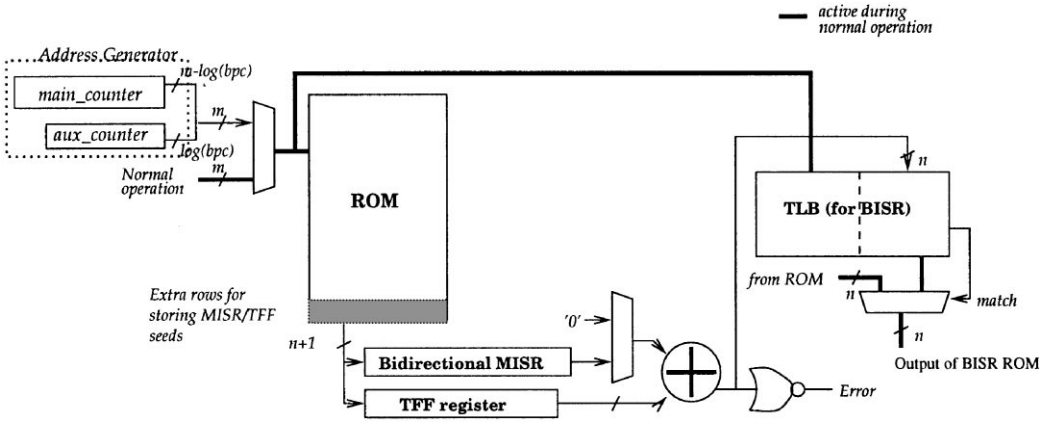


Fig. 8. Block diagram of BISR and BIST ROM.

column i (in each multi-bit column) is error-free. However, a non-zero residue in TFF register signals an error. The MISR is then configured as an autonomous LFSR (implementing the reciprocal polynomial). With each clock, the contents of LFSR are compared with those of TFF reg. and the **main_counter** is decremented. A match indicates that the faulty address is the current contents of the **main_counter** and the correct contents are $output_{ROM} \oplus TFF$. These are then stored in the TLB. This process is repeated for each of the *bpc* multi-bit columns.

The signatures for the MISR and the TFF reg. can be calculated as in [9] and stored in two additional rows in the ROM.

7. Circuit implementation

7.1. Column multiplexed addressing

Column multiplexed addressing is implemented by sending the outputs of thecolumn decoders to pass transistor multiplexers, which select one bit from each of the *bpc* multi-bit columns (Fig. 9). The two spare rows added (for storing MISR signatures and TFF residue) form part of the main array and an extra-bit in the address field is used to enable either the ROM contents or the extra two rows added for BIST and BISR purposes. A leading 0 is appended for the regular ROM addresses and a 1 for the extra two rows. As the address space spanned by the leading-1 rows is limited to the extra rows appended, a complete $2^{m-\log_2 bpc+1}$ row decoder is not required. Only the rows for the addresses $\{0, 1, \ldots, 2^{m-\log_2 bpc-1}-1, 2^{m-\log_2 bpc-1}, 2^{m-\log_2 bpc-1}+1\}$ exist in the new row decoder.

Column multiplexing improves the bandwidth of memory devices by increasing the number of bits accessible in a single cycle. This technique would be of immense use in embedded applications like micro-code ROMs.

7.2. Address generator

The repair phase requires the reading of $2^{m-\log_2 bpc}$ ROM locations in each of the *bpc* multi-bit columns. Also,once a fault is detected, we need to configure the $m-\log_2 bpc$ counter in *down*

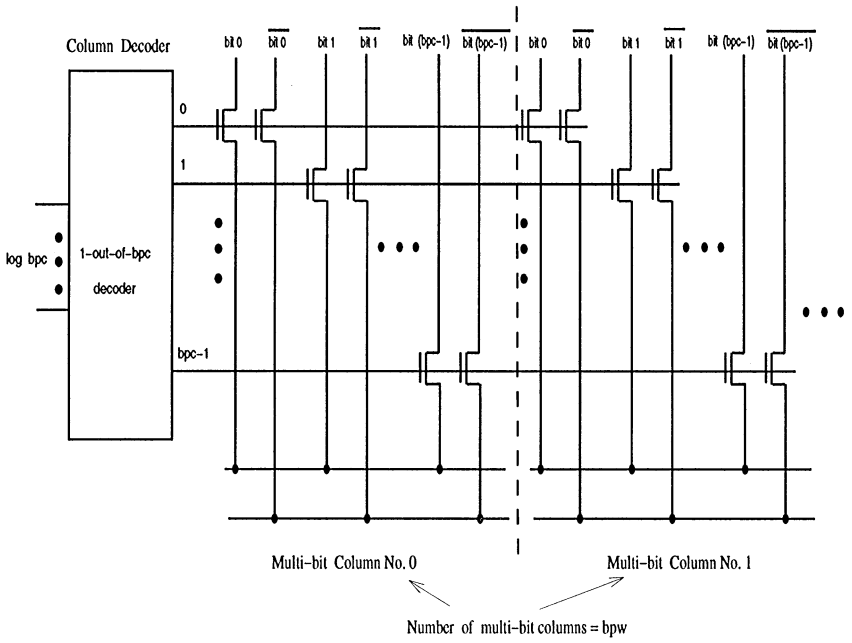


Fig. 9. Column multiplexed addressing.

mode. This requires a $m - \log_2 bpc$ bit up-downcounter and a $\log_2 bpc$ bit counter. During the second phase (re-testing of BISR ROM), these two counters can be used for cycling exhaustively through the address space.

7.3. Self-test and self-repair control logic

The self-repair and self-test controller has been implemented as a psedo-NMOS NOR-NOR PLA. The controller area is constant independent of the size of the ROM under test and is found to be approximately 0.1% of the memory area for a $2^8 \times 38$ ROM. The PLA is placed close to the region where the density of control signals is a maximum to reduce the interconnect delays.

7.4. Built-in self-repair circuitry

The faulty addresses detected during the repair phase are stored in a translation lookaside buffer (TLB). The proposed circuit design uses an innovative implementation for a TLB that achieves a very low delay overhead built-in self-repair. Whenever a fault is detected in the repair phase, the faulty address is stored in the CAM, and the corrected contents are stored in the corresponding row of the RAM. In the normal mode, the input address is compared with the set of faulty addresses stored in the CAM part of the TLB. Since the comparison with the set of faulty addresses is done in parallel, this approach has a negligible matching time as compared to if the scheme proposed in [9] were generalized; i.e., using a set of registers for storing the faulty addresses and corrected contents and a set of XOR gates along with a multi-input OR to detect the match (Fig. 10). Comparison of the delays incurred in the two approaches are shown in Fig. 12.

Suppose the user specifies s spares. Since we are assuming at-most one faulty location per column in every multi-bit column, the maximum number of faulty locations allowed $\leqslant bpc$. In a CAM,

$$match = match_0 + match_1 + \cdots + match_{s-1}.$$

If for an input address i, a match is detected, then there exists exactly one row j in CAM such that $match_j = 1$. Therefore, the contents of row j of RAM are the correct contents of the ROM; else the contents from the ROM are outputted. The total delay encountered is:

$$t_{\text{BISR\_ROM}} = \begin{cases} \max(t_{match}, t_{precharge}) + t_{sense} + t_{mux} & \text{if } match = 1 \\ \max(t_{match}, t_{\text{ROM}}) + t_{mux} & \text{otherwise.} \end{cases}$$

Therefore, the maximum delay encountered is

$$t_{\text{BISR\_ROM}} = \max(\max(t_{match}, t_{\text{ROM}}), \max(t_{match}, t_{precharge}) + t_{sense}) + t_{mux}.$$

Experimental results show that t_{ROM} dominates t_{match} and $\max(t_{match}, t_{precharge}) + t_{sense}$. Hence, the delay penalty is t_{mux} which is approximately 0.079 *ns* for 2.0 μ pwell CMOS process.

The TLB circuit for the fault-tolerant ROM is shown in Fig. 10 (corresponding layout in Fig. 11). During the repair phase, the shift register keeps track of the CAM array to be filled with

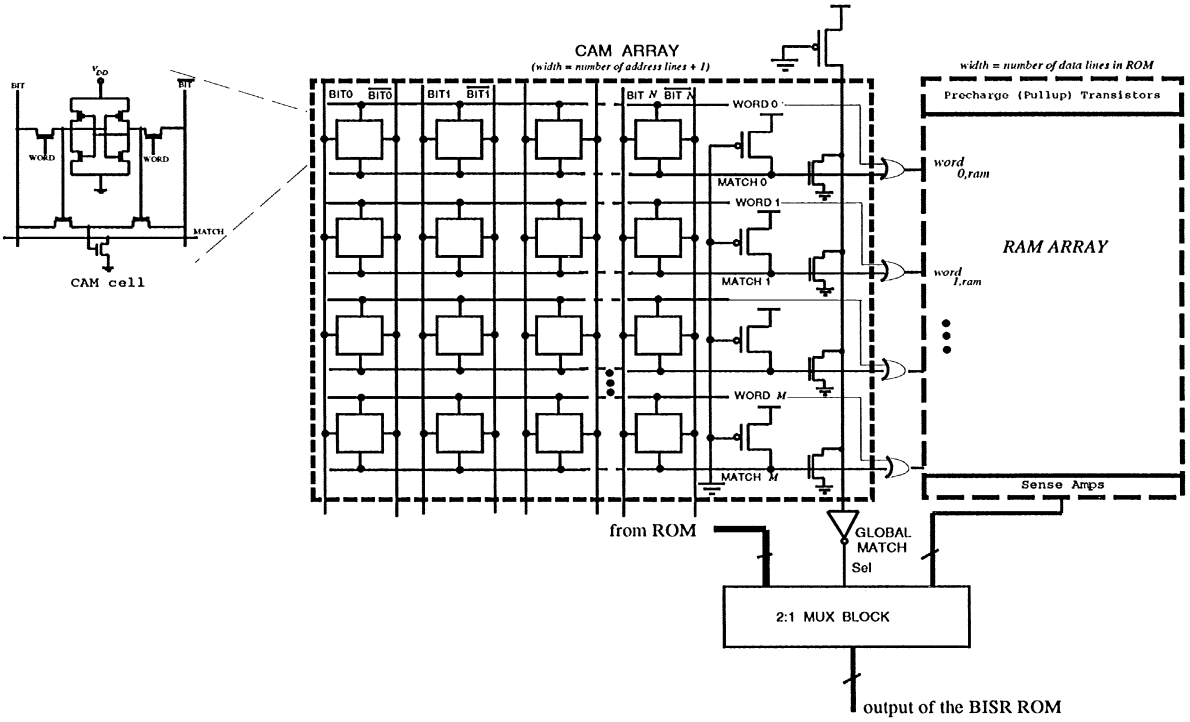


Fig. 10. TLB circuit used for BISR.

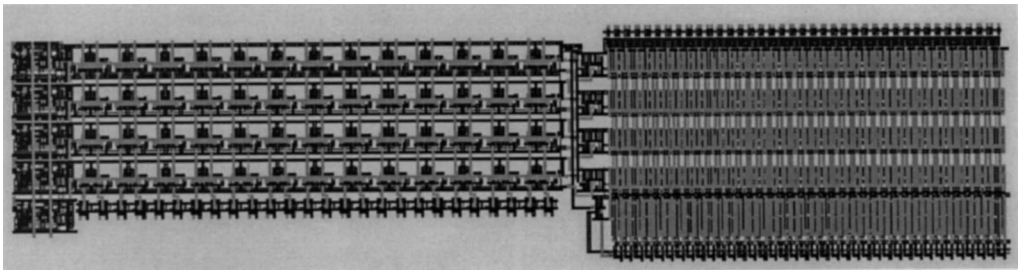


Fig. 11. TLB layout (*buffer size* = 1, *number of spares* = 8, *number of address lines* = 13, *Word size* = 38, *process*: CDA.7μ3m1p).

the next faulty address. A row in the RAM array is selected either in the repair phase (when the contents of flip-flop in the shift register for that row is high) or during the normal mode, when a match is detected between the contents of the corresponding CAM array and the input contents.

The advantages of using a TLB for the BISR scheme (over other schemes like storing faulty addresses and the corrected contents in registers) are:

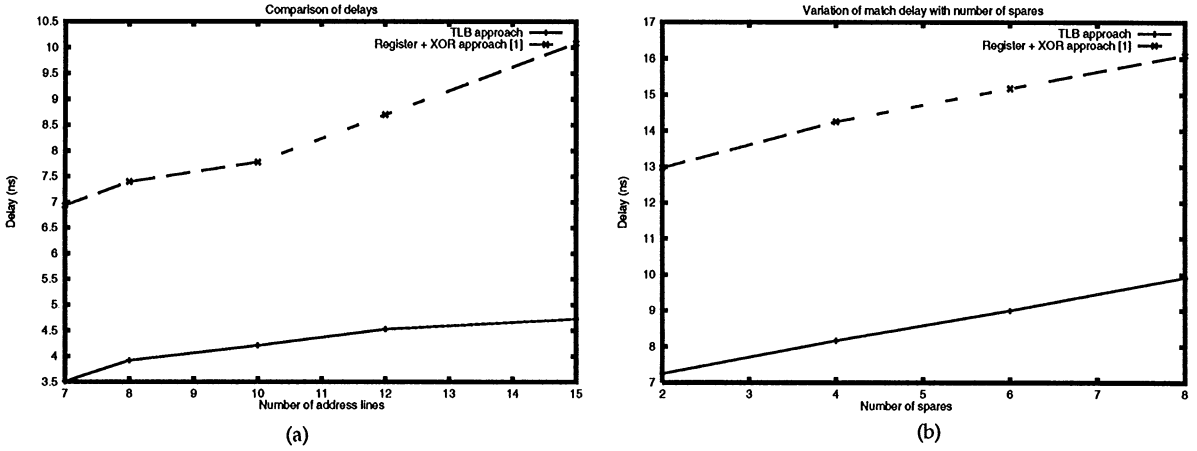


Fig. 12. Comparison of the match operation delay for TLB and register + XOR approach: (a) variation with number of address lines ($number_{spares} = 1$) (b) variation with number of spares ($number_{address\_lines} = 13$).

- **Low area overhead** – The addresses are stored in the CAM (Content Addressable Memory) array and the area of a CAM cell is significantly lower than that of a D flip-flop cell (used in [9]). This is based on the observation that a typical DFF cell has approximately 30 transistors in comparison to 9 required for a CAM cell. [For CMOS process CDA.7μ3m1p, **FTROM** generates a DFF cell of area 1.39×10^3 μ^2m and a CAM cell of area 0.792×10^3 μ^2m]. Also, the CAM cell performs comparison; therefore, no extra gates are required for the match operation.
- **Fast access time** – The scheme proposed in [9] employs a set of XOR gates and a multi-input OR to detect a match between the input address and a faulty address. However, the match is much faster in a CAM; hence much less penalty on the ROM access time. Fig. 12a shows the comparison of times required by the TLB vs. that required by using XOR and a multi-input OR gate. Note that the TLB match times is practically independent of the number of address lines in the ROM; whereas, the delay is proportional to the number of address lines m in the latter case (because we require an m-input OR gate for each spare).
- **Easily scalable to the number of spares available, without causing extra delay penalty** – Circuit simulations using SPICE confirm that the delay penalty is only slightly dependent on the number of spares available (Fig. 12b) unlike competing approaches where the delay increases significantly with the number of spares and the number of faulty addresses stored within the BISR circuitry.

7.5. Bidirectional MISR

We have used the EEODM concept for testing of the ROM. This requires an additional ODM column in the main ROM. Since we are assuming single address faults, there might be potentially $n + 1$ bits in error per word. Hence we need a $n + 1$ bit up-down MISR for the repair phase. During the re-testing of the repaired ROM, we need a bidirectional MISR [10]. Using two MISRs for the two phases would result in a large area overhead. However, we can use the MISR used in repair

Table 1

Phase	Number of bits	Polynomial implemented	Comments
Repair	$n+1$	$P_l(x)$	$L \rightarrow R$
	$n+1$	$P'_l(x)$	$R \rightarrow L$ (in autonomous mode)
Test	$n+1$	$P_r(x)$	$R \rightarrow L$
	n	$P^n_l(x)$	$L \rightarrow R$ (only ROM[0..$n-1$] are fed to MISR)

phase for BIST also. Let $P_l(x)$, $P_r(x)$ be the polynomials (of degree $n+1$) implemented by the MISR when shifting from left to right and right to left respectively. $P'_l(x)$ represents the reciprocal polynomial of $P_l(x)$savir and $P^n_l(x)$ is the n-degree polynomial derived from $P_l(x)$ by feeding only bits $[0..n-1]$ of the ROM outputs. Table 1 outlines the desired operations.

8. User interface and features

The user has to specify a set of circuit parameters for the word-wide ROM employing column-multiplexed addressing. In this scheme, a single column stores multiple bits, the number of bits per column denoted by *bpc*. The number of bits per memory word is denoted by *bpw*. A log- *bpc* to *bpc* column decoder chooses exactly one out of *bpc* bitline pairs from each of the *bpw* IO sub-arrays, producing a *bpw*-bit word. This allows for a higher memory bandwidth and regular layouts of the memories. The user specified parameters include - *number of address lines*, *bpw*, *bpc*, and the *codefile containing the contents of the* ROM. As described earlier, the BISR and BIST scheme employed uses MISR for error detection and correction. The user has to specify the polynomials for the corresponding MISR(s). FTROM generates the MISR seed depending on the ROM contents and the polynomial specified, which is then hardwired while generating the layouts of the corresponding macrocells (like TFF register, MISR). The *n* and *p* transistors are automatically sized to balance the rise and fall times.

The features of **FTROM** that make it attractive to a CAD designer include:

- **Design rule independence** – As explained in Section 4, the user specifies the CMOS process for which the layout is generated. This implies that **FTROM** achieves *design-rule independence*, i.e. the ability to generate layouts for a wide range of processes achieved by representing the process specific design rules by a set of equations (landmarks).
- **Structured design** – **FTROM** exploits the array-like regularity of the various modules to build the macrocells and interconnect them. The structured design approach aids in reducing the interconnects (by placing the modules abut) and thereby minimizing signal propagation times.
- **Efficient place-and-route** – The sizes of the macro-cells increases linearly with width of the ROM. **FTROM** sorts the rectangular macrocells in decreasing order of areas and uses heuristics to make the final layout as squarish as possible. Also timing delays are reduced by an efficient placement strategy that tries to place ports to be connected as close as possible and often uses over-the-cell routing (using third metal), instead of global or channel routing to reduce the interconnect length and hence, the delays.

- **Small area overhead** – The area overhead with BISR and BIST is approximately 19% for a $2^{13} \times 38$ ROM. The delay penalty occurred is due to a multiplexer, which is approximately 0.079 ns for 2.0 μ CMOS pwell process (see Section 10.1 for details).

8.1. Comparison with error-control coding (ECC)

Prior approaches for fault-tolerance have primarily employed various coding schemes. However, for error correction, there should be sufficient check bits contained in the code word to encode all possible ways of error occurrence plus the no-error situation. The number of possible ways of error occurrence is determined by the targeted error correction capability. For example, suppose that the input data words are b bits wide and we want single error correction. If c check bits are used, then a single error can occur in any of $b + c$ different locations. This, plus the no-error situation, gives a total of $b + c + 1$ possible ways of having at most one error. To distinguish all these possibilities from one another, the following inequality must be satisfied: $2^c \geqslant b + c + 1$. In a 4-bit word ($b = 4$), therefore, 3 check bits ($c = 3$) make this inequality just hold. In a 38-bit word (e.g. for a $2^{13} \times 38$ ROM), the minimum number of check bits needed to satisfy the above inequality is 6, causing an area overhead of 15.8% for the ROM array alone, excluding the encoder/decoder circuitry. Furthermore, using 6 check bits would achieve only single-error correction (i.e. a single faulty bit in each multi-bit ROM column); larger number of check bits would be needed for multiple error correction, causing the area overhead to be *significantly* larger than the approximately 19% overhead produced by our scheme.

9. Layout and simulation model generation

9.1. Module layout generation

The module layout generation process in **FTROM** is hierarchical. Using the user-defined geometry parameters, **FTROM** creates a database of leaf cells (at level 0). At the next level, structured custom design is employed to build macrocells. This process is repeated to create macrocells at different levels of hierarchy. At each stage, pitch-matching and width-matching techniques are employed to avoid any significant routing overhead. Routing is performed by defining channels between macrocells and placing the routes in those channels. Left Edge Algorithm (LEA) is applied to perform channel routing between ordered sequences of ports on both sides of the channel. The channel width is proportional to $K \times D_{m2m}$ where K is the number of ports and D $\{m_2m\}$ is the minimum spacing required between two interconnects.

9.1.1. Pitchmatching (horizontal alignment)

The layout of a leaf cell generated by **FTROM** is *elastic*, which means that the layout can be "stretched" in any desired direction to enable it match some standard pitch. However, the desired pitch should be at least the minimum pitch of the cell in that direction, because the leaf cell generator creates the layouts of the leaf cells of minimum area and any attempt at minimizing the dimension would lead to design rule violations. Stretching of the layout can be easily implemented using landmarks by passing the reference pitch as an input parameter to the geometry generator module.

9.1.2. *Width-matching* (*vertical alignment*)

Vertical alignment (of the ports) is required while connecting two modules. It may often be the case that the ports in one module may be spaced widely as compared to the other. Using a naive approach of placing one of the modules aligned with the edge of the other can lead to significant routing overhead. **FTROM** uses a combination of (a) Stretching one block horizontally until its width equals that of the other, and (b) Positioning one of the cells with respect to the other with an objective of minimizing the routing area overhead to obtain optimal (or near-optimal) layouts. In some cases, where the difference in the widths is very small, the first approach may be advantageous, as the increase in the area overhead by expanding the cell masks the routing area and interconnect delay which would have been incurred using the second approach. However, if the difference is large, the second approach would be better.

9.2. *Floorplan*

Fig. 13 shows the floorplan of the BISR ROM generated by **FTROM**. It includes a memory array in which bit-lines, implemented in metal, run vertically and word lines, implemented in polysilicon, run horizontally. To reduce the word line series resistance and hence, the word-line delay, first metal is shunted onto the polysilicon lines [11] (Fig. 14).

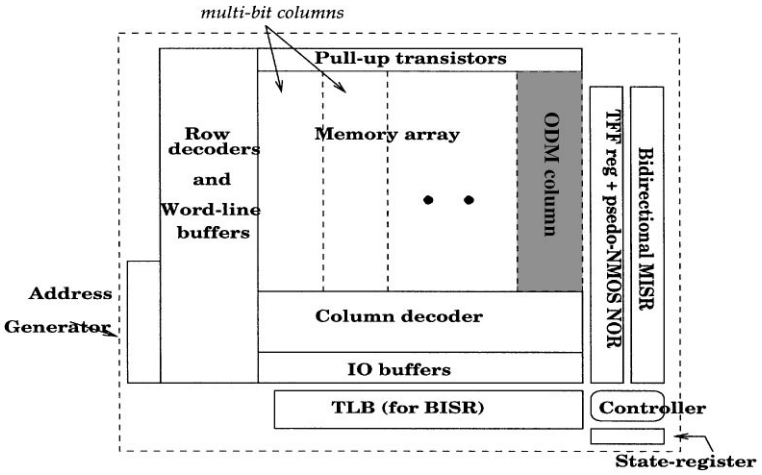


Fig. 13. Approximate floorplan of BISR ROM generated by **FTROM**.

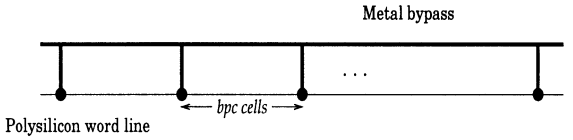


Fig. 14. Reducing word-line delay by metal-1 shunting.

Table 2
Lower-bound on the area overhead for a typical ROM

ROM size	Number of address lines (m)	Number of data lines (n)	BPC (b)	Theoretical lowerbound on area	Actual area overhead
$8K \times 38$	13	38	8	16.8%	19.2%

The ROM decoder is a pseudo-NMOS decoder. Experimental results showed that the delay through a psedo-NMOS decoder was much less as compared to a decoder employing predecoding [11]. Although the number of transistors required in a pseudo-NMOS decoder is large, area-efficient delays can be obtained by packing the transistors densely.

9.3. Design rules verification

The concept of landmarks ensures design-rule correct layouts generated by **FTROM**. However, they are re-verified using ICRules™, a Mentor Graphics product.

9.4. Generation of simulation models

After the generation of layouts of the building blocks (leaf cells and macrocells like counter, MISR, etc.) are simulated using SPICE to verify their functional and timing correctness. The SPICE netlist (extracted using Mentor Graphics tools) contains information regarding the parasitics and can be used for timing analysis. The individual leaf cells and macrocells with fewer than 3000 transistors are thus simulated and verified using this approach.

The final BISR ROM layout contains transistors of the order of 10^6 for ROMs (Table 2) of practical interest. Simulation using SPICE is infeasible for such a large netlist. Behavioral simulation is a practical approach for verification of such large circuits. Since the functionality of the individual macrocells could be tested using SPICE, their behavioral models can be easily tested and delays back-annotated. **FTROM** generates a (hierarchical) behavioral model for the complete circuit which can be used for estimation of critical paths and other timing information.

10. Results

10.1. Area overhead

The following components contribute to the hardware overhead in our fault-tolerant design – ODM column, Test address generator, T flip-flop register, Bidirectional MISR and the TLB. The theoretical lower-bound on the area overhead can be calculated by computing the sum of the areas of the above macrocells individually as a fraction of the original ROM. This analysis assumes the blocks can be placed as close as possible, thereby requiring no routing area. This is an optimistic lower bound, because in VLSI, a significant part of the layout area is used for interconnects.

Table 2 summarizes the result obtained for ROM used in HP9000 CPU. A small deviation from the lower bound indicates that the tool is successful in achieving high-quality layouts, comparable to custom design approach.

10.2. Delay overhead

As described in Section 6.2, the extra delay incurred is due to the multi-plexer which selects either the outputs of the TLB or the ROM depending on the address input. Thus the delay overhead is t_{mux} which is approximately 0.0729 ns for CMOS 2.0 μ pwell process. During normal mode of operation, if a *faulty* access occurs (i.e. access to a faulty location), the effective access time is t_{access}(TLB) instead of t_{access}(ROM). For non-faulty accesses, the degradation of access time is minimal ($= t_{mux}$).

10.3. Yield improvement of ROM layouts with built-in self-repair

Yield for a die is defined as the probability that the die is "good". **FTROM** generates layouts for fault-tolerant ROMs assuming that at most one row is faulty for a given *bpc*-bit pattern. Thus it allows for a maximum of *bpc* word faults.

Suppose we use the Poisson model for a single cell yield, i.e.,

$$Y_{\mathrm{sc}} = \mathrm{e}^{-\lambda_{\mathrm{sc}}}$$

where λ_{sc} represents the average number of defects per cell. In this model, each defect is regarded as a perfect point. The original yield of the memory array without fault-tolerance can be calculated assuming the well-known yield formula due to Stapper [12,13]:

$$Y_{old} = (1 + A\delta/\alpha)^{-\alpha},$$

where δ is the defect density, A is the area of the memory array, and α is some clustering factor of the defects. Let P_{r} denote the probability for a defect pattern to be repairable with respect to the fault universe and extra circuitry. Then the yield Y_{new} is:

$$Y_{new} = Y'_{old} + (1 - Y'_{old})P_r$$

where $Y'_{old} = [1 + (A + B)\delta/\alpha]^{-\alpha}$ and B is the area of the additional circuitry for fault-tolerance.

The probability of having at least one fault in a *bpw*-bit word is $1 - Y_{\mathrm{sc}}^{bpw}$. Assuming a uniform defect distribution, for a given number d of single cell defects, λ_{sc} is inversely proportional to the product $(N + S) \times bpw$ (N is the number of words in the memory array and S is the number of spares provided). Hence,

$$Y_{\mathrm{sc}} = \mathrm{e}^{-d/(N+S)\times bpw}.$$

It follows that the probability P^f_{word} that a word is faulty is

$$P^f_{word} = 1 - \mathrm{e}^{-d/(N+S)}.$$

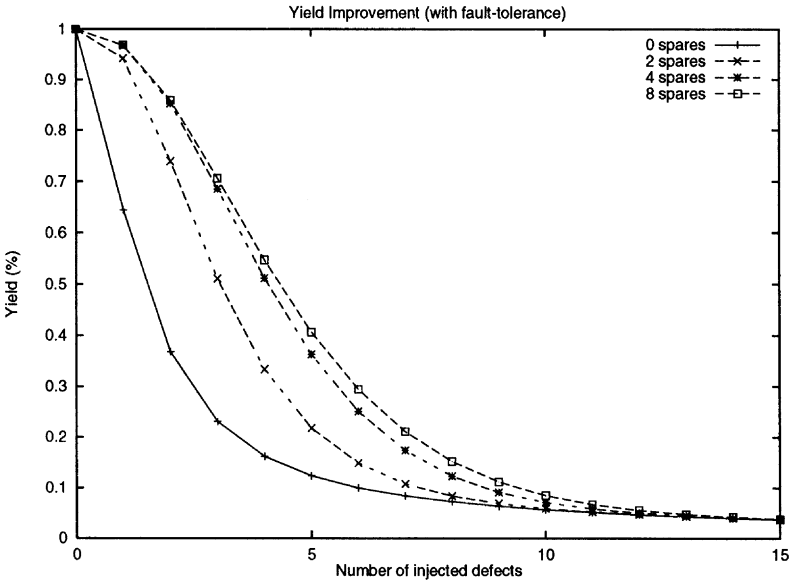


Fig. 15. Yield improvement with fault-tolerance.

Also, the probability of exactly one faulty word for a given *bpc* is

$$P(\textit{one faulty word}) = f_1 = \binom{N/bpc}{1} \cdot P_{word}^{f} \cdot (1 - P_{word}^{f})^{N/bpc - 1}$$

and the probability of no fault is $f_0 = (1 - P_{word}^{f})^{N/bpc - 1}$. For successful repair, it must be ensured that atmost one word is faulty for a given value of *bpc*, the total number of such faults is less than or equal to the number of spares and the spares (in TLB) are fault free. Therefore P_r (given S spares) is

$$P_r = P(\textit{total number of faults} \leqslant S) \cdot P(\textit{spares are fault} - \textit{free})$$

$$= \sum_{k=0}^{S} \binom{bpc}{k} \cdot f_1^k \cdot f_0^{bpc-k} \cdot (1 - P_{word}^{f})^k.$$

The yield improvement for $8K \times 38$ ROM with $bpc = 8$ ($N = 2^{13}$) and spares $S = 0, 2, 4, 8$ is shown in Fig. 15.

10.4. Sample layout plot

Typical layout plots produced by **FTROM** are shown in Fig. 16. The ROM size is $8K \times 38$. An example of the circuit floorplan produced by the tool is shown in Fig. 16a and the corresponding layout is shown in Fig. 16b.

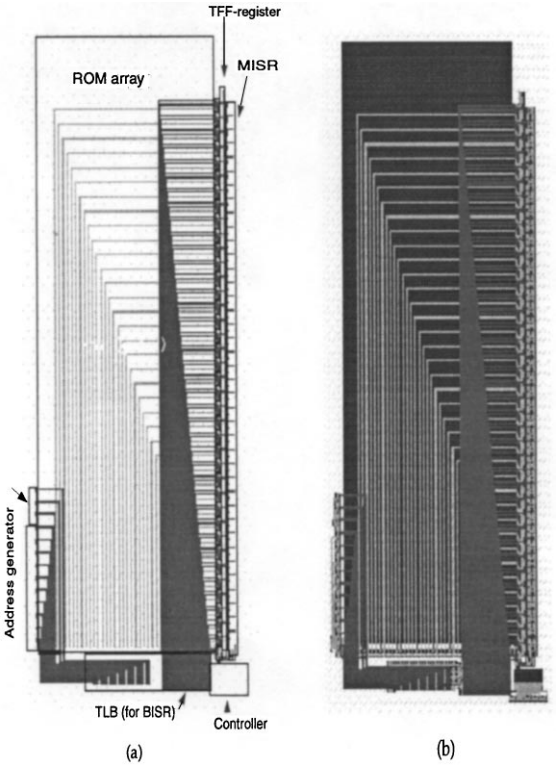


Fig. 16. (a) Floorplan of $8K \times 38$ ROM produced by FTROM ($bpc = 8$, *number of spares* $= 4$) (b) corresponding layout.

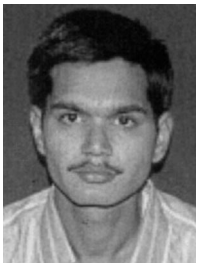
11. Conclusion

The ROM layouts produced by **FTROM** use efficient circuit techniques for fast access and high data bandwidth. Such techniques include: static row and column address decoding, column multiplexed addressing. The area overhead with BIST and error-correction alone (i.e. without considering the redundant rows) is quite modest. The TLB technique used for built-in self-repair allows for a negligible penalty on the ROM access time. For CAD designers, **FTROM** affords a user-friendly and flexible way of implementing built-in self-repair in ROMs. The technology independence of the software minimizes the redesign efforts when changing or scaling the technology. Currently, the task of integrating BIST and ECC circuitry with ROM layouts is usually done manually or semi-custom by CAD designers, and is thereby a major bottleneck in the design cycle. Well-designed silicon compilers like **FTROM** are thus very effective tools for improving the CAD designers' efficiency in designing high-quality layouts and decreasing the time-to-market.

References

[1] Q. Wu et al., LILA: layout generation for iterative logic arrays, IEEE Trans. Comput. Aided Des. 14 (11) (1995) 1359–1369.

[2] R.W. Linderman et al., Design and application of an optimizing XROM silicon compiler, IEEE Trans. Comput. Aided Des. 8 (12) (1989) 1267–1275.
[3] W.P. Swartz et al., CMOS RAM, ROM and PLA Generators for ASIC Applications, Proc. IEEE 1986 Custom Integrated Circuits Conf. 1986, pp. 334–338.
[4] K. Chakraborty, BISRAMGEN – A silicon compiler for built-in self-repairable random-access memories, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, May 1997.
[5] P.H. Bardell, W.H. McAnney, J. Savir, Built-In Test for VLSI, Wiley, New York, 1987.
[6] A.J. van de Goor, Testing Semiconductor Memories: Theory and Practice, Wiley, New York, 1990.
[7] N. Saxena, E.J. McCluskey, Arithmetic and Galois CheckSums, Proc. IEEE Int. Conf. on Computer Aided Design, November 1989, pp. 570–573.
[8] O. Kebichi, V.N. Yarmolik, M. Nicolaidis, Zero aliasing ROM BIST, J. Electron. Testing: Theory and Appl. 5 (4) (1994) 377–388.
[9] P. Nagvajara, M.G. Karpovsky, Built-in self-diagnostic read-only-memories, Proc. IEEE Int. Test Conf., Washington, DC, September 1991, pp. 695–703.
[10] Y. Zorian, A., Ivanov, EEODM: an Effective BIST scheme for ROMs, Proc. IEEE Int. Test Conf. Washington, DC, September 1990, pp. 871–879.
[11] Jan M. Rabaey, Digital Integrated circuits: A Design Perspective, Prentice Hall, Englewood Cliffs, HJ, 1995.
[12] C.H. Stapper, A.N. McLaren, M. Dreckmann, Yield model for productivity optimization of VLSI memory chips with redundancy and partially good product, IBM J. Res. Dev. 24 (1980) 398–409.
[13] C.H. Stapper, On yield, fault distributions, and clustering of particles, IBM J. Res. Dev. 30 (3) (1986) 326–338.



Anurag Gupta received the B.S. degree in Computer Science and Engineering from the Indian Institute of Technology, Delhi, India in 1995 and the M.S in Computer Engineering from the University of Michigan, Ann Arbor in 1997.

He is currently with Intel Corp., Santa Clara, CA, where he is involved in the design and application of CAD software engineering methods and research techniques specifically to datapath designs. His interests include CAD and genetic algorithms, design for testability and IP issues.

Kanad Chakraborty received a B.Tech degree in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur, in 1989, an MS degree in Computer and Information Science from Syracuse University in 1992, and a Ph.D. degree in Computer Science and Engineering from the University of Michigan, Ann Arbor in 1997. During the summer of 1994, he worked as a Design Automation Engineer in the area of board-level circuit design at Omniview Inc., Pittsburgh. In 1996, he joined IBM Microelectronics, EDA Division as an Advisory Software Engineer and Scientist to work in the area of ASIC physical design. Presently, he is working at IBM Microelectronics, East Fishkill, and at the IBM Thomas J. Watson Research Center, Yorktown Heights, in the areas of ASIC placement and synthesis. His research interests are in the areas of VLSI physical design and logic synthesis, built-in self-test and fault-tolerance of semiconductor memories, board-level design-for-testability, and neural networks. His journal and conference publications are on a wide range of topics including technology-related testing of RAMs, board-level testing of multiport memories and interconnects, and neural net applications in time-series analysis, algebraic equation-solving, and pattern recongnition. He has co-authored a book, entitled *Testing and Testable Design of High-Density Random-Access Memories*, Kluwer Academic Publishers, 1996 and a book chapter, entitled *Forecasting the Behavior of Multivariate Time-Series Using Neural Networks, in Artificial Neural Networks: Forecasting Time Series*, IEEE Computer Society Press, 1994, edited by V. Vemuri and R.D. Rogers. As part of his Ph.D. research, he wrote a layout generation tool, BISRAMGEN, that synthesizes built-in self-repairable RAM modules employing a novel circuit technique for which an international patent is pending.

Pinaki Mazumder (S'84–M'87–SM'94) received the B.S.E.E. degree from the Indian Institute of Science in 1976, the M.Sc. degree in computer science from the University of Alberta, Canada, in 1985, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 1987.

Presently, he is with the Department of Electrical Engineering and Computer Science of the University of Michigan, Ann Arbor as a Professor. Prior to this he was a Research Assistant with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign for two years and was with the Bharat Electronics Ltd. (a collaborator of RCA), India, for over six years, where he developed several types of analog and digital integrated circuits for consumer electronics products. During the summer of 1985 and 1986, he was a Member of the Technical Staff in the Indian Hill branch of AT\ and T Bell Laboratories. During 1996–1997, he spent his sabbatical leave as a visiting faculty at Stanford University, University of California at Berkeley, and Nippon Telephone and telegraph, Japan.

His research interests include VLSI testing, physical design automation, and ultrafast circuit design. He has published over 100 papers on these topics in archival journals and proceedings of the international conferences.

Dr. Mazumder was a recipient of Digital's Incentives for Excellence Award, BF Goodrich National Collegiate Invention Award, National Science Foundation Research Initiation Award and Bell Northern Research Laboratory Faculty Award.

Dr. Mazumder has lead his research group's efforts in VLSI testing and built-in self-repair techniques and has developed silicon compilers for RAM, ROM and PLA with built-in self-repairable capabilities. Two international patents on this research work are pending now. He is the co-author of two books – "*Testing and Testable Design of High-Density Random-Access Memories*", Kluwer Academic Publishers, and "*Semiconductor Random-Access Memories*: *Testing and Reliability*" IEEE Computer Press (to be published). He was a Guest Editor of the IEEE Design and Test Magazine's special issue on multimegabit memory testing, March 1993 and the Journal of Eletronic Testing – Theory and Applications special issue on memory testing and reliability, June 1994.

Dr. Mazumder has also done quite extensive work in the area of VLSI physical design. He has developed a suite of distributed physical design tools and has co-authored a book on the subject, entitled: *Genetic Algorithms for VLSI Design*, *Layout and Test Automation*, Prentice Hall. He is also working on a book entitled: *Models and Techniques of VLSI Routing*.

Dr. Mazumder has worked over six years as an integrated circuit designer in semiconductor companies. He is currently leading ultrafast circuit design activities for nano and quantum electronic devices. He has successfully developed CAD tools for high-performance VLSI circuit simulation (NDR-SPICE) and numerous circuit topologies for quantum MOS and other quantum-well devices. Several US and Japanese semiconductor companies including Texas Instruments, Hughes Research Laboratory, Lockheed Martin, NTT and NEC have been collaborating with him on this research work. He is the guest editor of two special issues on emerging nanoelectronic technlogies and their applications in IEEE Transactions on VLSI Sytstems (December 1997) and the Proceedings of the IEEE (1998). He is also an associate editor of the IEEE Transactions on VLSI and an editor of IEEE VLSI (TC) news letters.

He is a member of Sigma Xi, Phi Kappa Phi, and ACM SIGDA.