Yield and Reliability-Enhancement Techniques for Random Access Memories

Jin-Fu Li
Advanced Reliable Systems (ARES) Lab.
Department of Electrical Engineering
National Central University
Jhongli, Taiwan

Outline

- Background
- Built-In Self-Repair (BISR) Techniques for RAMs
- Error Correction Code and Transparent Test Techniques for RAMs
- Conclusions

Outline

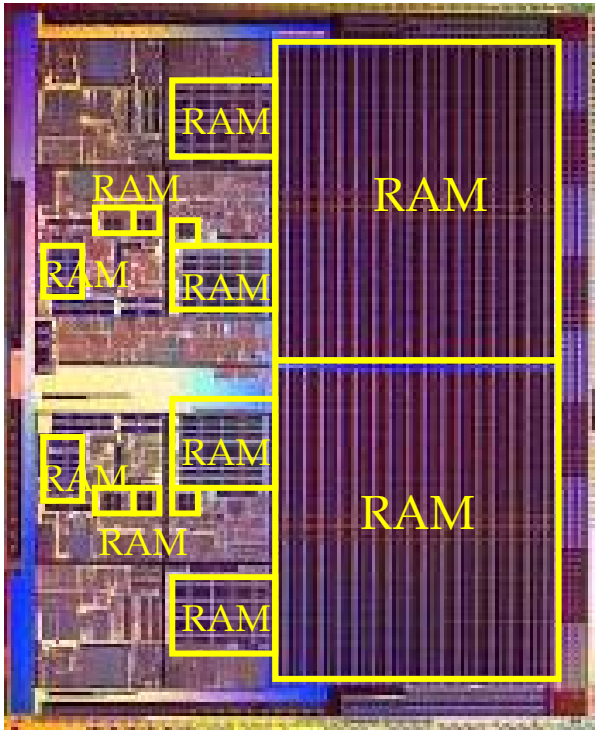
- Introduction
- Built-In Self-Repair (BISR) Techniques for RAMs
- Error Correction Code and Transparent Test Techniques for RAMs
- Conclusions

Embedded Memory–Quality

- During manufacture
  - Yield
  - Exponential yield model
  - $Y = e^{-\sqrt{AD}}$, where A and D denote the area and defect density, respectively
- After manufacture
  - Reliability
- During use
  - Soft error rate

An Explosion in Embedded Memories

- Hundreds of memory cores in a complex chip is common
- Memory cores usually represent a significant portion of the chip area

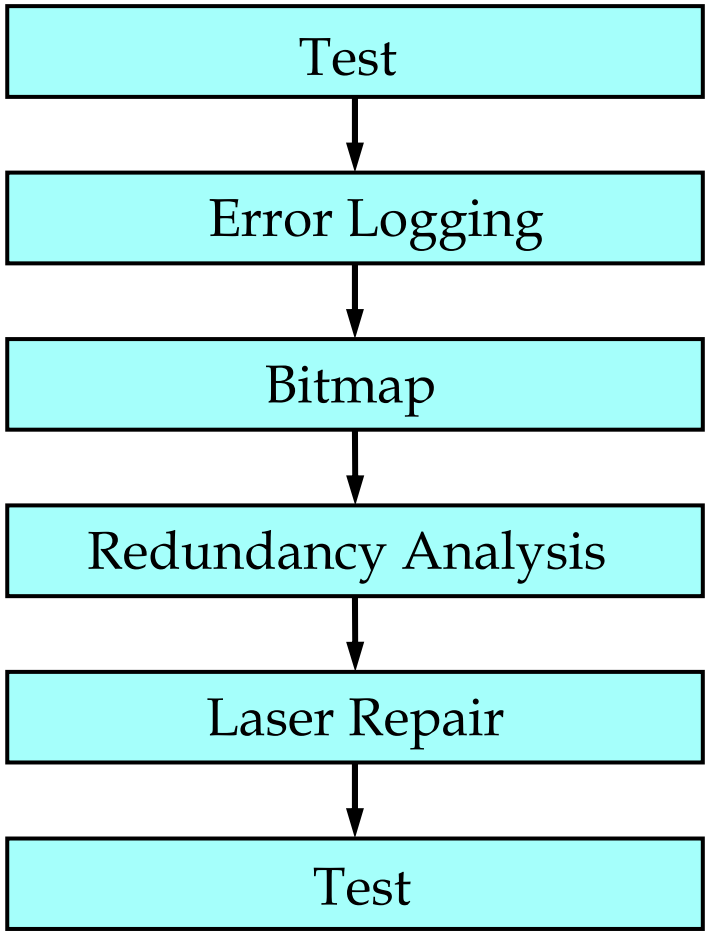


Dual-core AMD Opteron™ processor

Memory Repair

- Repair is one popular technique for memory yield improvement
- Memory repair consists of three basic steps
  - Test
  - Redundancy analysis
  - Repair delivery

Conventional Memory Repair Flow



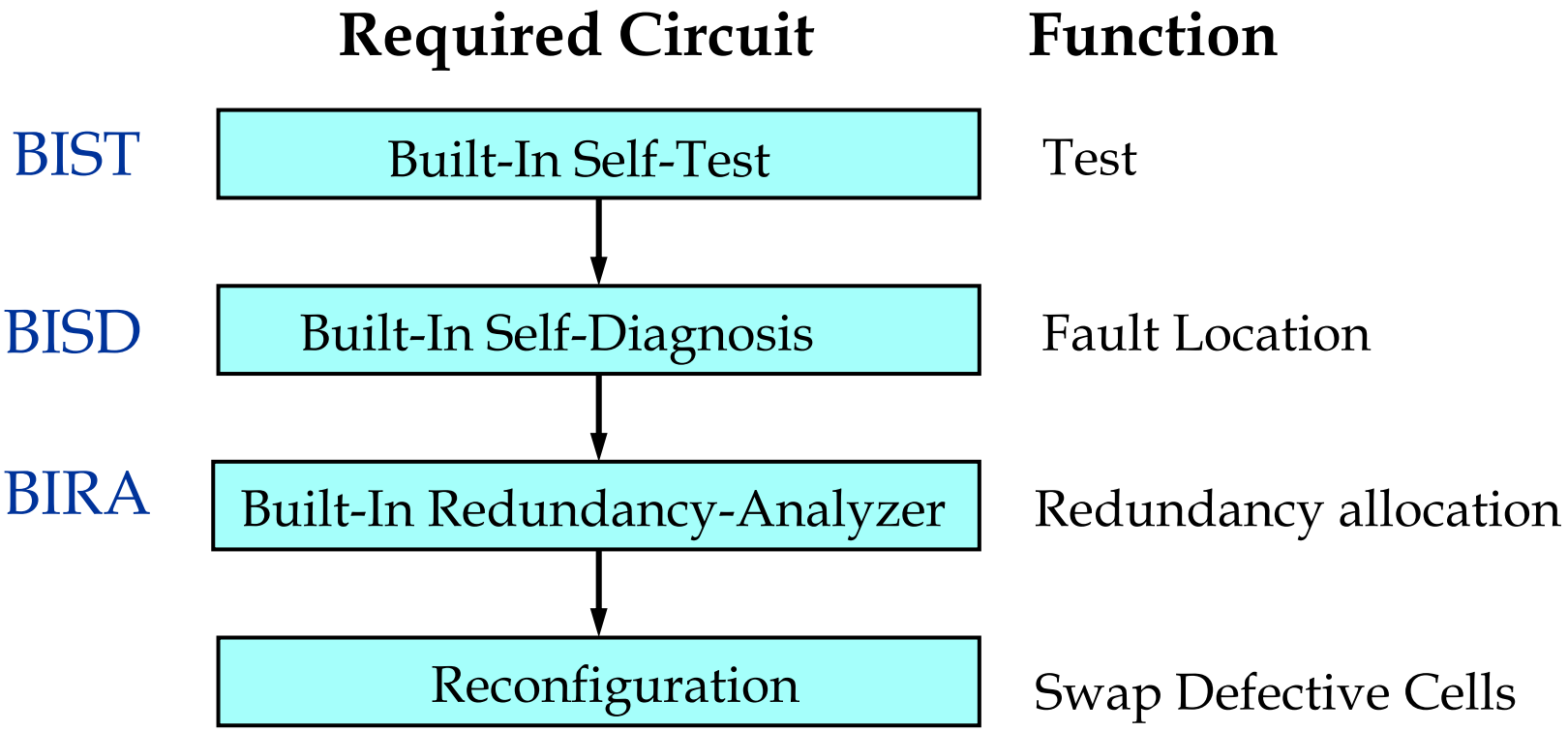
Requirements:

1. Memory tester
2. Laser repair equipment

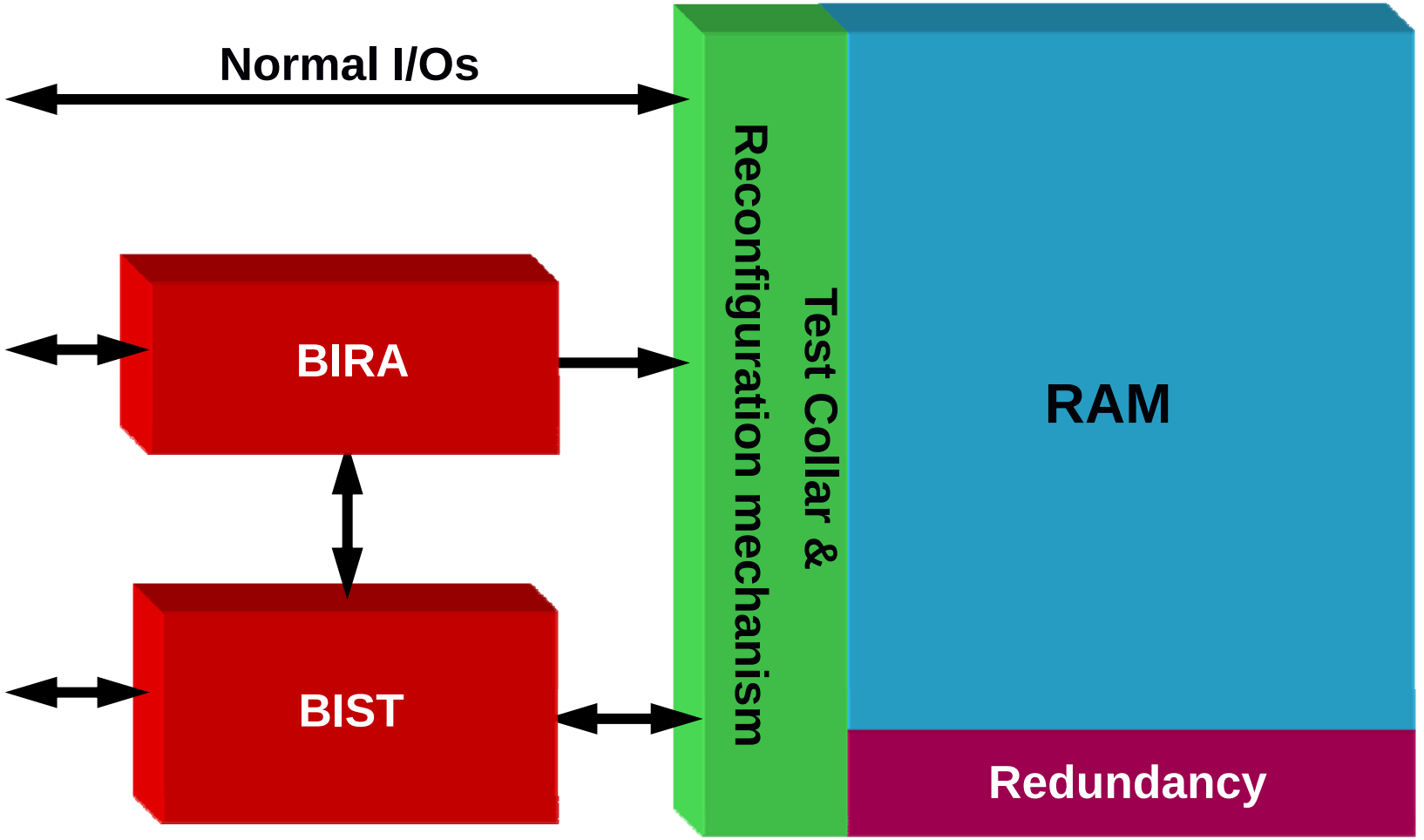
Disadvantages:

1. Time consuming
2. Expensive

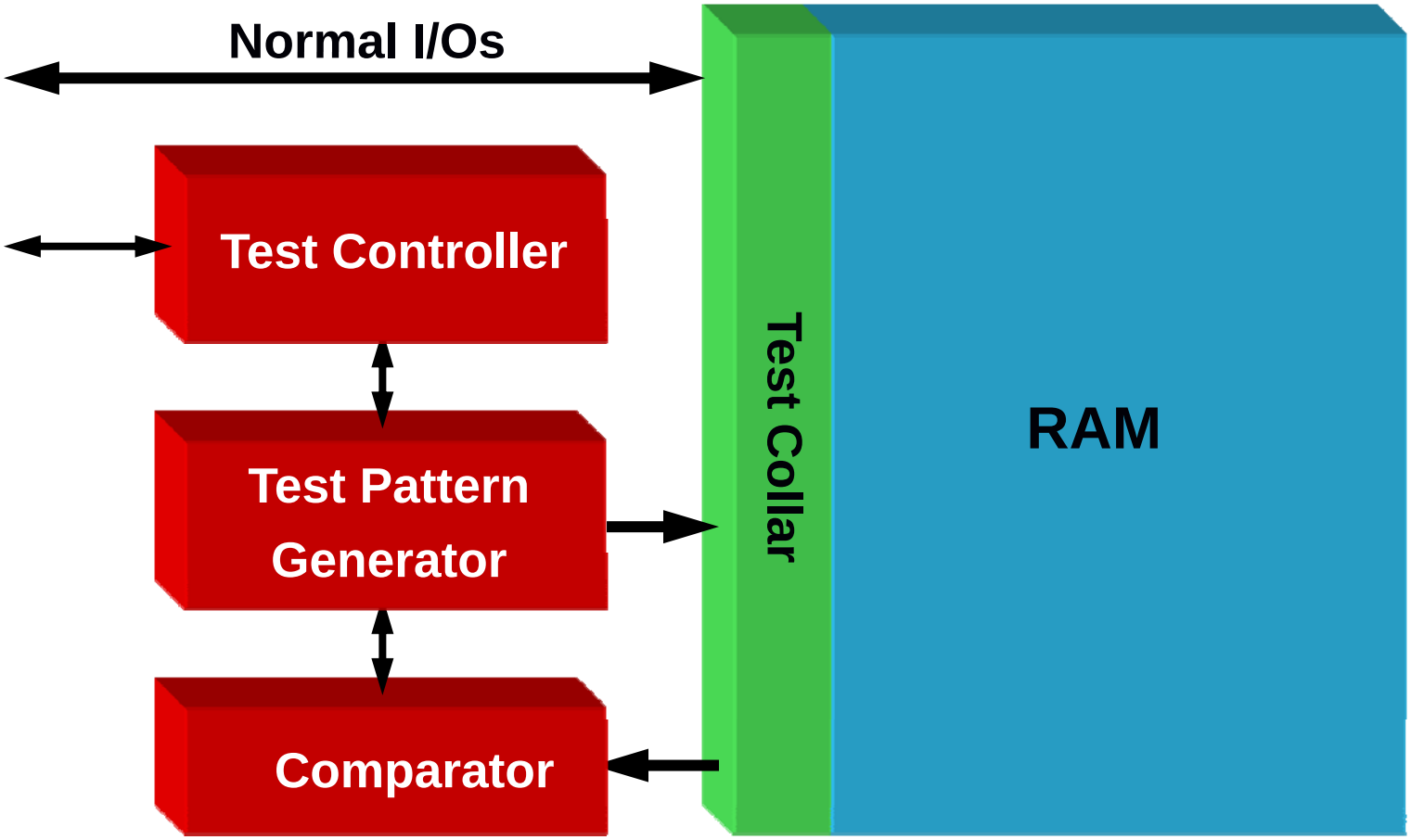
Memory BISR Flow



Typical Memory BISR Architecture

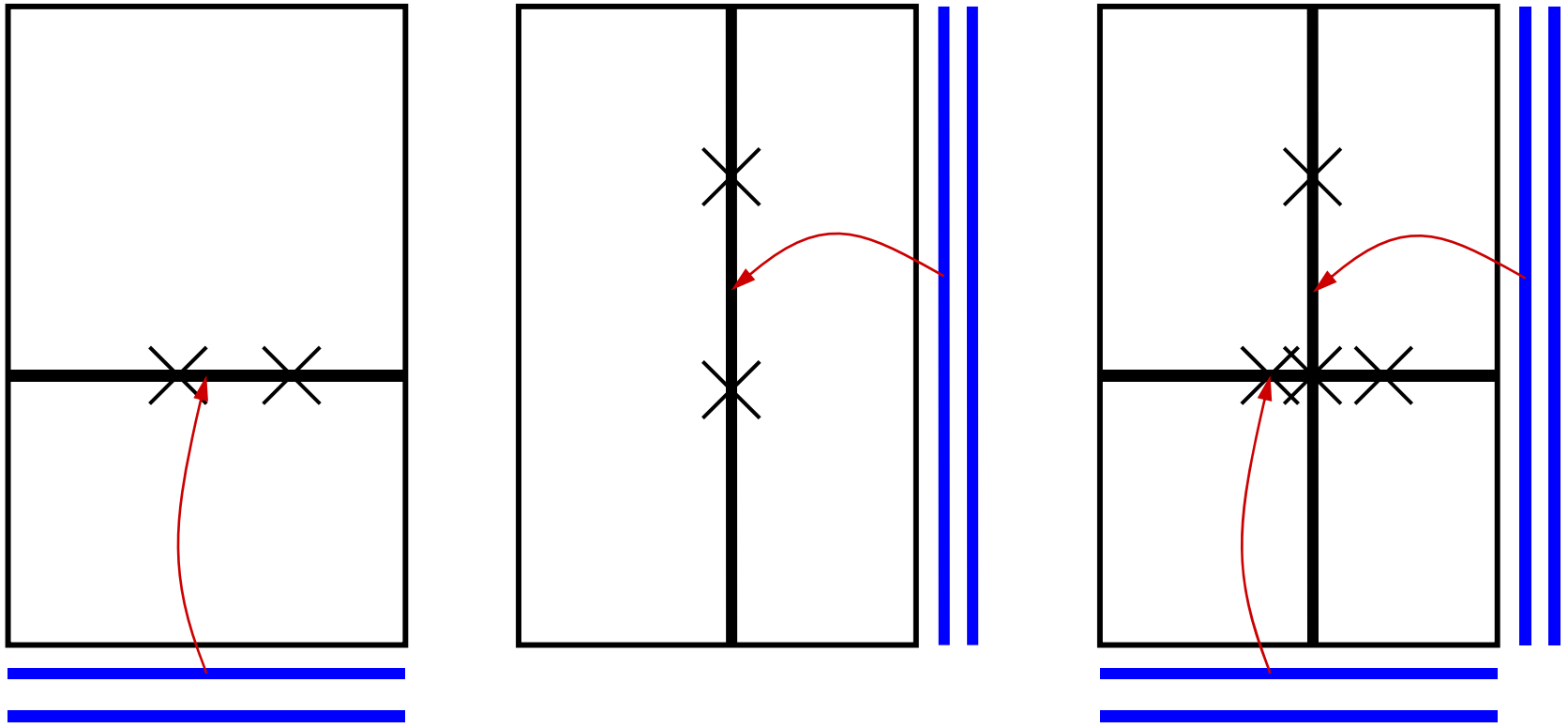


Typical Memory BIST Architecture



Redundancy Scheme

- Three typical redundancy schemes

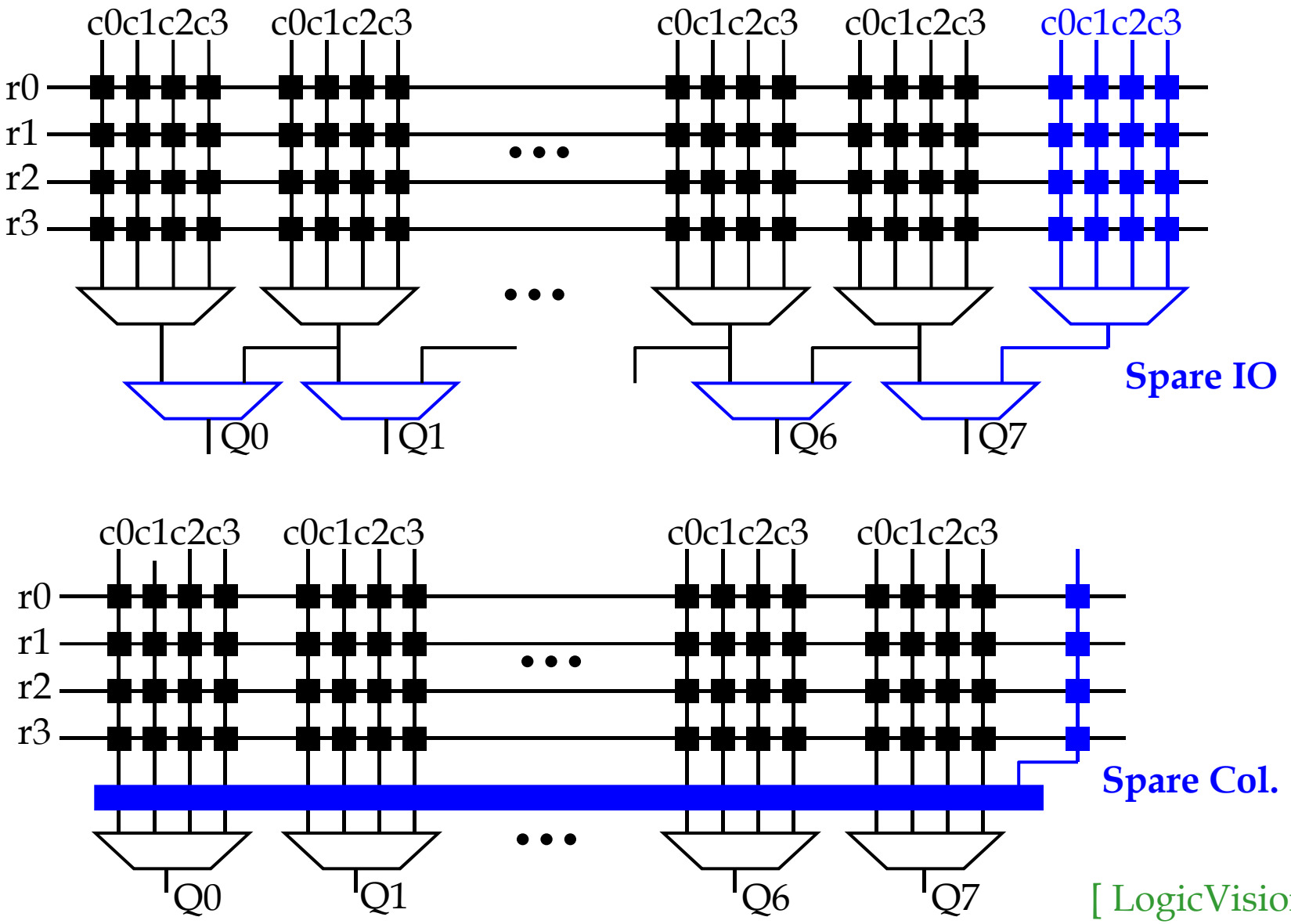


Spare rows

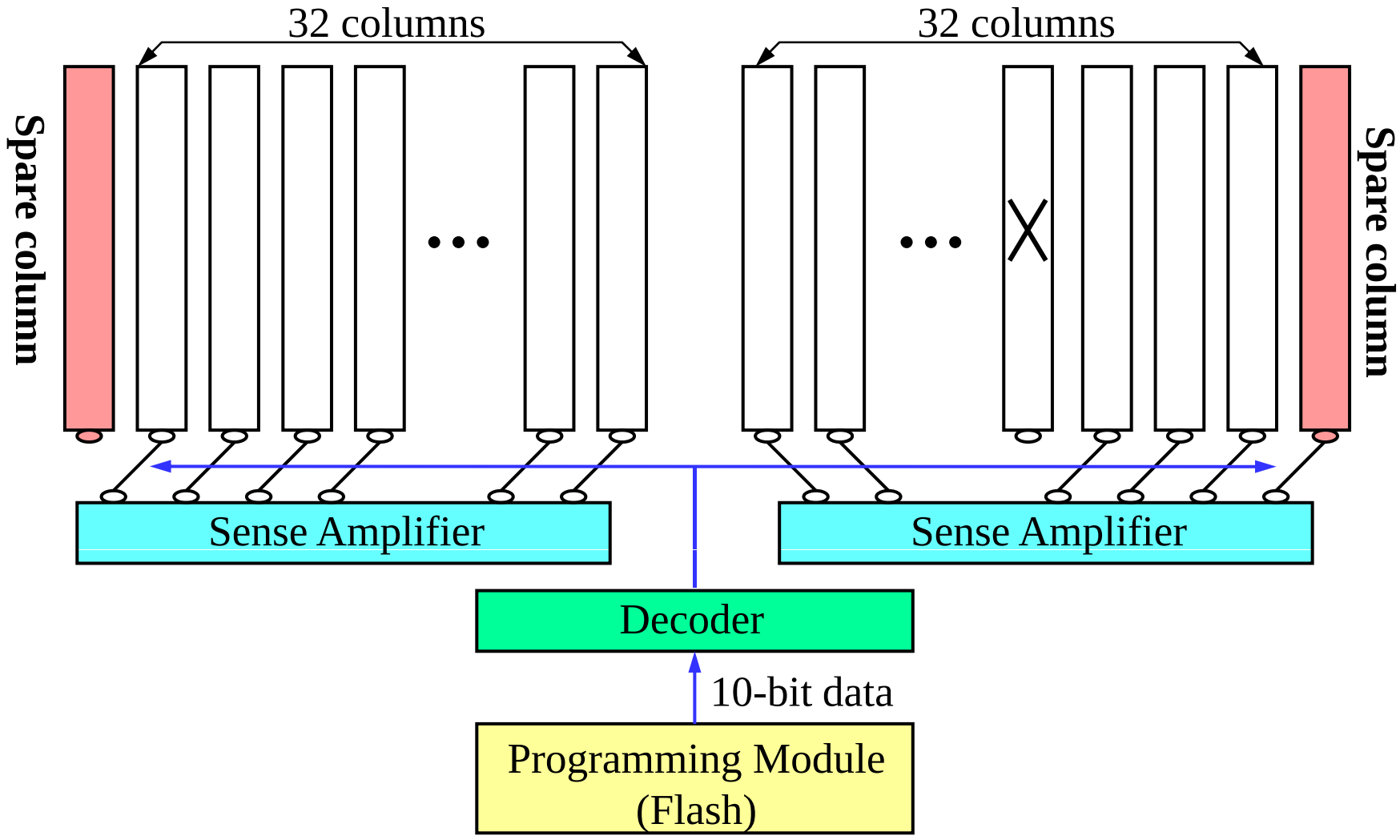
Spare columns

Spare rows and
Spare columns

Spare Column & Spare IO



Reconfiguration Scheme



[M. Yarmaoka, et al., JSSC, 2002]

Types of Reconfiguration Schemes

- Three kinds of reconfiguration techniques
  - Soft reconfiguration
    - By programming FFs to store repair information
  - Firm reconfiguration
    - By programming non-volatile memories to store repair information
  - Hard (permanent) reconfiguration
    - Laser-blown or electrically-blown polysilicon or diffusion fuses

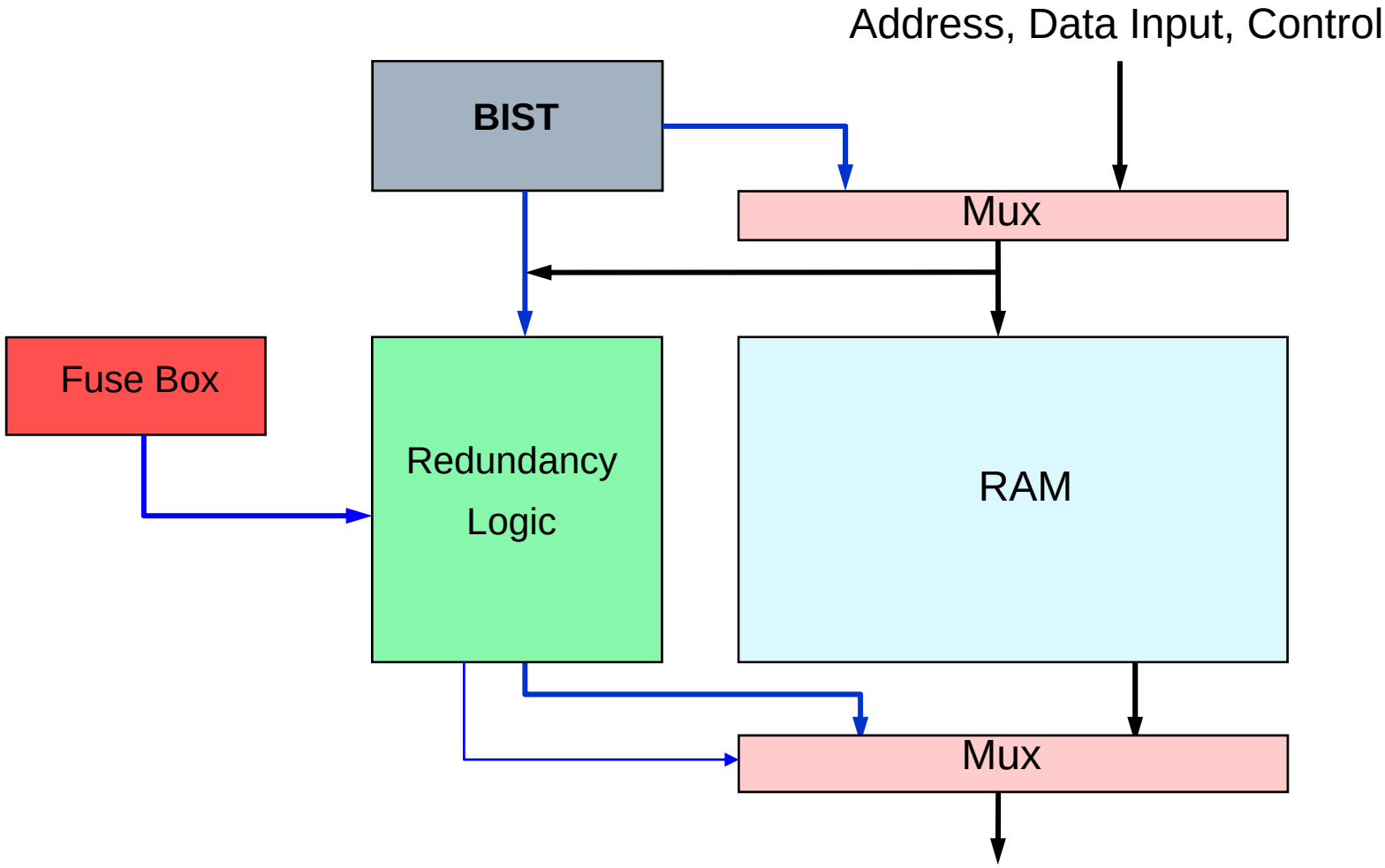
Comparison

	Advantages	Disadvantages
Soft	1. Multi-time repair 2. Low design overhead	1. Some latent defects cannot be repaired 2. Long repair setup time
Firm	1. Multi-time repair 2. Short repair setup time	1. High-voltage programming circuit is required
Hard	1. Short repair setup time	1. One-time repair 2. Specific technology is required

Memory BISR Techniques

- Dedicated BISR scheme
  - A RAM has a self-contained BISR circuit
- Shared BISR scheme
  - Multiple RAMs share a BISR circuit
  - E.g., processor-based BISR scheme and IP-based BISR scheme
- BISR classification according to the capability of redundancy analysis
  - BISR with redundancy analysis capability
  - BISR without redundancy analysis capability

RAM BISR Using Redundant Words

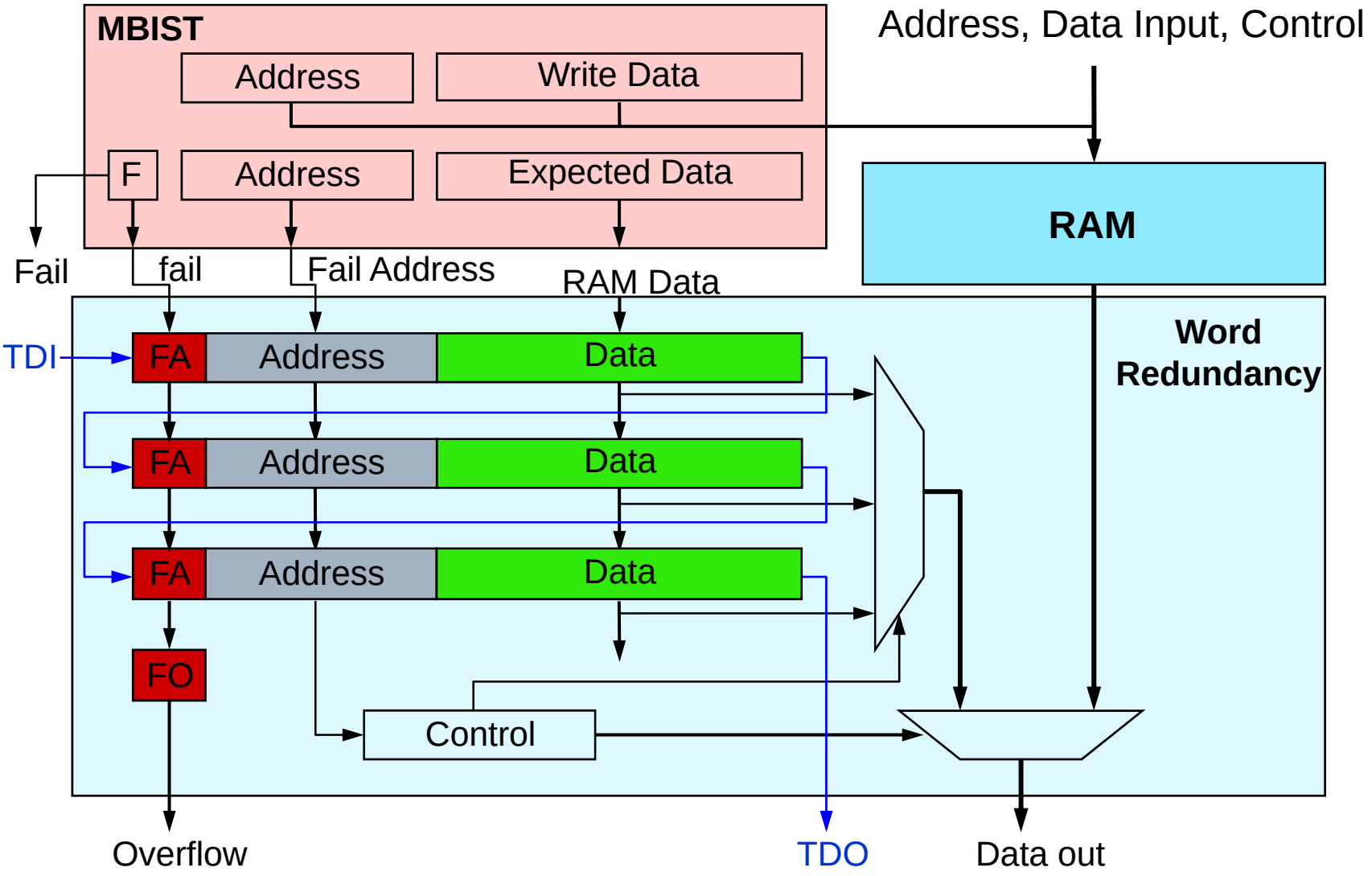


[V. Schober, et. al, ITC01]

Redundancy Wrapper Logic

- The redundancy logic consists of two basic components
  - Spare memory words
  - Logic to program the address decoding
- The address comparison is done in the redundancy logic
  - The address is compared to the addresses that are stored in the redundancy word lines
- An overflow bit identifies that there are more failing addresses than possible repair cells
- The programming of the faulty addresses is done during the memory BIST or from the fuse box during memory setup

An Array of Redundant Word Lines

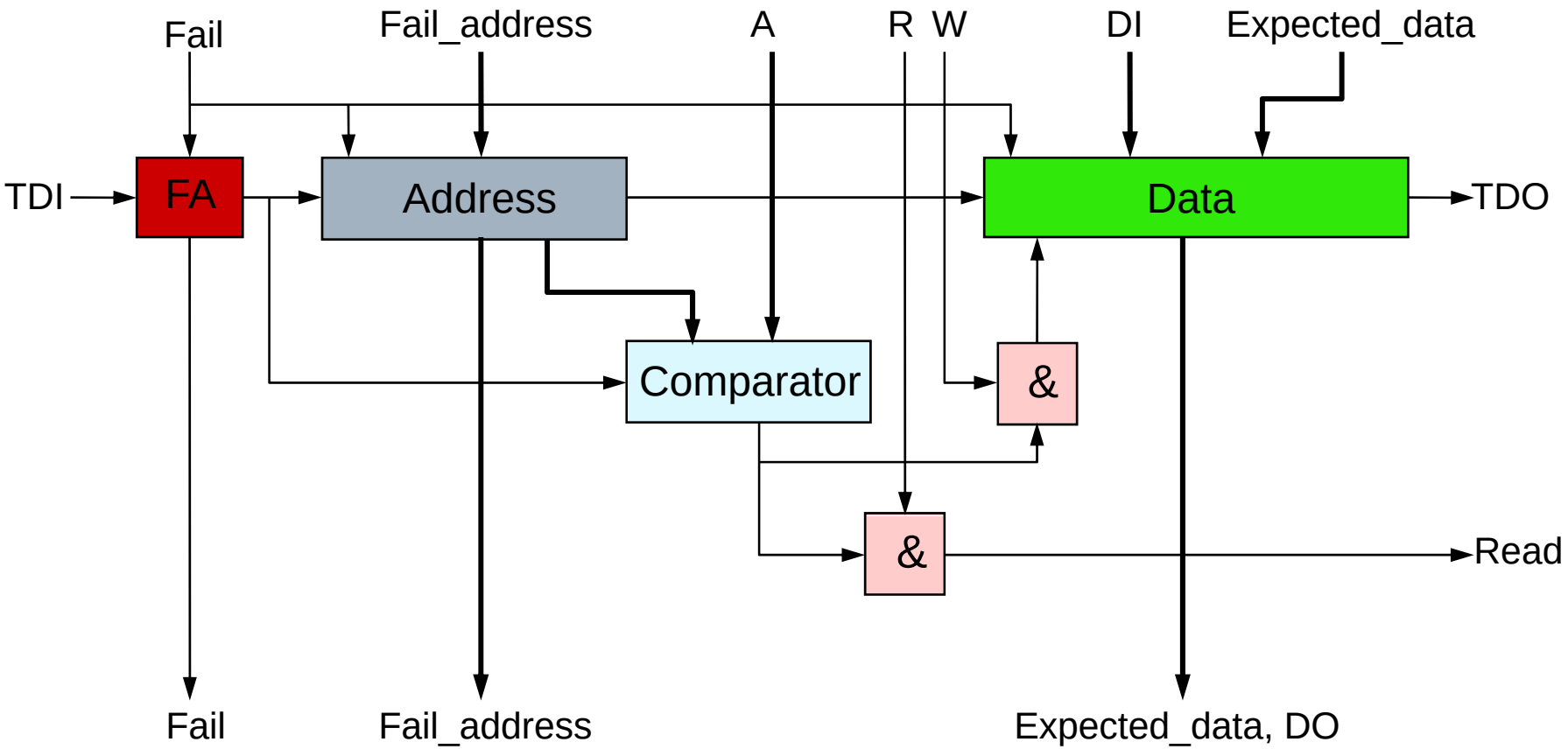


[V. Schober, et. al, ITC01]

Applications of Redundancy Logic

- Faulty addresses can be streamed out after test completion. Then the fuse box is blown accordingly in the last step of the test
  - This is called here hard repair
  - This is normally done at wafer level test
- Furthermore, the application can be started immediately after the memory BIST passes
  - This is called here soft repair

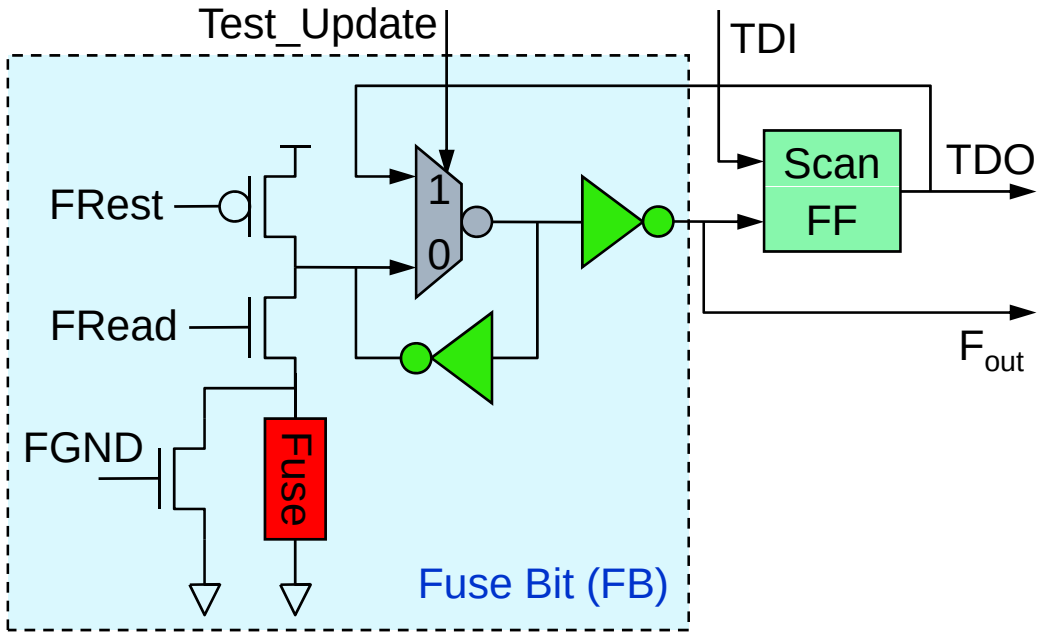
Redundancy Word Line

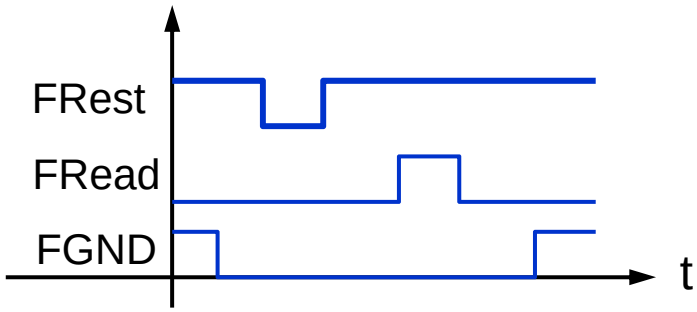


[V. Schober, et. al, ITC01]

One-Bit Fuse Box

- One-bit fuse box contains a fuse bit and a scan flip flop for controlling and observing the fuse data
  - Test_Update=0: the chain of inverters is closed (The value is latched)
  - Test_Update=1: it is possible to set the internal node from TDO
  - The ports TDI and TDO are activated at scan mode



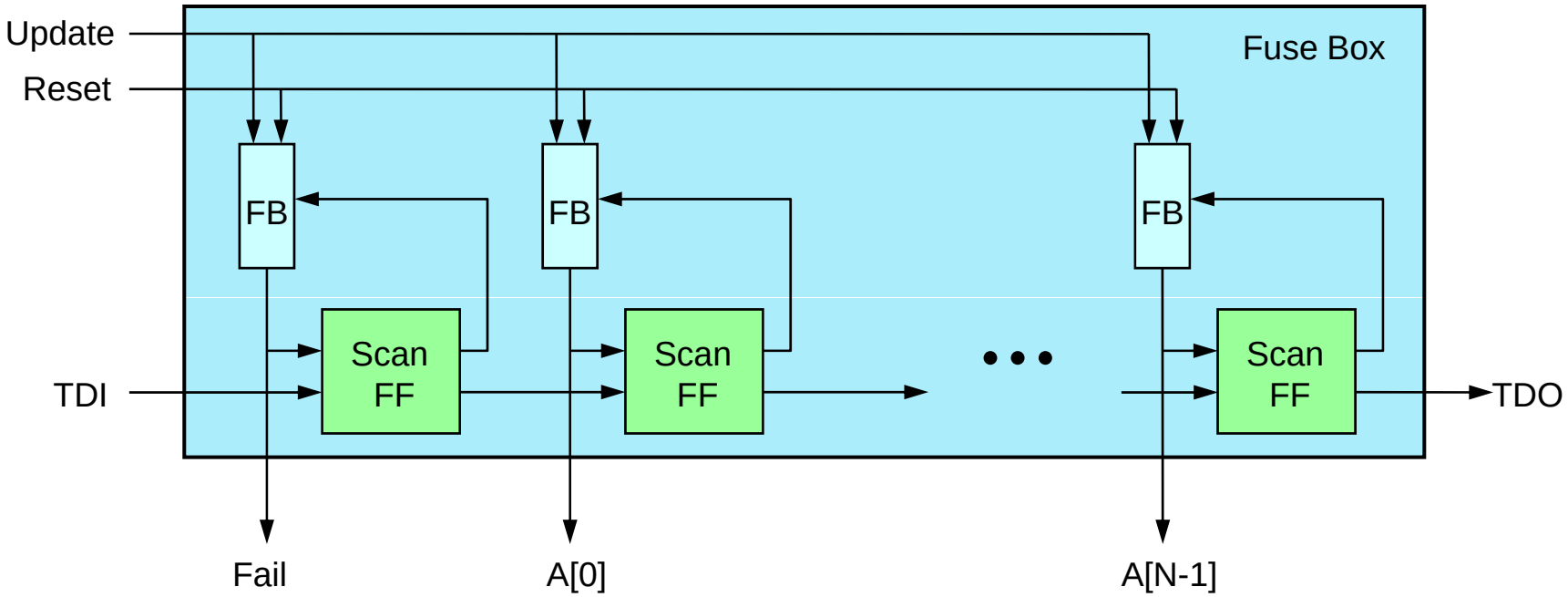


Reset cycle to read out the fuse information

[V. Schober, et. al, ITC01]

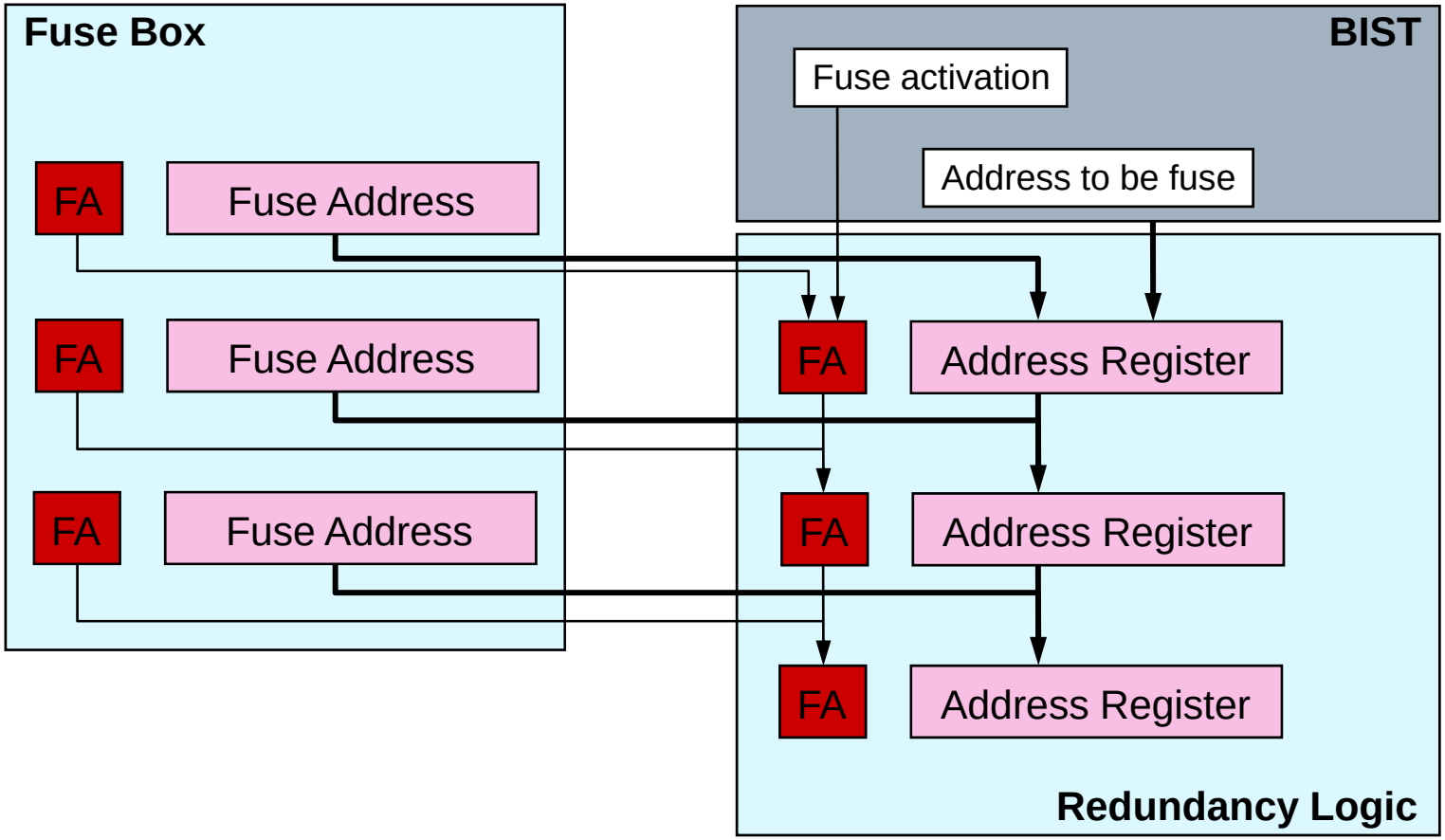
Fuse Boxes

- The fuse box can be connected to a scan register to stream in and out data during test and redundancy configuration



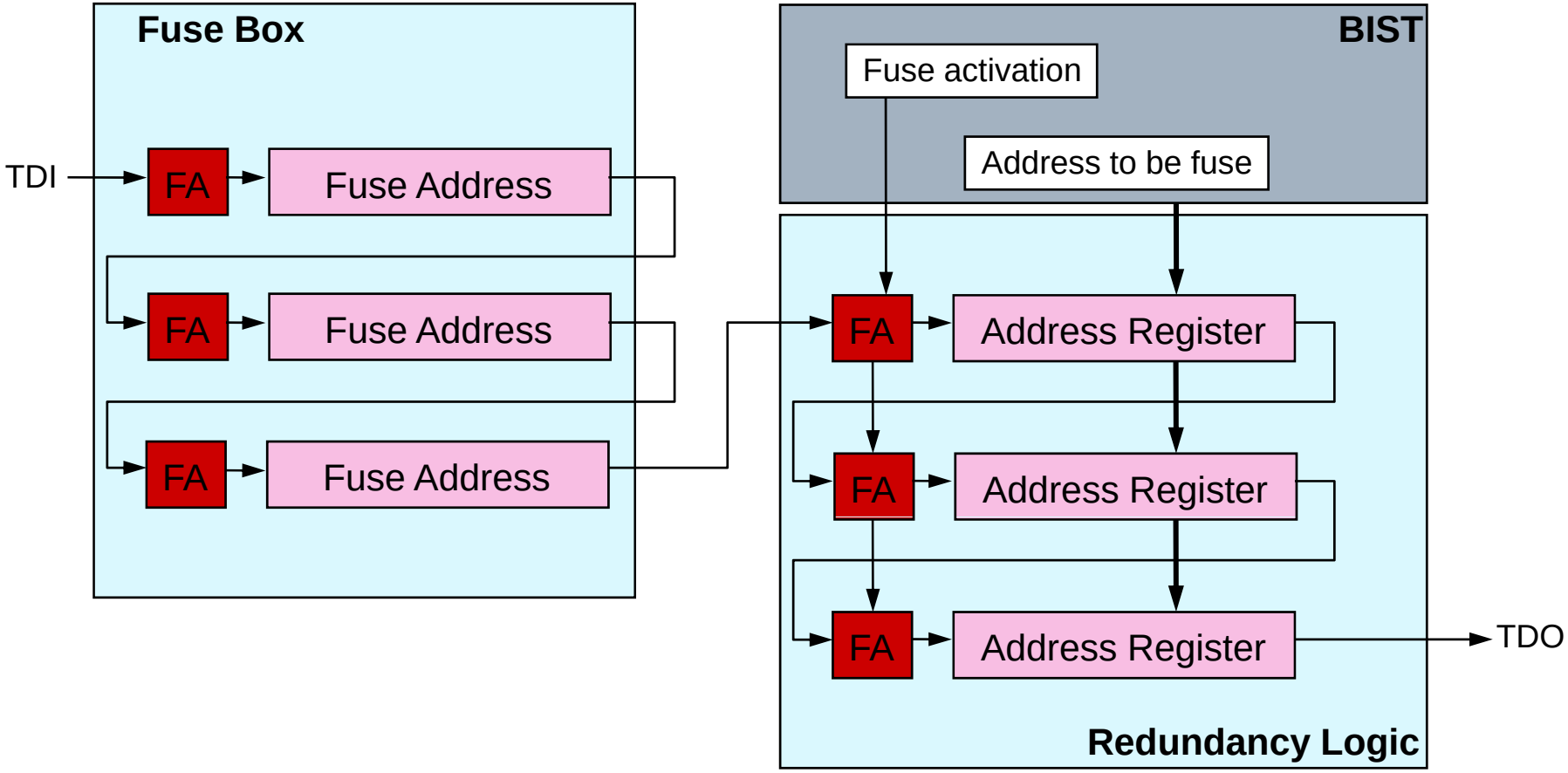
[V. Schober, et. al, ITC01]

Parallel Access of the Fuse Information



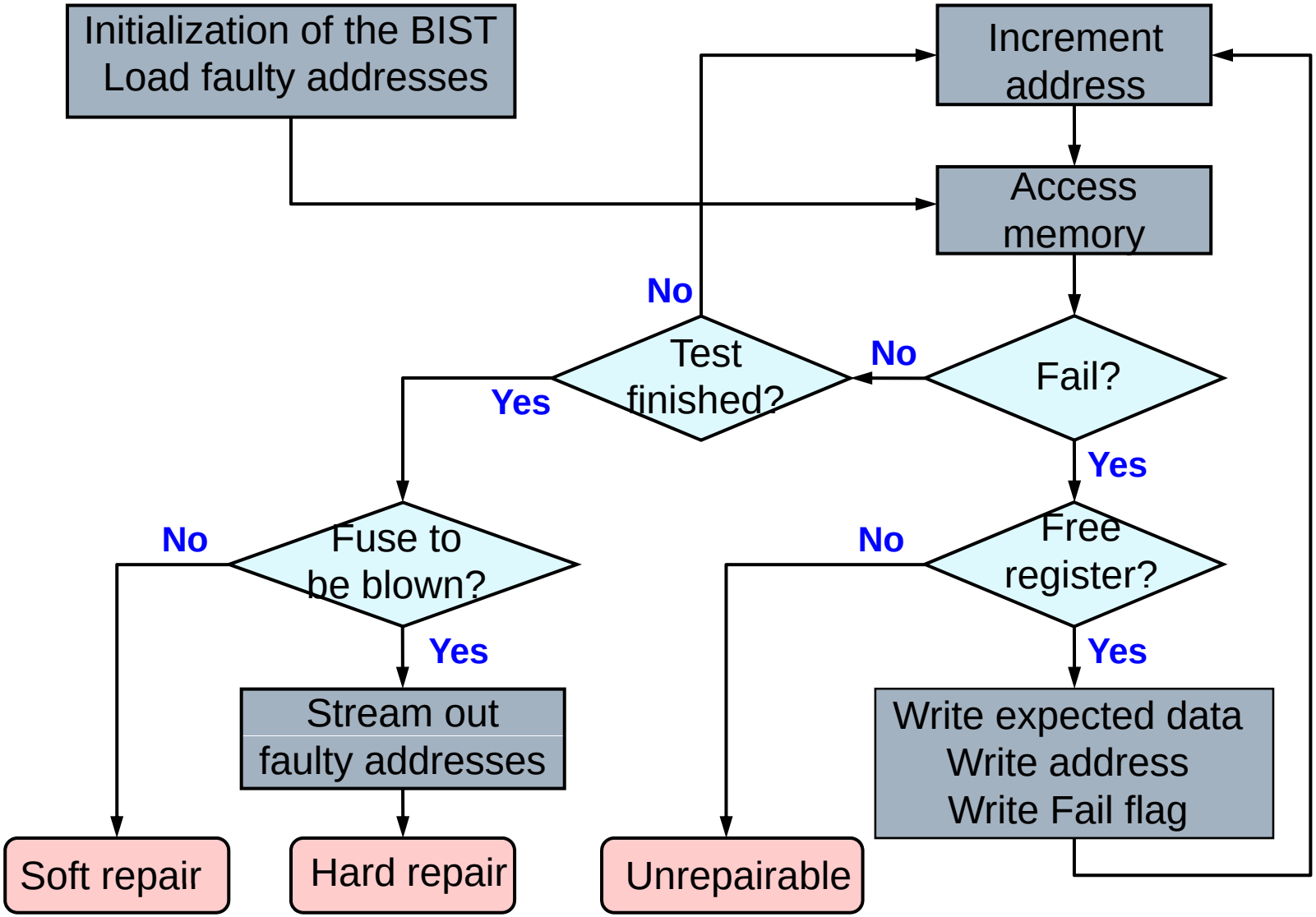
[V. Schober, et. al, ITC01]

Serial Access of the Fuse Information



[V. Schober, et. al, ITC01]

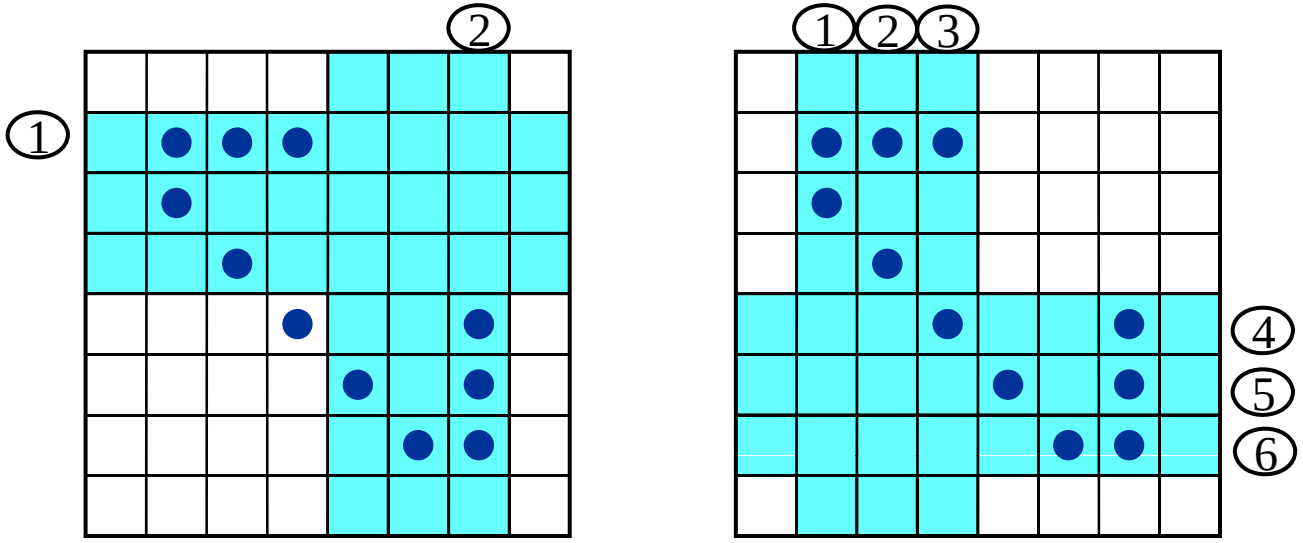
Test Flow to Activate the Redundancy



[V. Schober, et. al, ITC01]

Redundancy Analysis

- A repairable memory with 1D redundancy
  - Redundancy allocation is straightforward
- A repairable memory with 2D redundancy
  - Redundancy analysis (redundancy allocation) is needed
- Redundancy analysis problem
  - Choose the minimum number of spare rows and columns that cover all the faulty cells

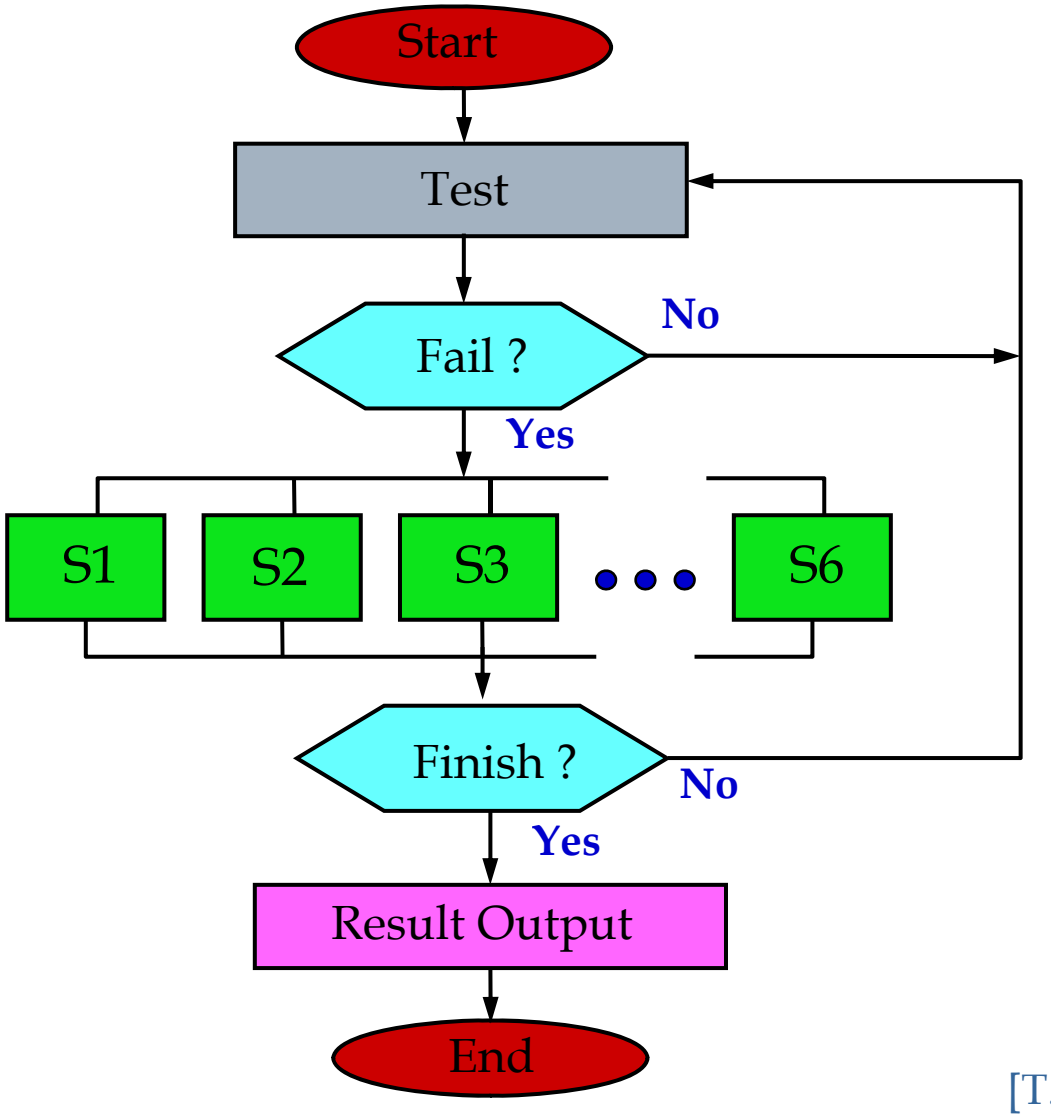


BIRA Algorithm – CRESTA

- **C**omprehensive **R**eal-time **E**xhaustive **S**earch **T**est and **A**nalysis
- Assume that a memory has 2 spare rows (Rs) & 2 spare columns (Cs), then all possible repair solutions
  - R-R-C-C (Solution 1)
  - R-C-R-C (Solution 2)
  - R-C-C-R (Solution 3)
  - C-R-R-C (Solution 4)
  - C-R-C-R (Solution 5)
  - C-C-R-R (Solution 6)

[T. Kawagoe , et. al, ITC00]

CRESTA Flow Chart



[T. Kawagoe , et. al, ITC00]

Heuristic BIRA Algorithms

- Most of heuristic BIRA algorithms need a local bitmap for storing the information of faulty cells detected by the BIST circuit
- An example of 4×5 local bitmap

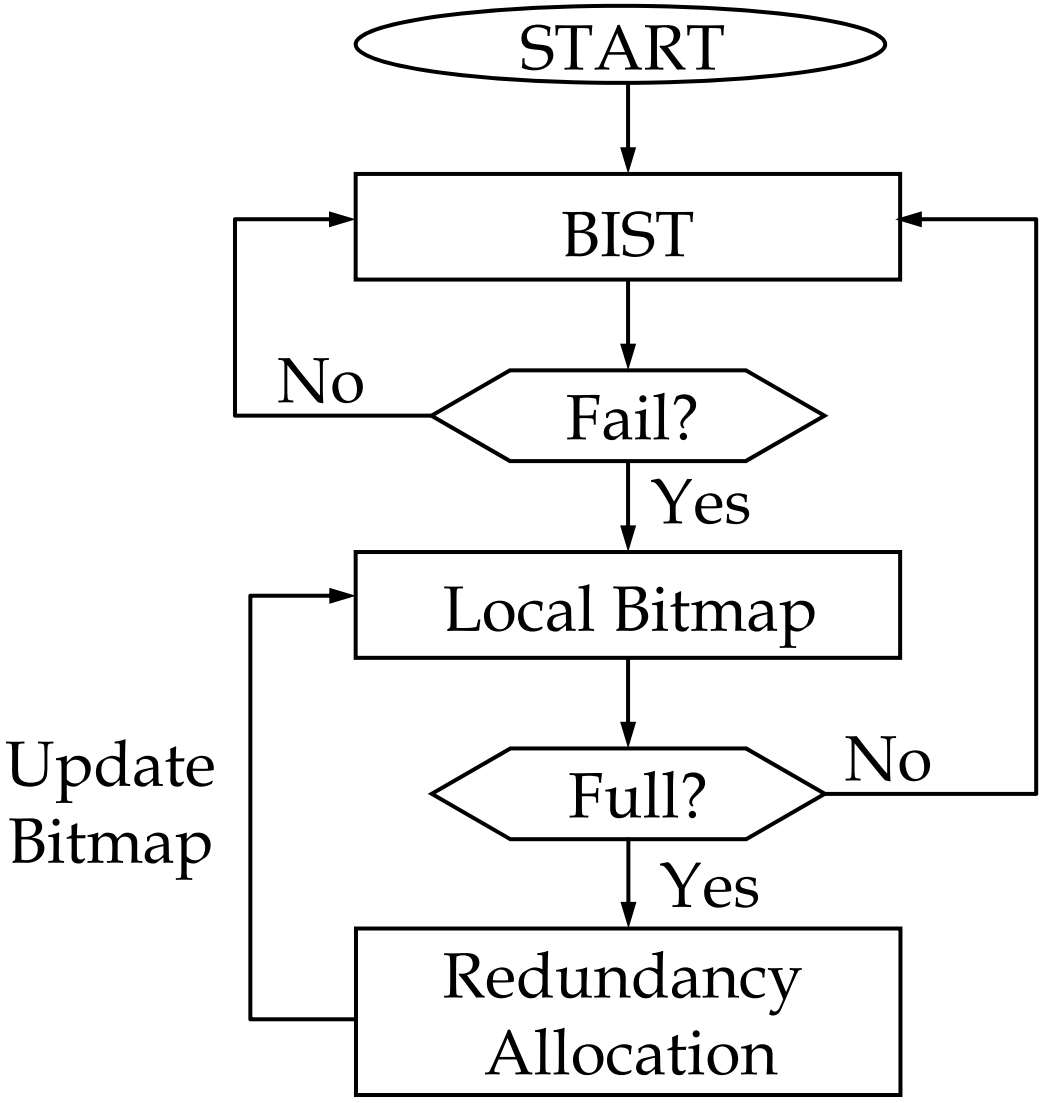
Address →

	C1	C1	C1	C1	C1
R1	1	0	0	0	0
R2	0	1	0	1	0
R3	0	0	1	0	1
R4	0	0	0	0	0

CAR

RAR

A BIRA Flow for Performing Heuristic RAs



Redundancy Allocation Rules

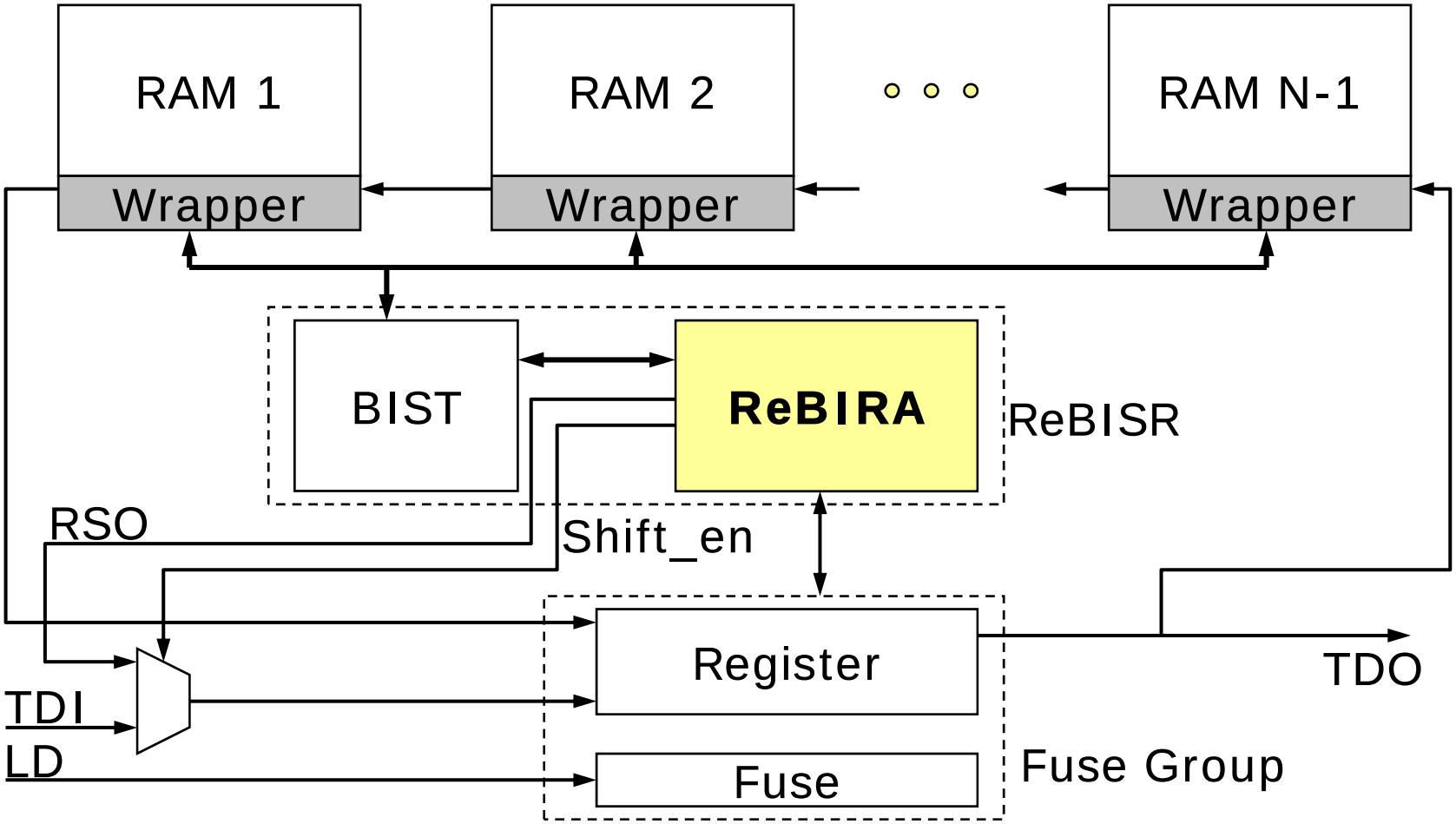
- Typical redundancy analysis algorithms
  - Two-phase redundancy allocation procedure: *must-repair phase* and *final-repair phase*
- Must-repair phase
  - Row-must repair (column-must repair): a repair solution forced by a failure pattern with $>S_C$ ($>S_R$) defective cells in a single row (column), where S_C and S_R denote the number of available spare columns and spare rows
- Final-repair phase
  - Heuristic algorithms are usually used, e.g., repair-most rule

Shared BISR Techniques

- A complex SOC usually has many RAMs with different sizes
- Each repairable RAM has a dedicated BISR circuit
  - Area cost is high
- If a BISR circuit can be shared by multiple RAMs, then the area cost of the BISR circuit can drastically be reduced
- Shared BISR techniques
  - Reconfigurable BISR or IP-based BISR technique

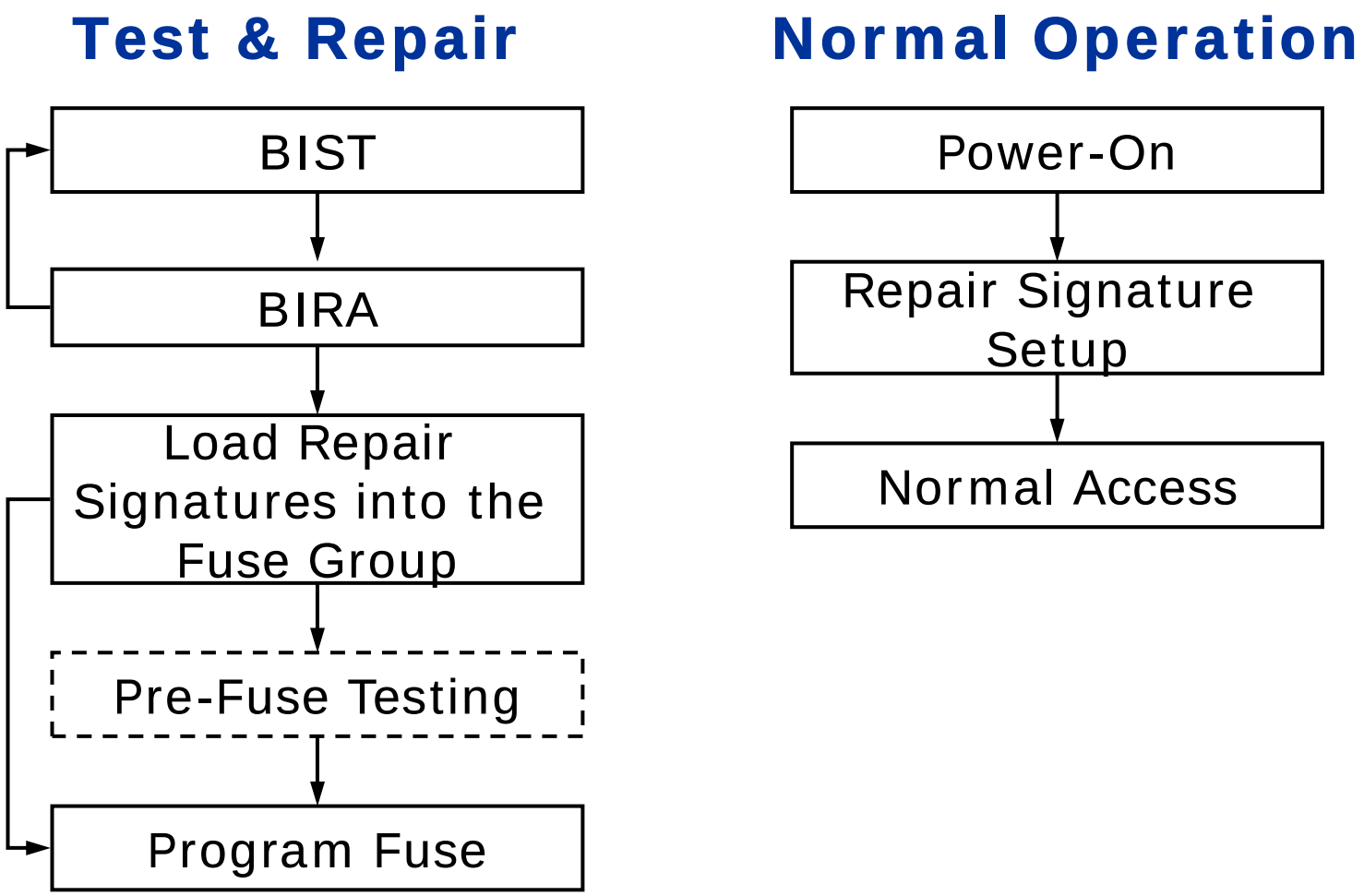
NCU/FTC BISR Scheme

- Reconfigurable BISR scheme for multiple RAMs

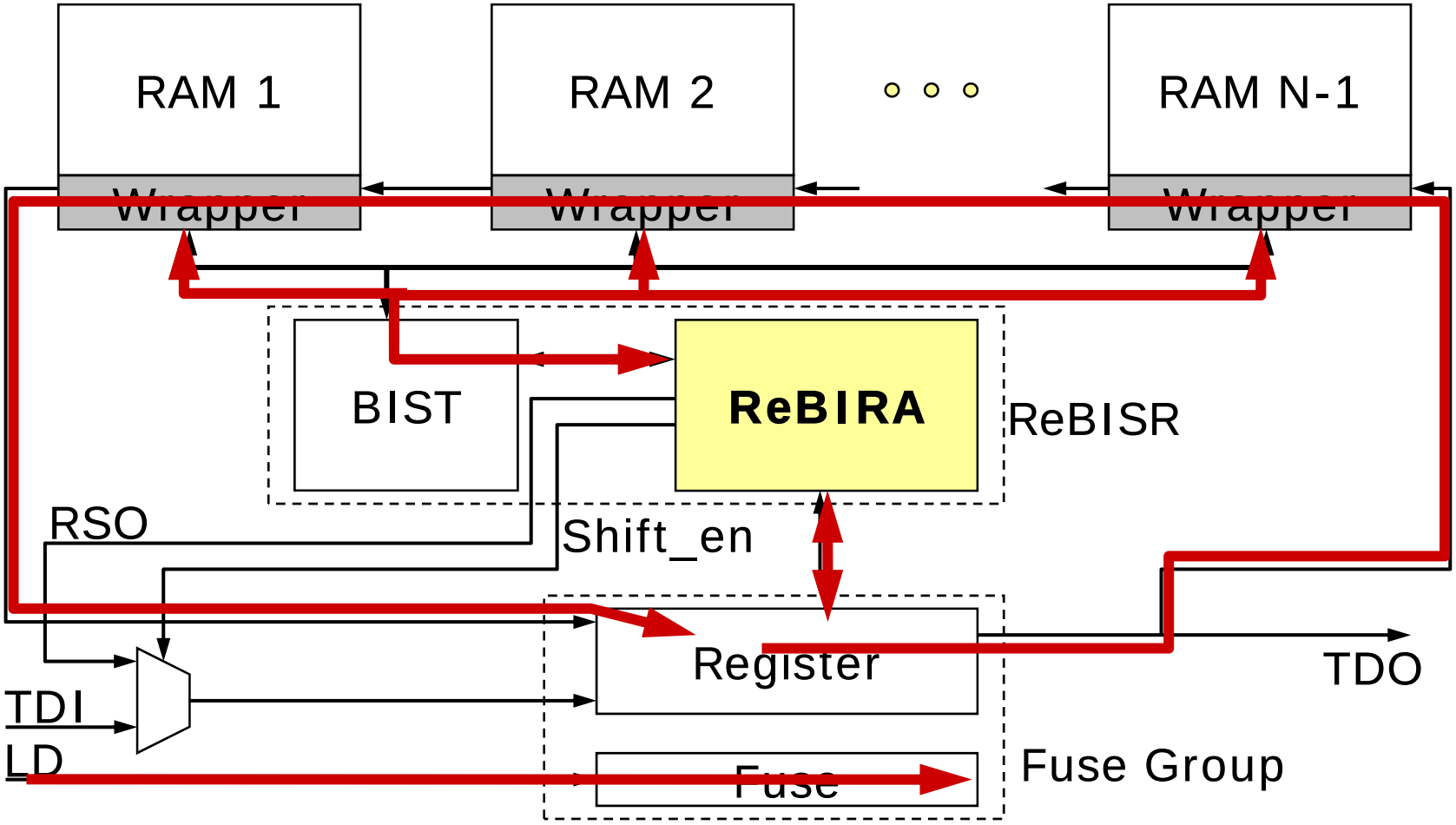


[T. W. Tseng, et. al, ITC06]

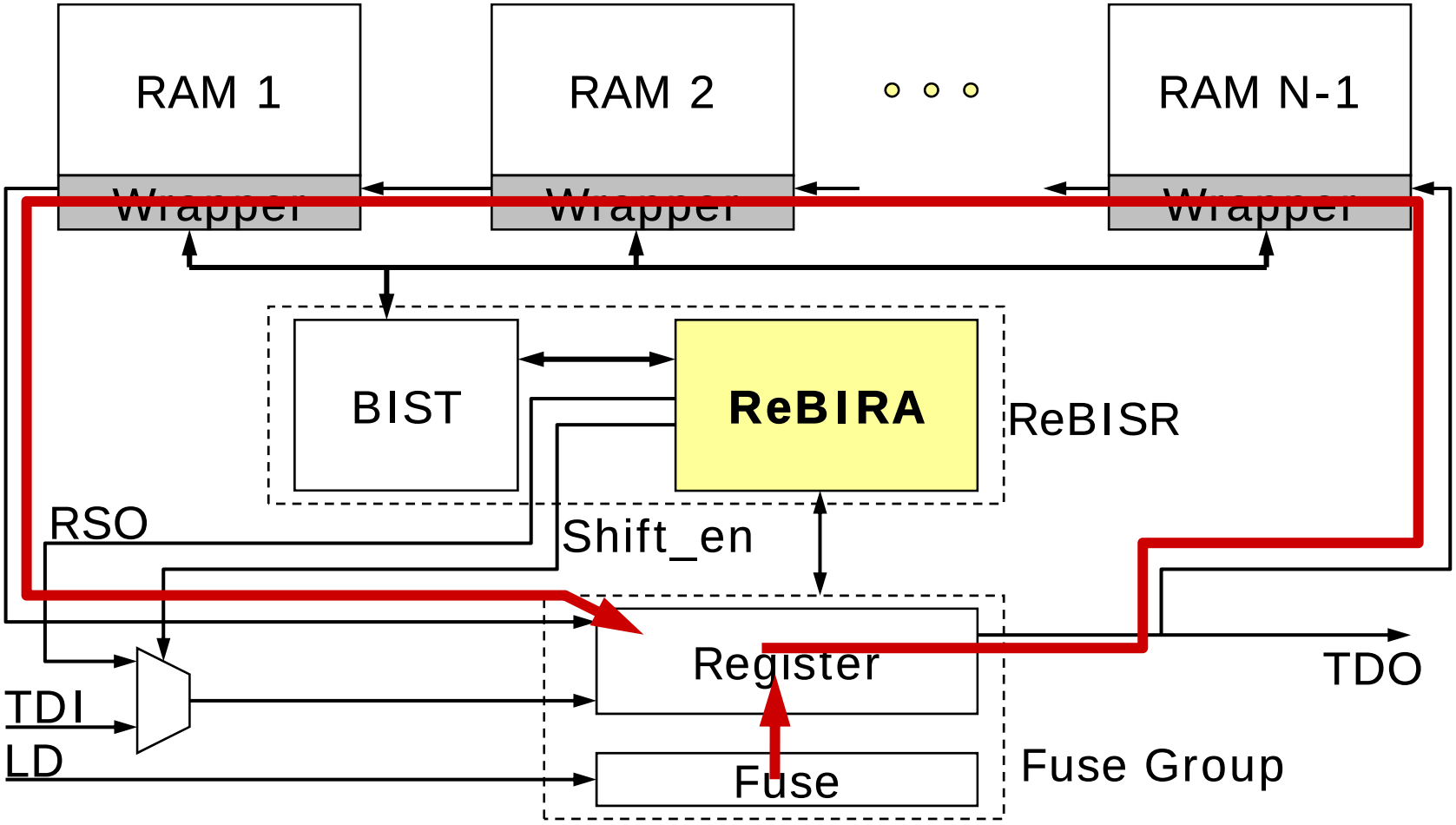
Repair Process



Test and Repair Mode

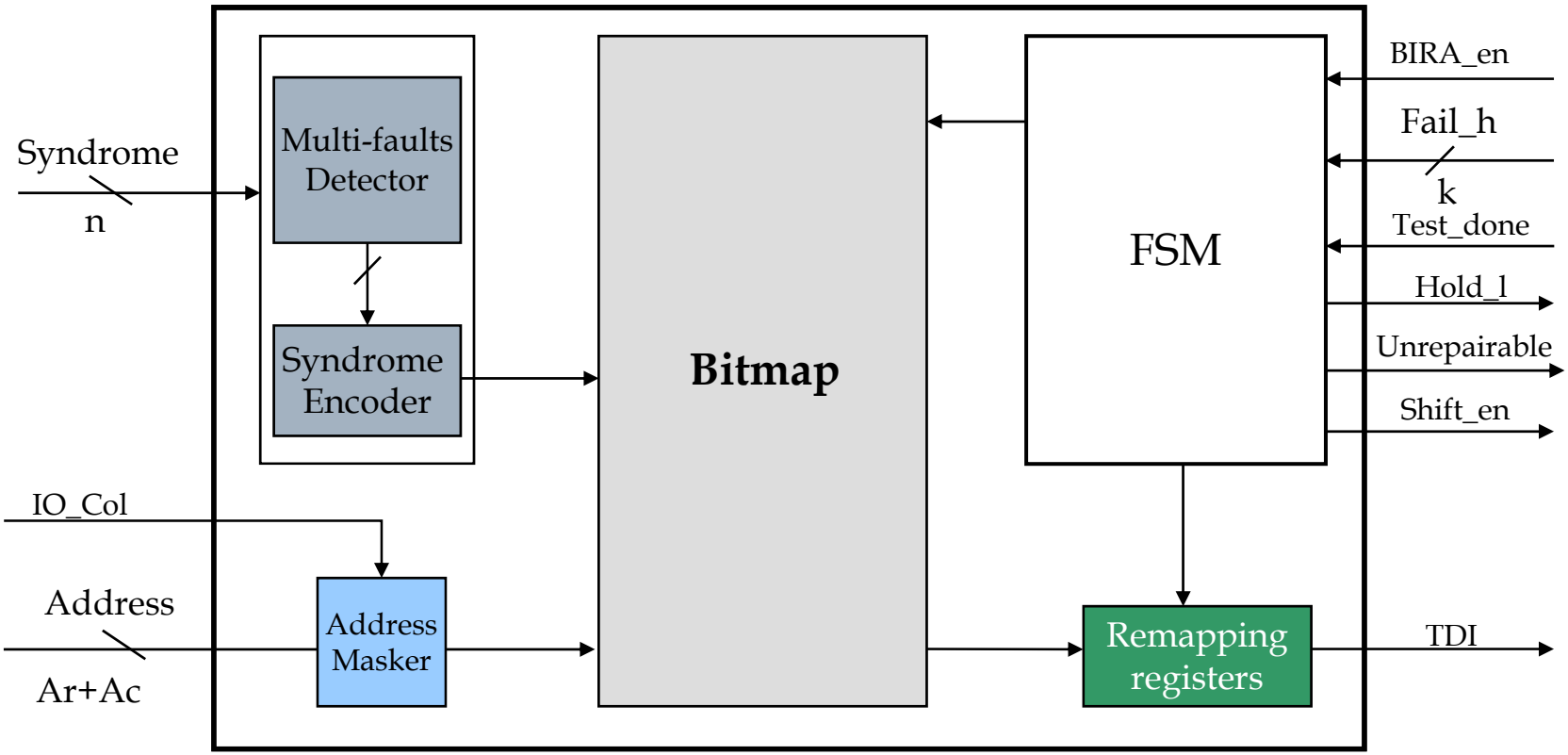


Normal Mode


RAM 1
RAM 2
RAM N-1
Wrapper
Wrapper
Wrapper
BIST
ReBIRA
ReBISR
RSO
Shift_en
TDI
LD
Register
Fuse
Fuse Group
TDO

NCU/FTC BISR Scheme

- Reconfigurable BIRA architecture



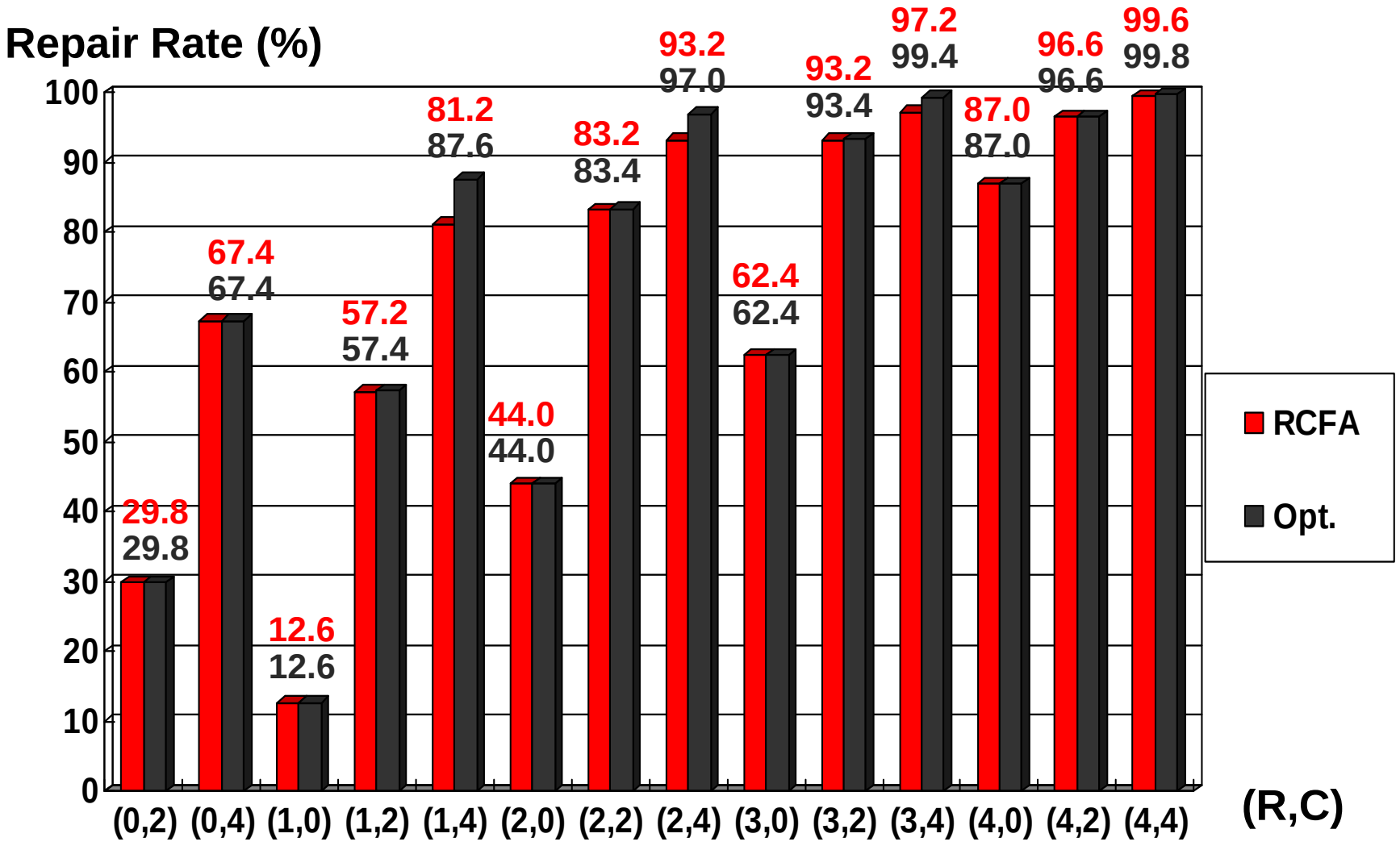
[T. W. Tseng, et. al, ITC06]

Evaluation of Repair Efficiency

- Repair rate
  - The ratio of the number of defective memories to the number of repaired memories
- A simulator was implemented to simulate the repair rate [R.-F. Huang, et. al, IEEE D&T, 2005 (accepted)
- Simulation setup
  - Simulated memory size: 4096x128
  - Simulated memory samples: 500
  - Poisson defect distribution is assumed
  - Original yield is about 60%

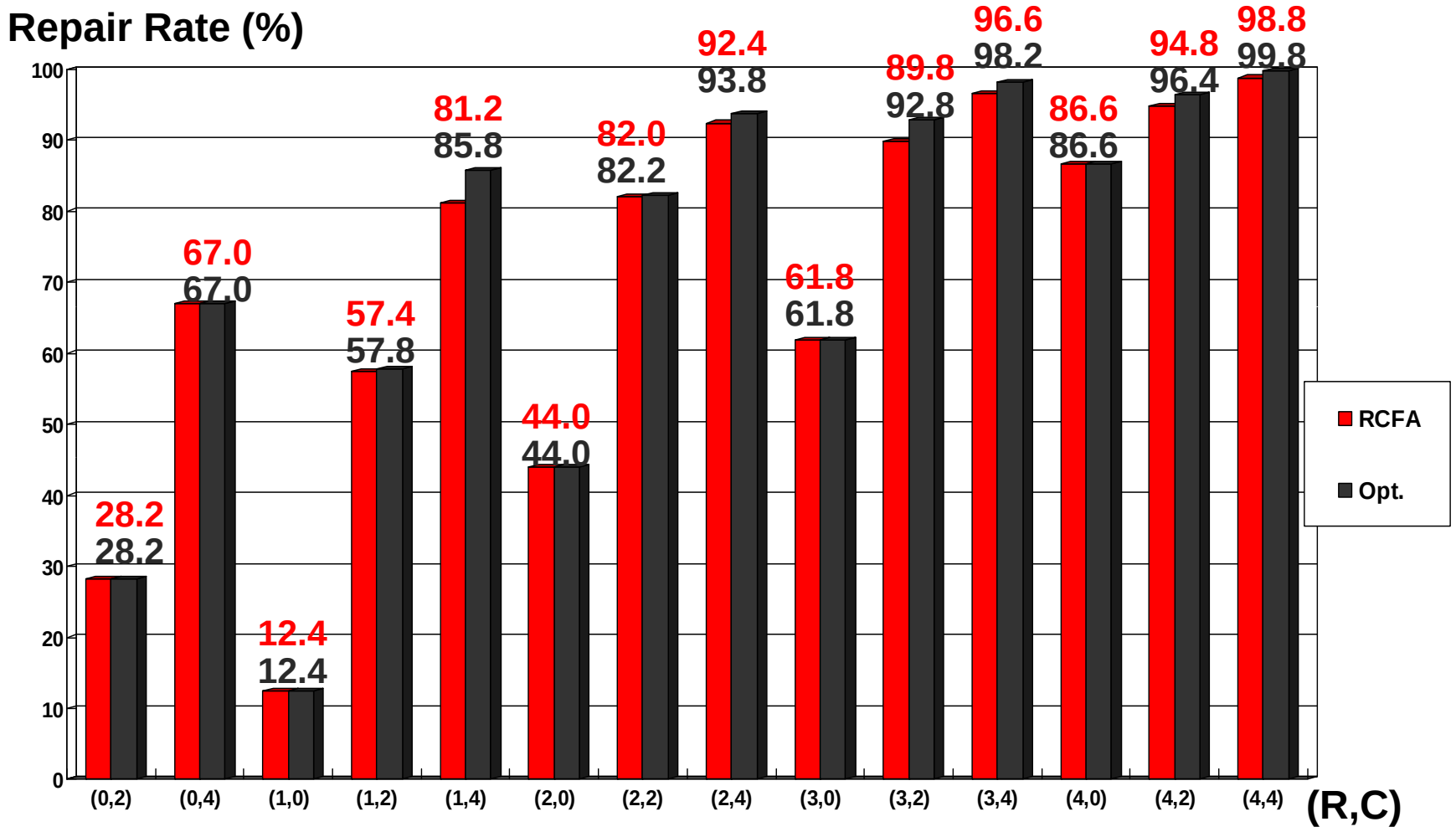
Repair Rate

- Case 1: 100% single-cell faults



Repair Rate

- Case 2: 50% single-cell faults, 20% faulty rows, 20% faulty columns, and 10% column twin-bit faults



ReBISR Implementations

- FTC 0.13*um* standard cell library is used
- Three cases are simulated

	Case 1	Case 2	Case 3
Core 0	64x2x8	64x2x16	64x2x32
Core 1	128x4x16	128x4x32	128x2x64
Core 2	256x8x32	256x8x64	256x4x128
Core 3	512x16x64	512x8x128	512x4x256

Simulation Results

- Delay and area overhead

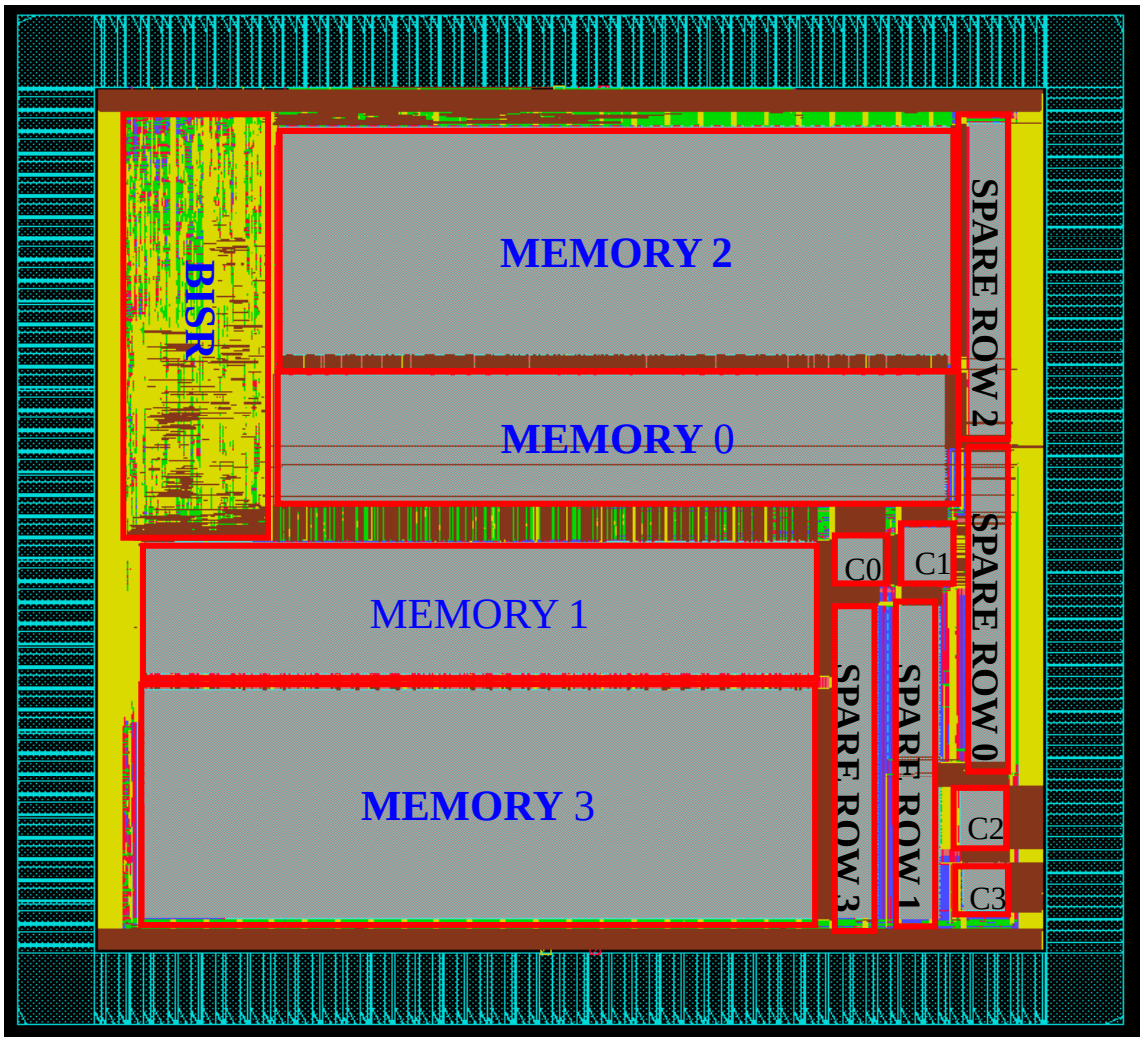
ReBIRA Parameter	Memory Area (um2)	ReBIRA Area (um2)	Ratio (%)	Delay (ns)
512x16x64	1496258.4	18766	1.25	2.5
512x8x128	1497561.6	20303	1.36	2.5
512x4x256	1528848	23255	1.52	2.5

- BIRA time overhead w.r.t. a 14N March test with solid data background

ReBIRA Parameter	Repair Rate (%)	ReBIRA Cycles	BIST Cycles	Ratio (%)
512x16x64	83.6	29952	47939584	0.06
512x8x128	82.3	30698	23605658	0.13
512x4x256	83.8	30404	12013568	0.25

NCU/FTC BISR Scheme

- Layout view for an experimental case

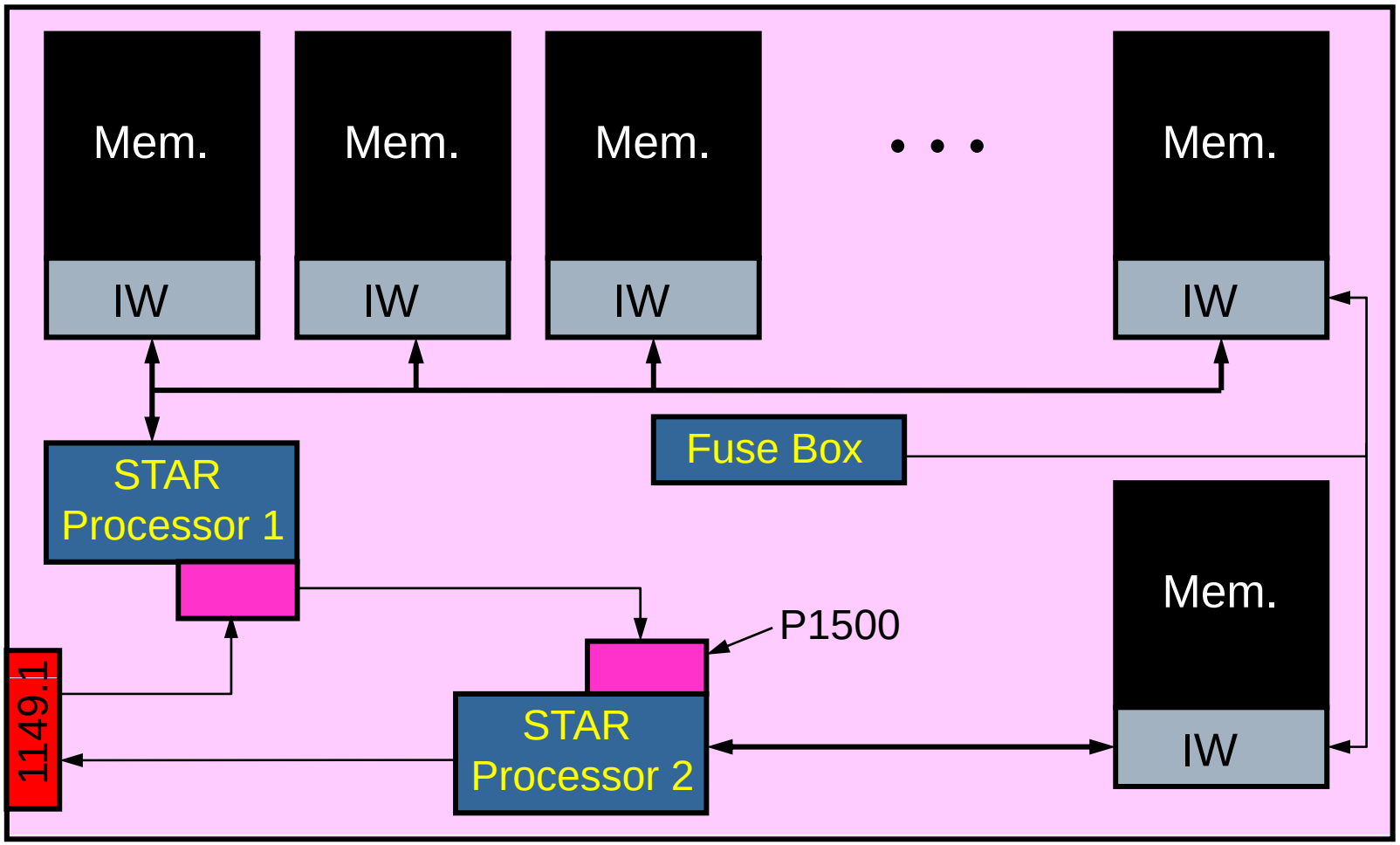


Infrastructure IP

- What is Infrastructure IP
  - Unlike the functional IP cores used in SOCs, the infrastructure IP cores do not add to the main functionality of the chip. Rather, they are intended to ensure the manufacturability of the SOC and to achieve lifetime reliability
- Examples of such infrastructure IPs
  - Process monitoring IP, test & repair IP, diagnosis IP, timing measurement IP, and fault tolerance IP

Infrastructure IP – STAR

- STAR IIP



[Y. Zorian, ITC02]

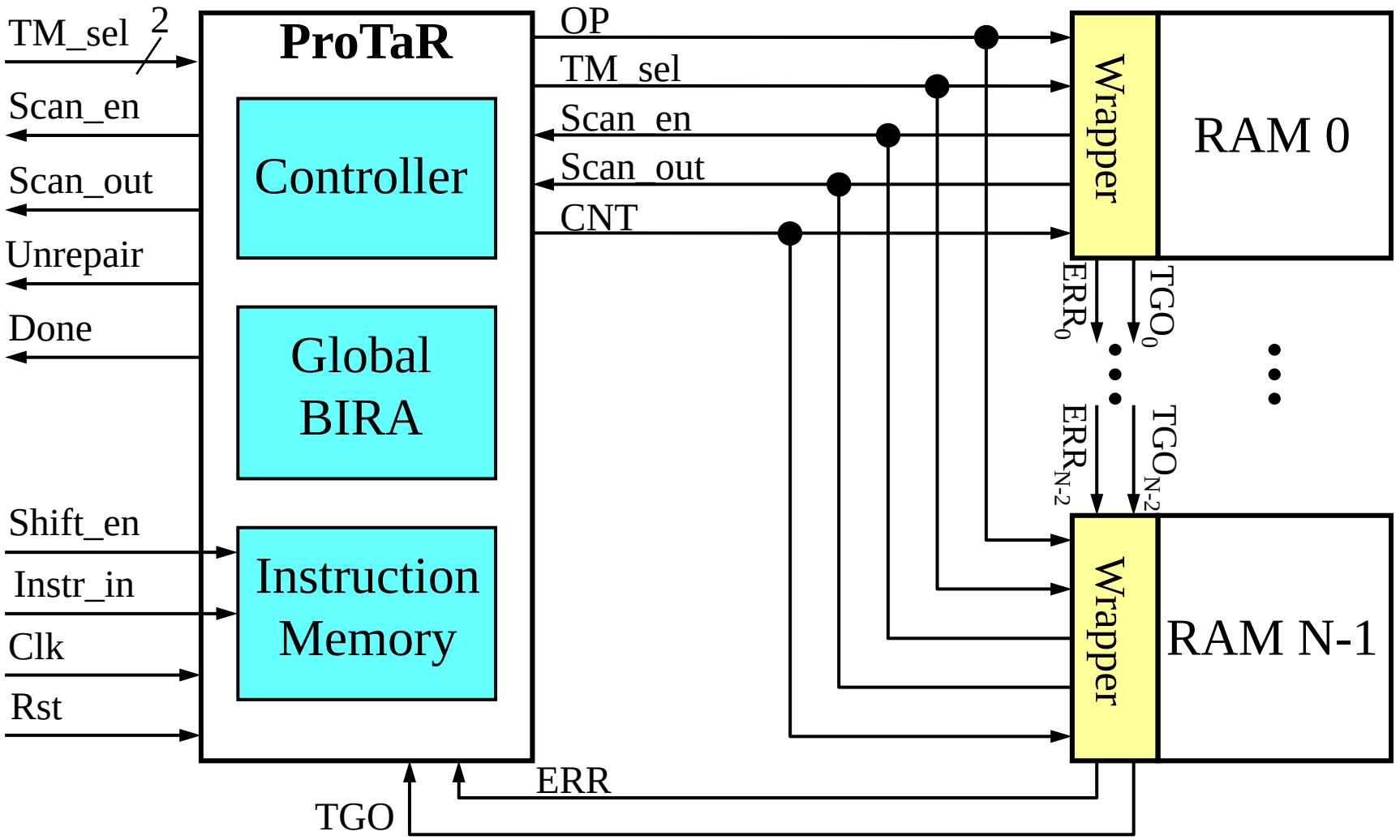
Infrastructure IP – STAR

- The infrastructure IP is comprised of a number of hardware components, including
  - A STAR processor, a fuse box, and intelligent wrappers (IWs)
- The STAR Processor
  - Performs all appropriate test & repair coordination of a STAR memory
  - It is programmed by a set of instructions to control the operation of the internal modules
- The Intelligent Wrapper
  - Address counters, registers, data comparators and multiplexers

Infrastructure IP – ProTaR

- ProTaR [C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]
  - **Pro**cessor for **T**est **a**nd **R**epair of RAMs
- The infrastructure IP is comprised of a number of hardware components, including
  - A ProTaR processor
  - A wrapper
- Features
  - Parallel test and diagnosis
  - Serial repair
  - Support multiple redundancy analysis algorithms

Architecture of the Proposed IIP



[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]

Multiple Redundancy Analysis Algorithms Support

- In the IIP, the ProTaR has one global BIRA module and each wrapper has one local BIRA module
- The local BIRA module performs the must-repair phase of a redundancy analysis algorithm
- Then, the global BIRA module performs the final-repair phase of the redundancy analysis algorithm

Global/Local Bitmaps and RA Instructions

Local bitmap

	C0	C1	C2	C3	CAR
R0	1	0	0	0	
R1	0	1	0	1	
R2	0	1	0	0	
R3	0	0	1	0	

RAR

Global bitmap

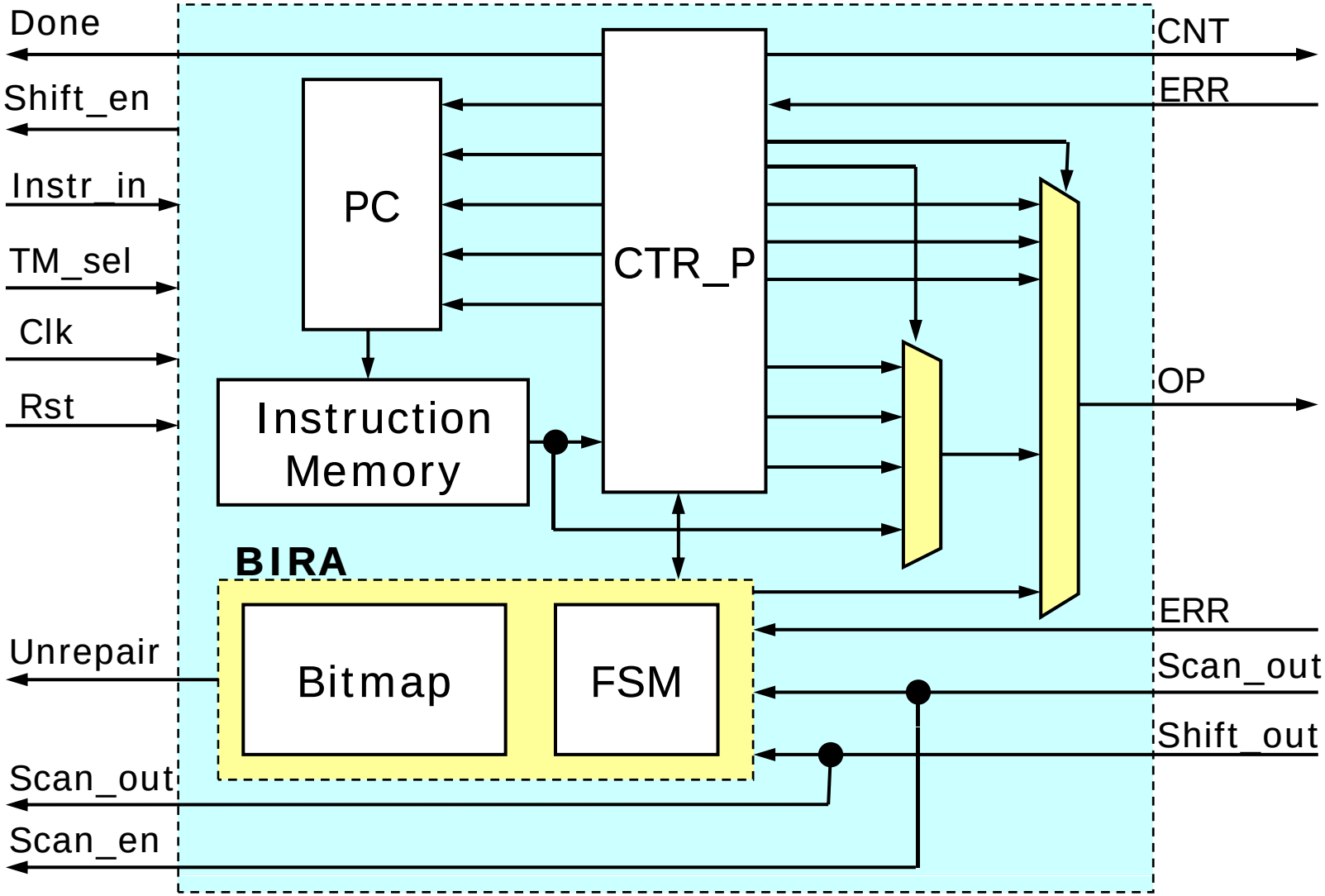
	c0	c1	c2	c3	CID
r0	1	0	0	0	
r1	0	1	0	1	
r2	0	1	0	0	
r3	0	0	1	0	

RID

RA algorithm	**Instructions**
Local repair-most (LRM) alg.	LRM
Essential spare pivoting (ESP) alg.	{FHFR, ROW_FIRST, COL_FIRST}
Row first alg.	{ROW_FIRST, COL_FIRST}
Column first alg.	{COL_FIRST, ROW_FIRST}

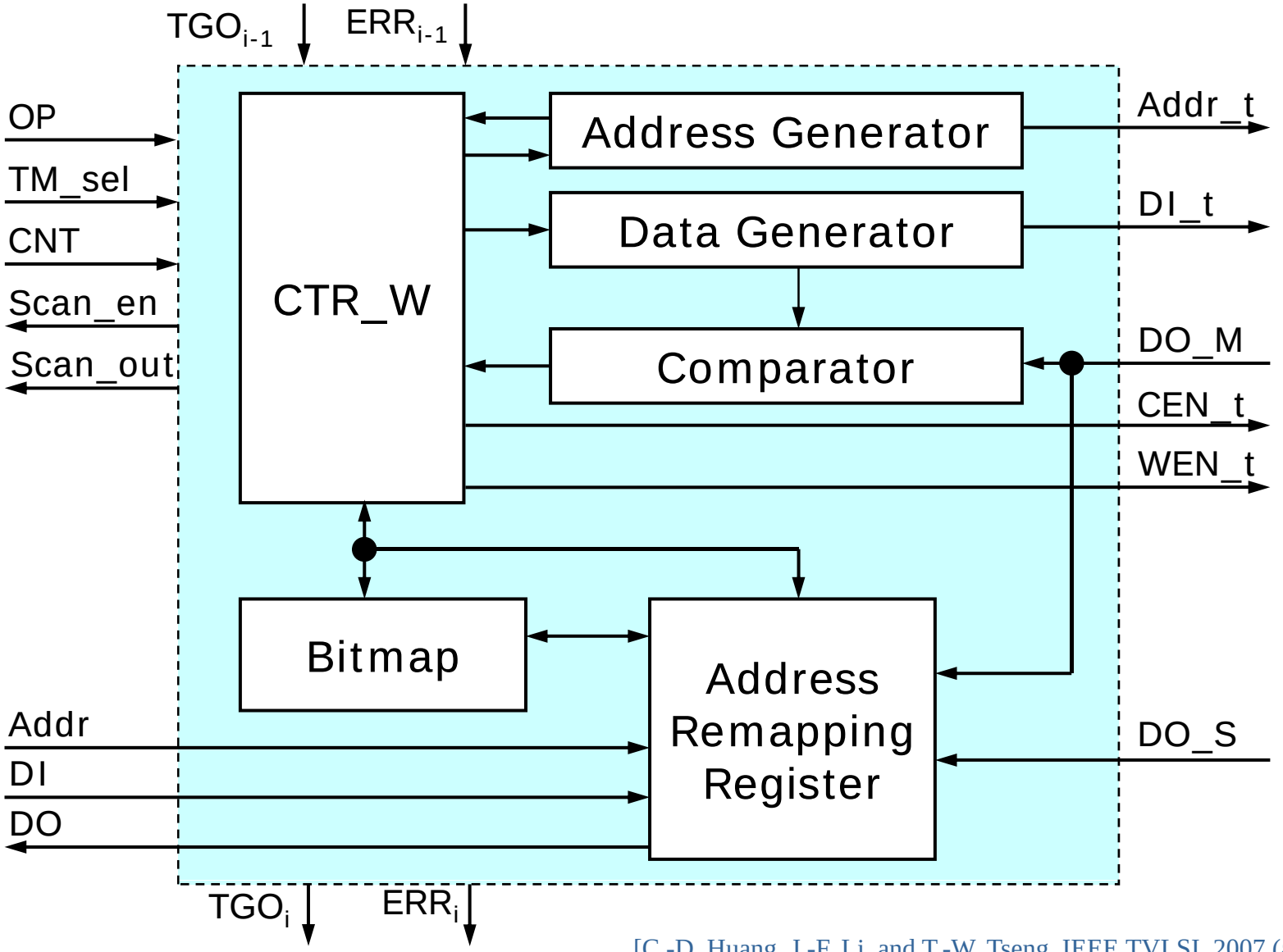
[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]

Block Diagram of the ProTaR



[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]

Block Diagram of the Wrapper



[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]

Area Cost of the Wrapper

- Area overhead of the Wrapper is defined as the ratio of the area of the wrapper to the area of the corresponding memory
- Experimental results for an 8Kx64-bit memory

Redundancy Configuration	Wrapper Area	Area Overhead
2R2C	6739 gates	2.3%
2R3C	7342 gates	2.5%
3R3C	8317 gates	2.8%

- Area cost of Wrappers for different memory sizes

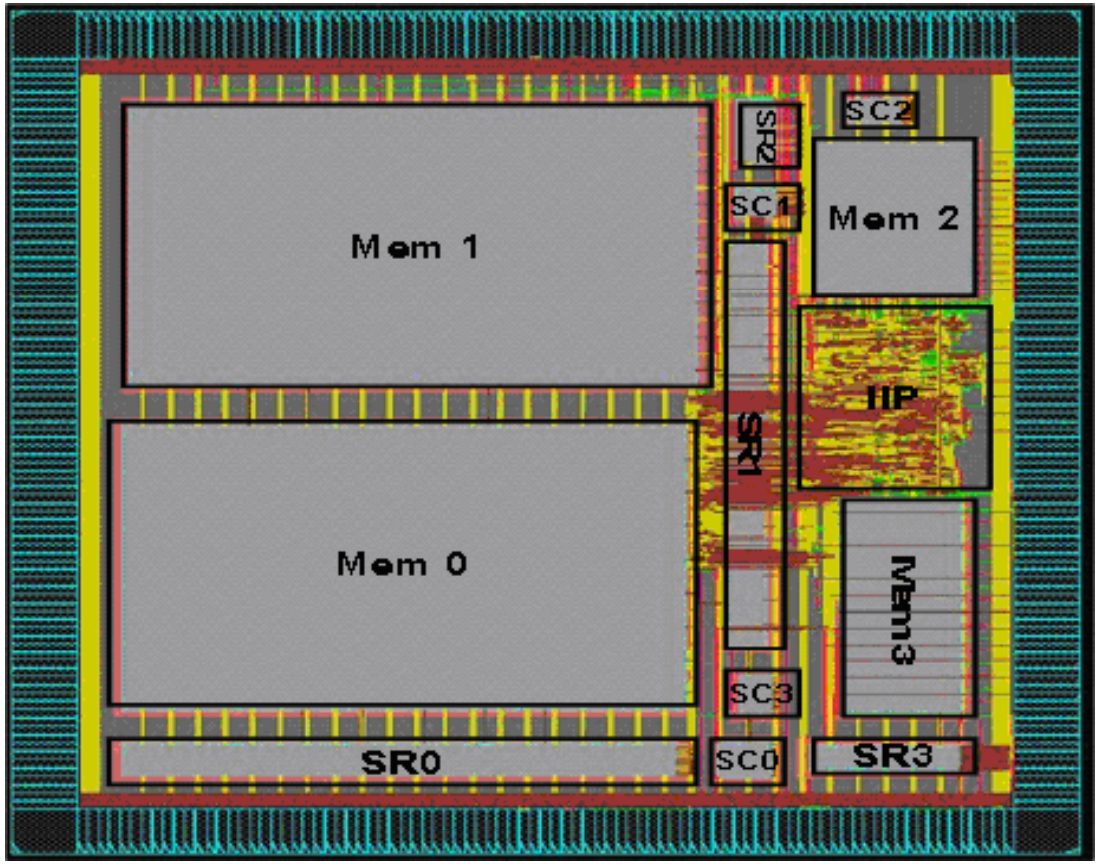
Memory Configuration	Wrapper Area	Area Overhead
8K x 16	3944 gates	4.6%
4K x 32	4825 gates	5.8%
2K x 64	6501 gates	7.1%

Area Cost of the IIP

- An IIP for four memories is implemented
  - The size of Mem0, Mem1, Mem2, and Mem3 are 8Kx64, 8Kx64, 4Kx14, and 2Kx32, respectively
  - The redundancy configurations of the Mem0, Mem1, Mem2, and Mem3 are (3x3), (2x2), (2x2), and (2x2), respectively
- The area of the four memories is 6798472um^2
- The area of all the redundancies is about 896060um^2
- The area of the IIP is only about 309893um^2
- Thus, the area overhead of the IIP is only about 4.56%

Layout View of the IIP

- Layout view of the proposed IIP for four RAMs



[C.-D. Huang, J.-F. Li, and T.-W. Tseng, IEEE TVLSI, 2007 (accepted)]

Outline

- Introduction
- Built-In Self-Repair (BISR) Techniques for RAMs
- Error Correction Code and Transparent Test Techniques for RAMs
- Conclusions

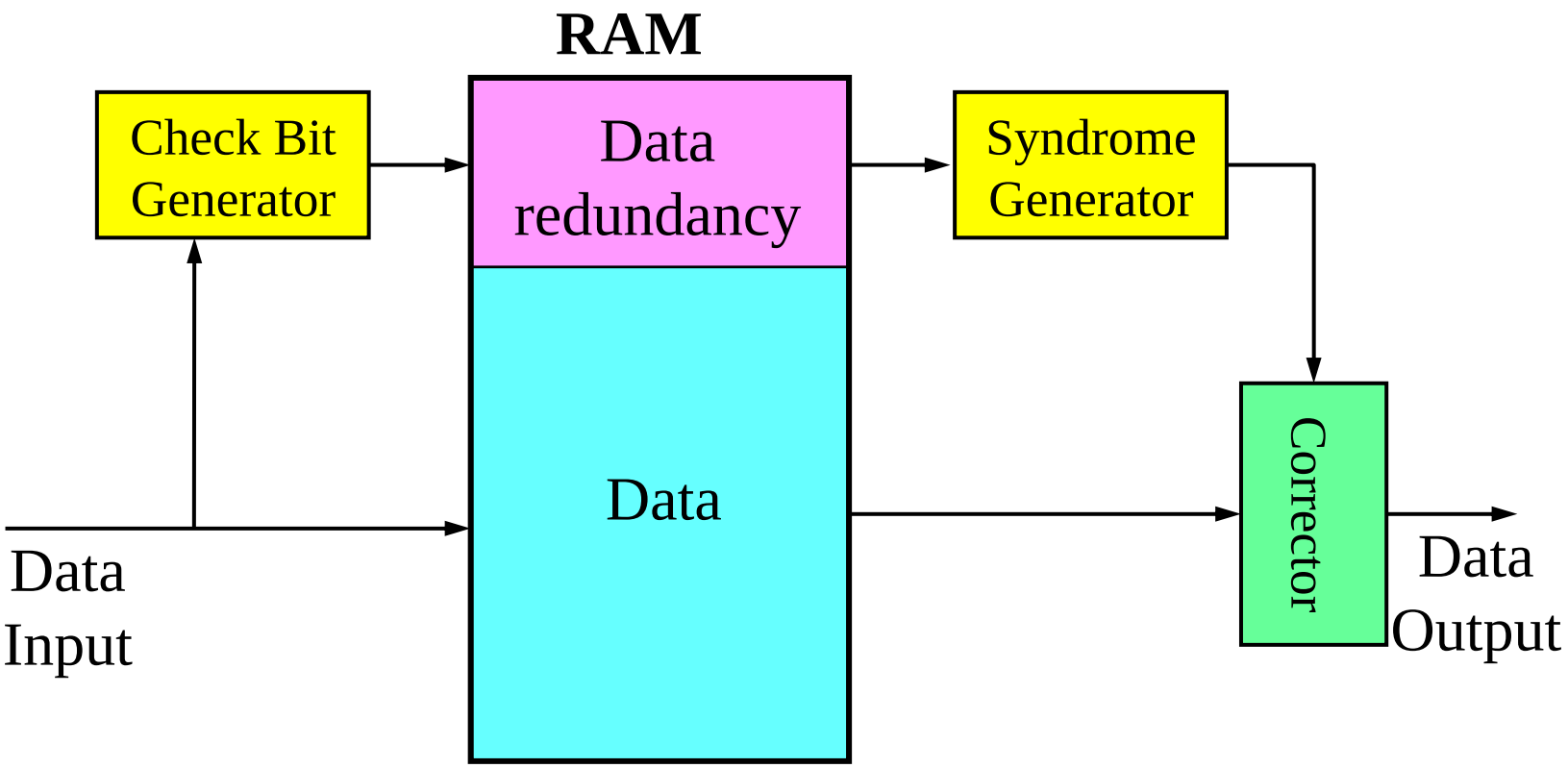
Reliability-Enhancement Techniques

- Fault-tolerant techniques are widely used to improve the reliability of systems
- All fault-tolerant techniques require redundancy
  - Redundancy is simply the addition of information, resources, or time beyond what is needed for normal system operation
- Types of redundancy
  - Hardware redundancy
  - Software redundancy
  - Information redundancy
  - Time redundancy

Memory Reliability-Enhancement Techniques

- Hardware redundancy
  - Built-in self-repair technique
- Error correction code
  - Use information redundancy to protect stored data from soft error
- Periodic transparent testing
  - Periodically apply tests to detect hard faults manifested by latent faults

Typical Error-Correction-Code Scheme



Hamming Error-Correction Code

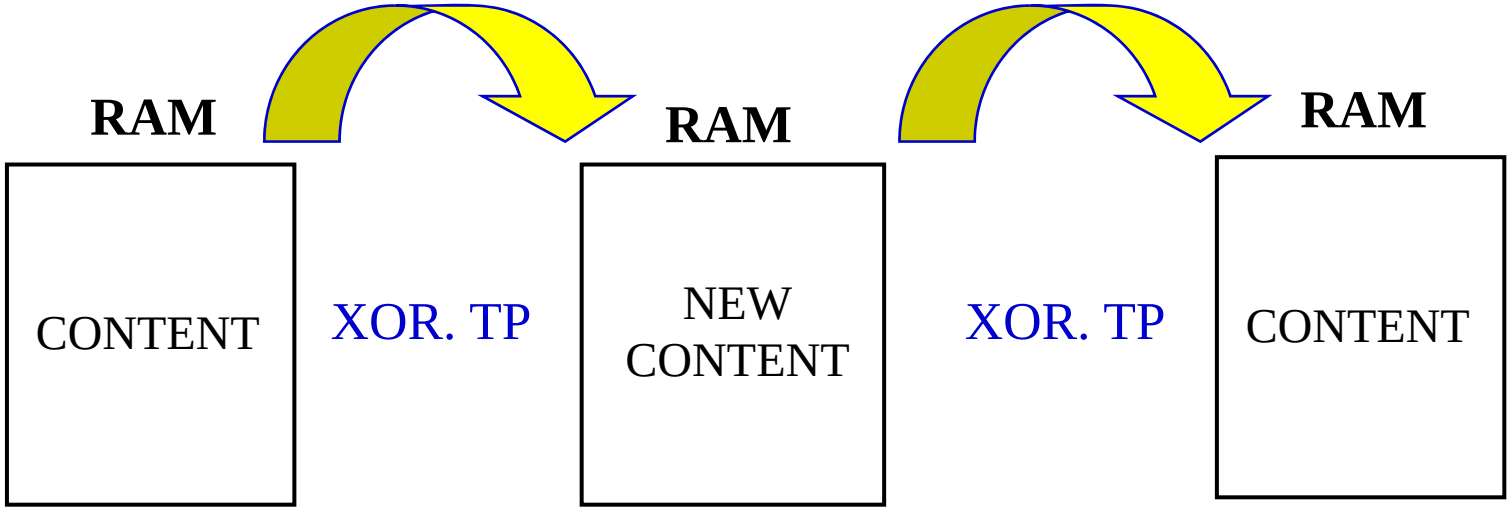
- The Hamming single error-correction code uses c parity check bits to protect k bits of information. The relationship between the values of c and k is
  - $2^c \geq c + k + 1$
- Suppose that there are four information bits (d_3, d_2, d_1, d_0) and, as a result, three parity check bits (c_1, c_2, c_3). The bits are partitioned into groups as (d_3, d_1, d_0, c_1), (d_3, d_2, d_0, c_2), and (d_3, d_2, d_1, c_3). Each check bit is specified to set the parity of its respective group, i.e.,

$$c_1 = d_3 + d_1 + d_0$$
$$c_2 = d_3 + d_2 + d_0$$
$$c_3 = d_3 + d_2 + d_1$$

What is Transparent Test?

- Transparent testing
  - Leave the original content of the circuit under test unchanged after the testing is completed if no faults are presented
- Features
  - Ensure the reliability of stored data throughout its life time
  - Provide better fault coverage than non-transparent testing for unmodeled faults
- Limitation
  - Must be performed while systems are idle

Principle of Transparent Testing



1. Read (CONTENT), take signature S(CONTENT)
2. Read (CONTENT), Write (CONTENT. XOR. TP)=NEW_CONTENT
3. Read (NEW_CONTENT), take new signature S(NEW_CONTENT)
4. Write (NEW_CONTENT. XOR. TP)

NEW_CONTENT. XOR. TP=CONTENT. XOR. TP. XOR. TP=CONTENT

S(NEW_CONTENT)=S(CONTENT. XOR. TP)=S(CONTENT). XOR. S(TP)

Issues of Transparent Testing

- Test interrupts
  - In comparison with manufacturing testing, one special issue of transparent testing is that the transparent testing process may be interrupted
- Aliasing
  - If a transparent built-in self-test scheme is considered, the signature generation typically is done by a MISR
- Fault location
  - If a fault is detected, it is very difficult to locate the fault

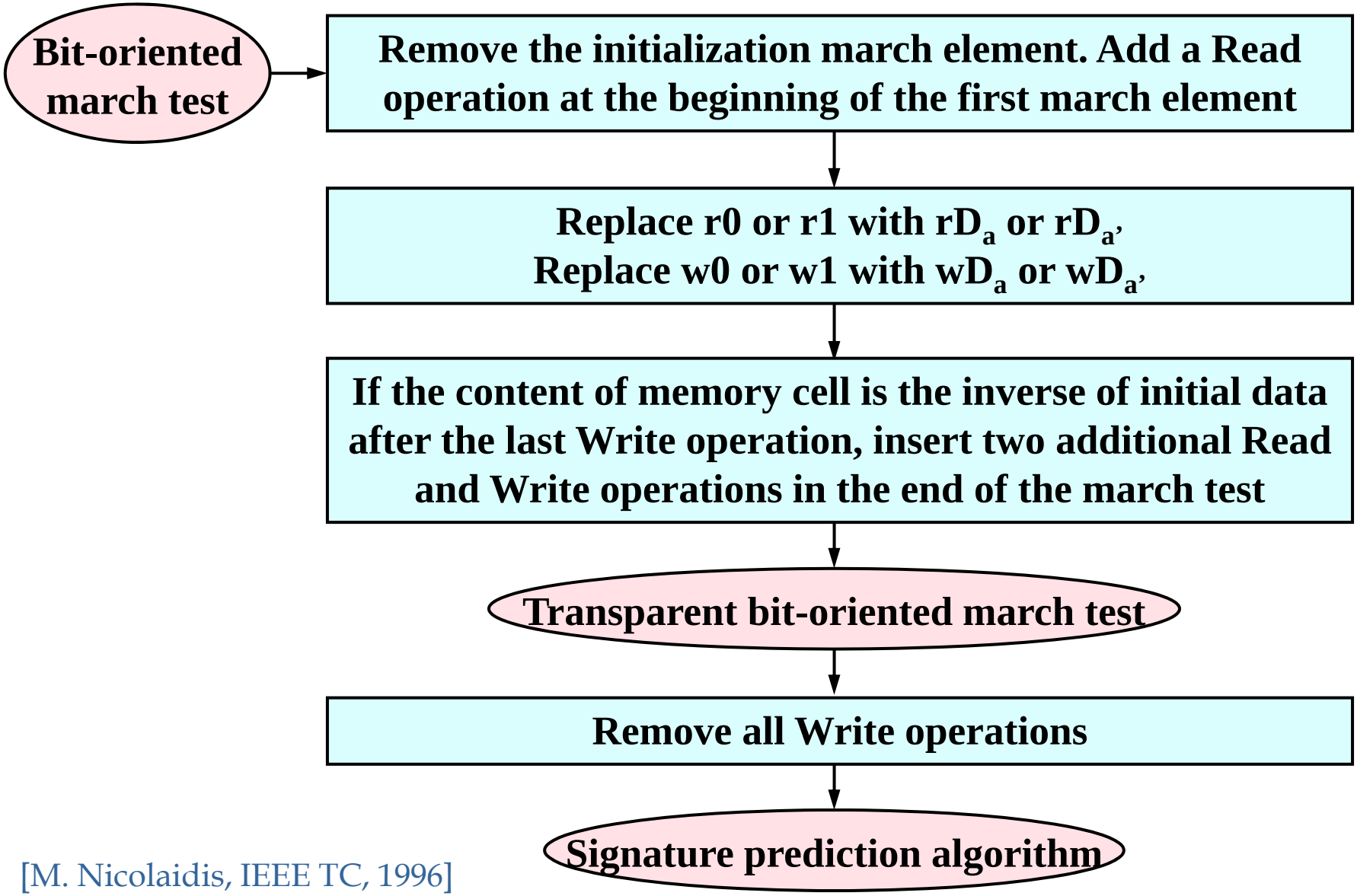
A Typical Transparent March Test

- A typical transparent March test consists of two-phase tests
  - Signature-prediction test
  - Transparent March test
- Types of transparent test schemes
  - Transparent March tests
  - Symmetric transparent March tests
  - Combination of Transparent March tests and ECCs

Notation

- In a test algorithm
  - D denotes the initial content of a cell or a word for bit-oriented or word-oriented memories
  - D_a is data of the bit-wise XOR operation on D and a
  - $\Uparrow (\Downarrow)$ represents the ascending (descending) address sequence
  - $\Updownarrow$ denotes either ascending or descending address sequence
  - wX denotes a write X operation
  - rX denotes a read operation with expect data X

A Typical Transformation Method



[M. Nicolaidis, IEEE TC, 1996]

An Example

- Consider the March C- test:
  - $\{\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r1,w0);\Downarrow(r0,w1);\Updownarrow(r0)\}$
- After Step 1 transformation:
  - $\{\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r1,w0);\Downarrow(r0,w1);\Updownarrow(r0)\}$
- After Step 2 transformation:
  - $\{\Uparrow(rD_a,wD_{\bar{a}});\Uparrow(rD_{\bar{a}},wD_a);\Downarrow(rD_a,wD_{\bar{a}});\Downarrow(rD_{\bar{a}},wD_a);\Updownarrow(rD_a)\}$
- The content of memory cell after the last operation is the same as the initial state. Step 3 is omitted.
- Thus, the transparent March C- test is as follows:
  - $\{\Uparrow(rD_a,wD_{\bar{a}});\Uparrow(rD_{\bar{a}},wD_a);\Downarrow(rD_a,wD_{\bar{a}});\Downarrow(rD_{\bar{a}},wD_a);\Updownarrow(rD_a)\}$
- Remove the Write operations. The signature prediction algorithm is as follows:
  - $\{\Uparrow(rD_a);\Uparrow(rD_{\bar{a}});\Downarrow(rD_a);\Downarrow(rD_{\bar{a}});\Updownarrow(rD_a)\}$

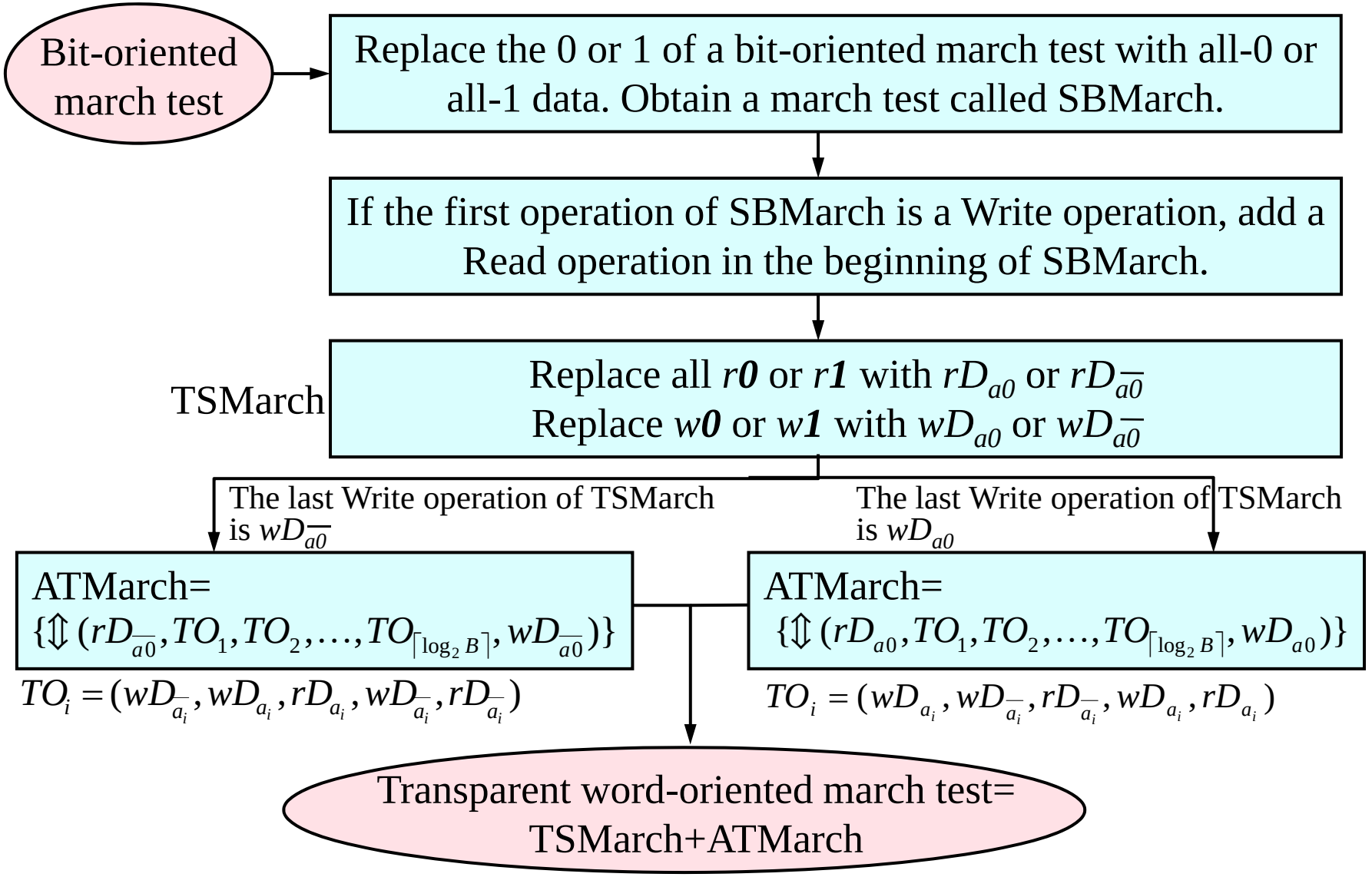
Word-Oriented Transparent Tests

- Word-oriented transparent test can be obtained
  - By applying the transformation rules to all the bits of each word [Nicolaids, ITC92].
- E.g., a word-oriented March C- for 4-bit words
  - T1: $\{\Updownarrow(w0000);\Uparrow(r0000,w1111);\Uparrow(r1111,w0000);\Downarrow(r1111,w0000);$ $\Downarrow(r0000,w1111);\Updownarrow(r0000)\}$
  - T2: $\{\Updownarrow(w0101);\Uparrow(r0101,w1010);\Uparrow(r1010,w0101);\Downarrow(r0101,w1010);$ $\Downarrow(r1010,w0101);\Updownarrow(r0101)\}$
  - T3: $\{\Updownarrow(w0011);\Uparrow(r0011,w1100);\Uparrow(r1100,w0011);\Downarrow(r0011,w1100);$ $\Downarrow(r1100,w0011);\Updownarrow(r0011)\}$
- Thus, the transparent word-oriented March C-
  - T1': $\{\Uparrow(rD_{a0},wD_{\overline{a0}});\Uparrow(rD_{\overline{a0}},wD_{a0});\Downarrow(rD_{a0},wD_{\overline{a0}});\Downarrow(rD_{\overline{a0}},wD_{a0});\Updownarrow(rD_{a0},wD_{a1})\}$
  - T2': $\{\Uparrow(rD_{a1},wD_{\overline{a1}});\Uparrow(rD_{\overline{a1}},wD_{a1});\Downarrow(rD_{a1},wD_{\overline{a1}});\Downarrow(rD_{\overline{a1}},wD_{a1});\Updownarrow(rD_{a1},wD_{a2})\}$
  - T3': $\{\Uparrow(rD_{a2},wD_{\overline{a2}});\Uparrow(rD_{\overline{a2}},wD_{a2});\Downarrow(rD_{a2},wD_{\overline{a2}});\Downarrow(rD_{\overline{a2}},wD_{a2});\Updownarrow(rD_{a2},wD_{a0})\}$

Problem

- Transparent tests are usually applied in the idle state of systems or components
- Reducing the test time is very important
  - Avoiding the interrupt of testing
- However, conventional transparent word-oriented march tests are directly obtained
  - By executing the corresponding bit-oriented march test on each bit of word
- Thus, conventional transformation does not generate a time-efficiency word-oriented march test

Efficient Word-Oriented Transparent Tests



Source: J.-F. Li, IEEE TCAD, 2007 (accepted)

Example

- Consider a bit-oriented March U [15]
  - $\{\Updownarrow(w0);\Uparrow(r0,w1,r1,w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0,w1);\Downarrow(r1,w0)\}$
- Then, the solid March U (SBMarch U) is as follows
  - $\{\Updownarrow(w\vec{0});\Uparrow(r\vec{0},w\vec{1},r\vec{1},w\vec{0});\Uparrow(r\vec{0},w\vec{1});\Downarrow(r\vec{1},w\vec{0},r\vec{0},w\vec{1});\Downarrow(r\vec{1},w\vec{0})\}$
  - where $\vec{0}$ and $\vec{1}$ denote all-0 and all-1 data
- According to the transformation rules described above, the transparent SBMarch U (TSMarch U) is
  - $\{\Uparrow(rD_{a0},wD_{\overline{a0}},rD_{\overline{a0}},wD_{a0});\Uparrow(rD_{a0},wD_{\overline{a0}});\Downarrow(rD_{\overline{a0}},wD_{a0},rD_{a0},wD_{\overline{a0}});\Downarrow(rD_{\overline{a0}},wD_{a0})\}$
  - where *a0* denotes all-0 data
- The last operation of TSMarch U is wD_{a0}
  - ATMarch= $\Updownarrow(rD_{a_0},wD_{a_1},wD_{\overline{a_1}},rD_{\overline{a_1}},wD_{a_1},rD_{a_1},wD_{a_2},wD_{\overline{a_2}},rD_{\overline{a_2}},wD_{a_2},rD_{a_2},wD_{a_3},wD_{\overline{a_3}},rD_{\overline{a_3}},wD_{a_3},rD_{a_3},wD_{a_0})$

Symmetric Transparent Tests

- Feature
  - The symmetric transparent test method take advantage of the symmetric characteristic of a signature analyzer to eliminate the signature prediction phase
- Symmetric characteristic of a signature analyzer
  - Let $sig(z, S, h)=u$ denote a serial signature analyzer which has an initial state S, a feedback polynomial h, a data string for analysis z, and the corresponding signature u. Then we can obtain $sig(z^*, u^*, h^*)=S^*$, where z^*, u^*, h^*, and S^* denote the reverse of z, u, h, and S, respectively [V. N. Yarmolik and S. Hellebrand, DATE99]

An Example

- A $2n$-bit data string $Z=(x_{2n-1}x_{2n-2}\ldots x_n x_{n-1}\ldots x_1 x_0)$ is called a symmetric data string if
  - $x_{n-1}=x_n, x_{n-2}=x_{n+1}, \ldots x_1=x_{2n-2}, x_0=x_{2n-1}$
  - or $x_{n-1}=\overline{x_n}, x_{n-2}=\overline{x_{n+1}}, \ldots, x_1=\overline{x_{2n-2}}, x_0=\overline{x_{2n-1}}$
- Consider a symmetric data string $Z=(zz^*)$. Assume that a reconfigurable signature analyzer $sig(-, \boldsymbol{0}, h)$ is used to analyze the symmetric data string Z
  - Step 1: z is analyzed and $sig(z, \boldsymbol{0}, h)=u$
  - Step 2: analyzer is configured as $sig(-, u^*, h^*)$
  - Step 3: z^* is analyzed and $sig(z^*, u^*, h^*)=\boldsymbol{0}^*=\boldsymbol{0}$

Symmetric Transparent March Tests

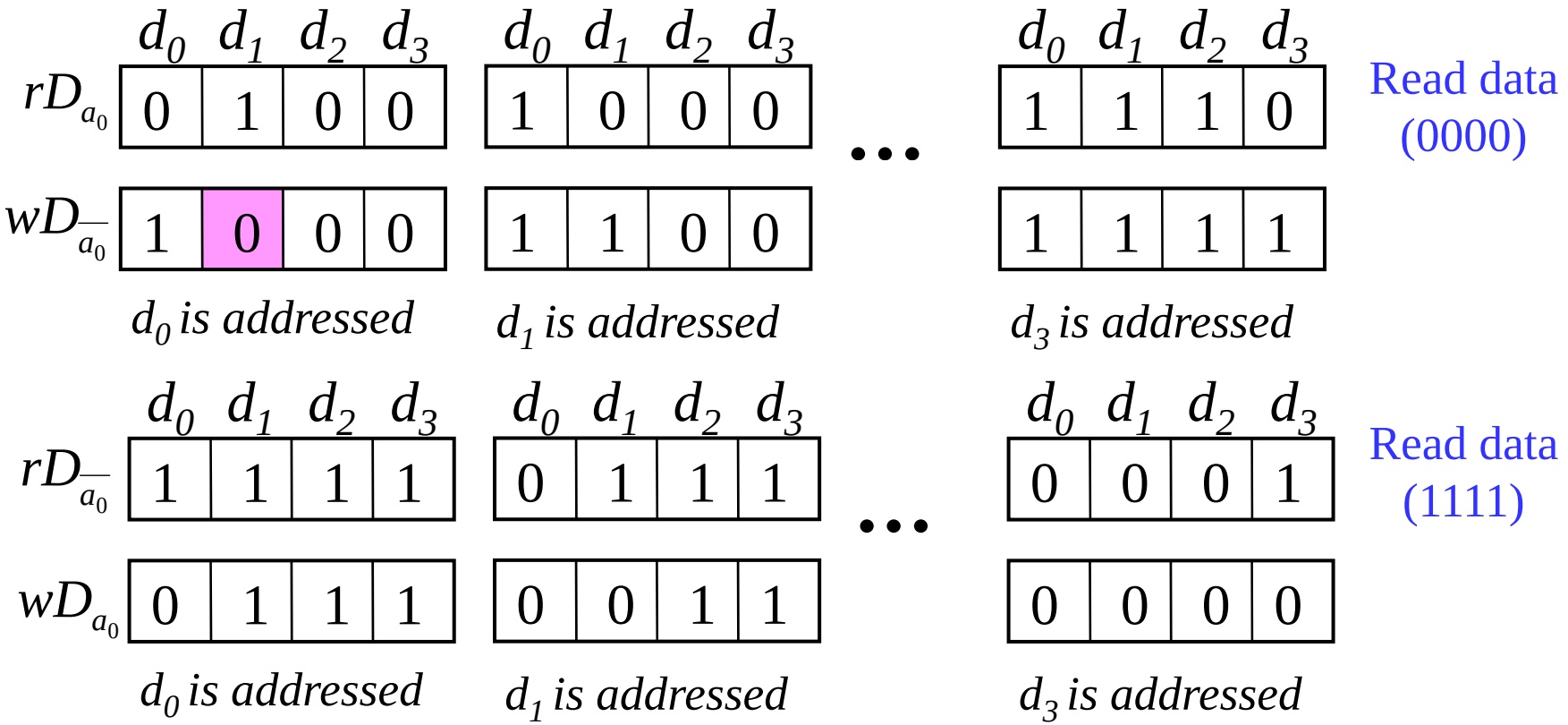
- A transparent March test is a symmetric transparent March test if the read data of the Read operations of the transparent March test is a symmetric data string Z
- For example, consider the March test MATS+
  - $\{\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0)\}$
- It can be transformed to a transparent March test
  - $\{\Uparrow(rD_{a_0},wD_{\overline{a_0}});\Downarrow(rD_{\overline{a_0}},wD_{a_0})\}$
  - The read data can be expressed as $Z=(z,z^{*c})$

Limitations

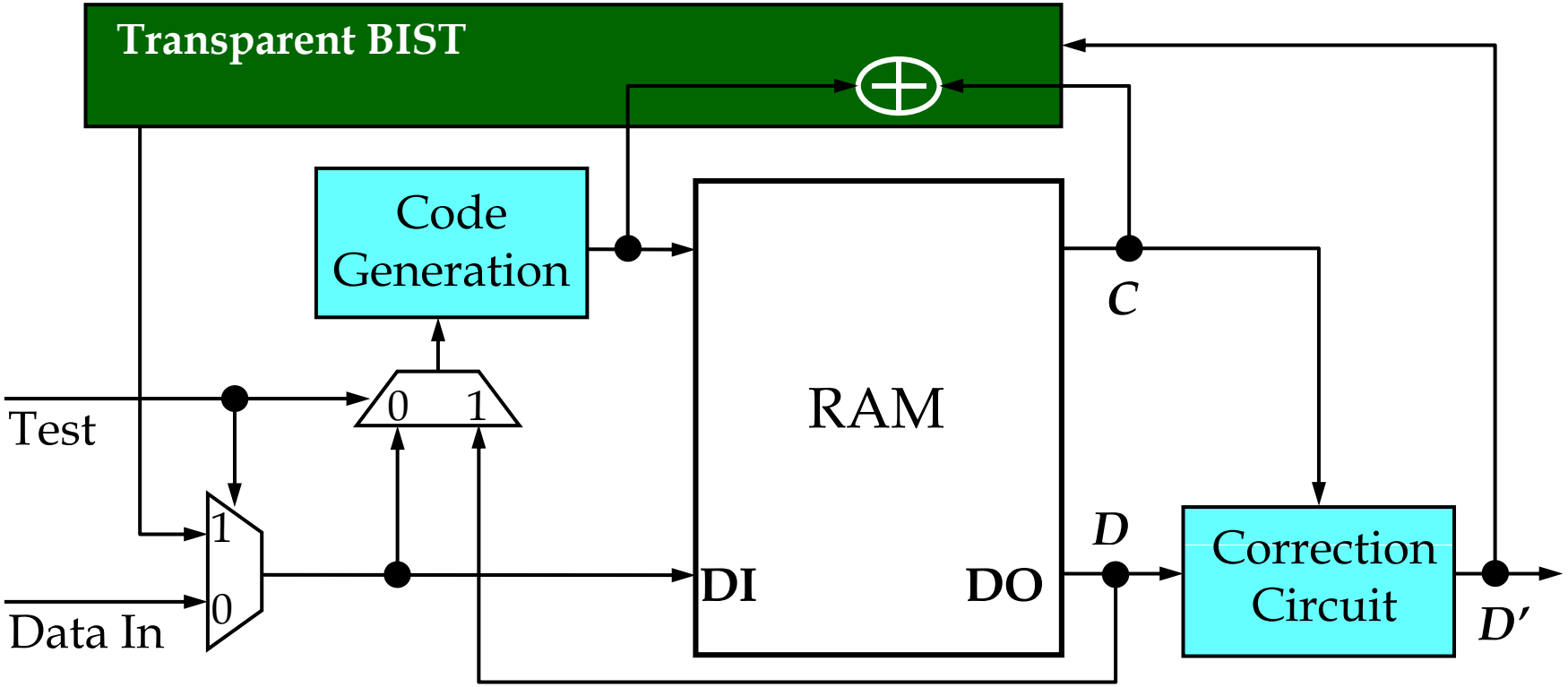
- Symmetric transparent March tests have two major limitations
  - Fault masking effect
  - Test interrupts cause the symmetric characteristic to be invalid
- Consider a 4-bit memory with initial content $(d_0 d_1 d_2 d_3)$=(0100). Assume that the memory has an idempotent coupling fault in which the aggressor and victim are at d_0 and d_1. Also, the value of the victim is forced to 0 while the aggressor has a 0 to 1 transition

Fault Masking Effect

- Assume that the symmetric transparent MATS+ is used to test a 4-bit memory with a CFid
  - Transparent MATS+: $\{\Uparrow(rD_{a_0}, wD_{\overline{a_0}}); \Downarrow(rD_{\overline{a_0}}, wD_{a_0})\}$



Transparent Test Scheme for a RAM with ECC



Read: Read the data D at DO; Check if Code_Gen(D)=C
Write: Write the data D'. XOR. TP

Source: J.-F. Li, IEEE TCAD, 2007 (accepted)

Features

- No signature prediction phase is needed. This shortens the testing time such that the probability of an interruption is reduced
- Really restoring the original content of the memory under test is achieved if the number of faulty bits of a word is less than the correction capability of the applied ECC
- It can locate the faulty bit of the faulty word by the checking response. The fault location capability is also related to the correction capability of the applied ECC

Example

- Consider a 3x4-bit memory with Hamming ECC. Also, the transparent March MATS+ is used to test the memory
  - Transparent March MATS+: $\{\Uparrow(rD_{a_0}, wD_{\overline{a_0}}); \Downarrow(rD_{\overline{a_0}}, wD_{a_0})\}$
  - Assume that d_2 of the first word has a stuck-at-0 fault

	$d_3\ d_2\ d_1\ d_0$	$h_2\ h_1\ h_0$
W_0	1 0 0 1	1 0 0
W_1	1 1 1 0	1 0 0
W_2	1 0 1 0	0 1 0

	$d_3\ d_2\ d_1\ d_0$	$h_2\ h_1\ h_0$
W_0	0 0 1 0	0 1 1
W_1	0 0 0 1	0 1 1
W_2	0 1 0 1	1 0 1

	$d_3\ d_2\ d_1\ d_0$	$h_2\ h_1\ h_0$
W_0	1 0 0 1	1 0 0
W_1	1 1 1 0	1 0 0
W_2	1 0 1 0	0 1 0

$h_0 = d_3 + d_1 + d_0$
$h_1 = d_3 + d_2 + d_0$
$h_2 = d_3 + d_2 + d_1$

Conclusions

- Embedded memories represent more and more area of system-on-chip (SOC) designs
  - The BISR technique is essential for memories in SOCs
- With the advent CMOS technology, enhancing the reliability of an integrated circuit becomes one major challenge
  - Effective and efficient reliability-enhancement techniques must be developed

References

1. T. Kawagoe, J. Ohtani, M. Niiro, T. Ooishi, M. Hamada, and H. Hidaka, “A built-in self-repair analyzer (CRESTA) for embedded DRAMs,” in *Proc. Int’l Test Conf. (ITC)*, 2000, pp. 567.574.
2. V. Schober, S. Paul, and O. Picot, “Memory built-in self-repair using redundant words,” in *Proc. Int’l Test Conf. (ITC)*, Baltimore, Oct. 2001, pp. 995-1001.
3. Y. Zorian, “Embedded memory test & repair: Infrastructure IP for SOC yield,” in *Proc. Int’l Test Conf. (ITC)*, Baltmore, Oct. 2002, pp. 340.349.
4. C.-T. Huang, C.-F. Wu, J.-F. Li and C.-W. Wu,”Built-in redundancy analysis for memory yield improvement,” *IEEE Trans. Reliability*, vol. 52, no. 4, pp. 386-399, Dec. 2003.
5. J.-F. Li, J.-C. Yeh, R.-F. Huang, C.-W. Wu, P.-Y. Tsai, A. Hsu, and E. Chow,”A built-in self-repair scheme for semiconductor memories with 2-D redundancies,” in *Proc. IEEE Int. Test Conf. (ITC)*, (Charlotte), pp. 393-402, Sept. 2003.
6. J.-F. Li, J.-C. Yeh, R.-F. Huang, and C.-W. Wu,”A built-in self-repair design for RAMs with 2-D redundancies,” *IEEE Trans. Very Large Scale Integration Systems*, vol.13, no.6, pp. 742-745, June, 2005.
7. S.-K. Lu, Y.-C. Tsai, C.-H. Hsu, K.-H. Wang, and C.-W. Wu, “Efficient built-in redundancy analysis for embedded memories with 2-D redundancy,” *IEEE Trans. on VLSI Systems*, vol. 14, no. 1, pp. 34–42, Jan. 2006.
8. T.-W. Tseng, J.-F. Li, C.-C. Hsu, A. Pao, K. Chiu, and E. Chen, "A reconfigurable built-in self-repair scheme for multiple self-repairable RAMs in SOCs," in *Proc. IEEE Int. Test Conf. (ITC)*, (Santa Clara), Paper 30.2, pp. 1-8, Oct. 2006.
9. C.-D. Huang, J.-F. Li, and T.-W. Tseng,”ProTaR: an infrastructure IP for repairing RAMs in SOCs,” *IEEE Trans. Very Large Scale Integration Systems*, vol., no., pp. , April 2007 (accepted).
10. R.-F. Huang, J.-F. Li, J.-C. Yeh, and C.-W. Wu,”RAISIN: a tool for evaluating redundancy analysis schemes in repairable embedded memories,” *IEEE Design and Test of Computers*, vol., no. , pp. , May 2005 (accepted).
11. M. Nicolaidis,“Theory of transparent BIST for RAMs,” *IEEE Trans. on Computers*,vol. 45, no. 10, pp. 1141–1156, Oct. 1996.
12. V. N. Yarmolik and S. Hellebrand, “Symmetric transparent BIST for RAMs,” in *Proc. Conf. Design, Automation, and Test in Europe (DATE)*, 1999, pp. 702–707.
13. J.-F. Li,”Transparent test methodologies for random access memories with/without ECC,” *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol., no., pp. , Feb. 2007 (accepted).