High Density 65nm and Low Power 90nm/130nm Test and Repair Application Note

Confidential

July 2009

ARM PAN 0026A

Revision History

Part Number	Release Date	Comments
app_metro_flexrepair 2005q3v1	July 2005	Initial Release
app_metro_flexrepair_2005q3v2	July 2005	Add BIST algorithm access instructions
app-metro-tr-2006q1v1	January 2006	Add confidentiality material and Format upgrade
app-metro-tr-2007q2v1	June 2007	Update copyright and preface
app-metro-tr-2008q3v1	September 2008	Update title and remove bibliography
ARM PAN 0026A	July 2009	Replaced Metro product name with high density, updated copyright information and added revision appendix table

Customer Support

Customers with active Support contracts can obtain support for ARM Physical IP products by going to access.arm.com and clicking on the “Products & Services > New Technical Request.” link on the left side of the webpage. ARM recommends using this method for customers with valid support contracts, in order to obtain prompt attention to issues and questions.

Information about available Support contract options can be seen at www.arm.com/products/physicalip/support.html.

If you cannot reach us via the web support channel you can contact ARM Physical IP support via email at support-pipd@arm.com for technical issues.

Typographic Conventions

The following typographic conventions are used to assist you in distinguishing special notations, values, and elements described in this manual.

Visual Cue	Meaning
(Bullet) •	Bulleted list of important items.
`Courier Type`	Commands typed on the keyboard, either examples or instructions.
Dash (-) `Courier Type`	Text set in Courier type and preceded by a dash represents a command name (for example, -`libname`).
<italic type> *italic type*	Variable names you select, such as file and directory names are enclosed within angle brackets (< >). Italic type is used to show variable values, file, and directory names.
(Ellipsis) …	Indicates commands or options that may be added.
Italic Type with Initial Capital Letters	Document, chapter, section and reference manual names.

Table of Contents

1. Overview

Since memories can occupy 50% or more of total System on Chip (SoC) area, high quality memory tests become vital to overall chip quality. Most literature concerning this subject has concentrated on built-in self-test (BIST) and repair solutions for embedded SRAM or on assisting ATE. Because SRAM circuits are extremely dense, they are more susceptible to defects than other types of circuitry such as standard cells or I/O cells. Many foundries report defect densities in SRAMs at twice the rate of other circuitry. These elements and limitations also apply to register files (RF).

This document describes the benefits of test and repair, and how to calculate and determine how much redundancy is needed. Also, this document describes test and repair features that enable user testability and are specific to ARM's high density memory compilers. This document does not address BIST or repair algorithms, nor does it consider the memory design or verification process itself.

2. Test and Repair Features

ARM's Artisan compiler has several test and repair features that are available with its high density memory compilers. These features may include Flex-Repair, Soft Error Repair (SER), Built-In Self Test (BIST) MUXes, and Extra Margin Adjustment (EMA). Other test features may be available with your specific memory compiler. You can check the "Compiler Options" section in Chapter 2 of any memory compiler user guide for a complete list of available test and repair features.

3. Calculating Redundancy

Redundancy calculations stem from a random defect yield model relating die area and an observed or estimated defect occurrence probability. In the foundry business, the details of these models are critical, and yield distinctions of less than a percentage point can mean the difference between profit and loss. For redundancy calculation on a given chip, however, these details are unimportant, because we are interested in the magnitude of yield improvement, not in precise yield numbers. For simplicity, we will use the Poisson yield model in this document to discuss calculating redundancy:

$$Y = e^{-AD}$$

where:

Y = yield
A = area
D = overall defect density

Note

Common yield models, such as the Poisson, Bose-Einstein, Murphy, and Negative Binomial, typically give a spread of estimated yields that totals about ±10% of predicted yield loss for given source data and die size in the 25 to 200 mm^2 range (that is, a spread of 27%-33% for a predicted yield loss of 30%). Individual ASICs, particularly those with low manufacturing volume, may not be able to statistically resolve observed yield to that level of accuracy. Redundancy calculation makes additional approximations that further obscure the distinctions between the models.

Redundancy improves yield by repairing defects, or more precisely, by eliminating failures associated with defects. In the Poisson model, the number of defects is given by the Poisson distribution, so the probability of *k* defects is given by the following equation:

$$P(k) = \frac{(AD)^k e^{-AD}}{k!}$$

Note the equation above reduces to the previous equation when *k* is 0.

Imagine a perfect repair process that can repair any *n* defects. The yield for a chip using this perfect repair scheme would be given by this equation:

$$Y_{perfect} = \sum_{k=0}^{n} P(k) = \sum_{k=0}^{n} \frac{(AD)^k e^{-AD}}{k!}$$

In reality, repair schemes are not perfect and cannot repair all defects. For example, a row-only repair scheme cannot repair a defective column. In general, a method like "X" will be able to repair some percentage of single defects, a different percentage of double defects, and so on. This can be represented as:

$$Y_{repair-X} = \sum_{k=0}^{n} f_x(k)P(k) = \sum_{k=0}^{n} f_x(k)\left(\frac{(AD)^k e^{-AD}}{k!}\right)$$

where $f_x(k)$ represents the probability of method X repairing *k* defects (clearly $f_x(0) = 1$). The details of this function are important when considering different repair schemes, but typical values for a double row repair scheme can repair 70% of all single defects, 50% of all double defects, and 0% of any defect number greater than 2. A single row scheme, on the other hand, can repair 70% of all single defects, but 0% for any higher order number.

3.1 Determining Amount of Redundancy Needed

Determining which redundant elements are needed to improve yield is application dependent. The need for redundancy is based primarily on the total number of memory bits on a chip, but the mechanics of implementing it depends on the distribution of individual memories. For example, an "extra I/O" scheme might make sense for 64-bit words, with less than 2% overhead, but will be unreasonable for 8-bit words, where an integrated physical column approach could be more reasonable.

Note

Memories are divided into physical rows and columns based on the number of words stored and their width. For example, a 128 Kbit memory of 16K words of 8 bits each requires 14 address bits, which can be implemented using 16K rows and 8 columns. However, the resulting memory would be area inefficient. More reasonably, it can be implemented using 1024 rows (10 bits of the address) and 128 columns (8 bits per word multiplexed 16 ways – the remaining 4 address bits). It can also be implemented using 512 rows and 256 columns (32 way multiplexing), and so on. A spare I/O redundancy scheme will add 16 physical columns to the 1024x128 memory and 32 to the 512x256 memory. A physical column redundancy scheme might add fewer columns to each.

In addition, deciding on which redundancy element is most needed depends on relative defect probabilities. For example, should repairable memory include extra rows or columns? Ideally, the memory should include both extra rows and columns for the broadest set of repair options. However, you might have area constraints that prevent the use of both extra rows and columns. In such an example, you might decide to use extra columns rather than rows because differential bit/bit-bar pairs may be susceptible to shorts if unshielded wires are run adjacent to one another and lead to a higher probability of column failures than row failures.

4. Flex-Repair

Flex-Repair (Redundancy) is available for all high density Register File (RF) and SRAM compilers. Flex-Repair for SRAM is more complex than for Register File.

Flex-repair involves the shifting of columns and rows. The CREN repair signal for RF1 and RF2 are configured statically, that is, set once and never changed. The repair signals for RA1 and RA2 need to be configured dynamically, based on the address.

Table 1 shows the redundancy options high density Flex-Repair.

Table 1. High Density Flex-Repair Matrix

Compiler	Spare I/Os	Spare Columns	Spare Rows
RA1	0	4 per bank	2 per bank
RA2	0	4 per bank	2 per bank
RF1	1	0	0
RF2	1	0	0

4.1 SRAM Flex-Repair/Redundancy

The Flex-Repair redundancy approach for SRAM has been developed to enable repair of the most common defects in sub-130nm CMOS memories. Specific failure modes and probabilities are process and bit cell dependent, but tend to follow a distribution similar to that given in Table 2. In addition to the typical distribution of defects described below, Table 2 also lists the probability of successful repair with the row and column redundancy approach used in the SRAM compilers ("RA1/RA2" column), and the single I/O approach used in the Register File compilers ("RF1/RF2" column).

Seven common single-point defects are considered in decreasing order of general likelihood: failure of a single bit cell, failure of a horizontally or vertically adjacent pair of bit cells (for example, due to failure of a shared contact), complete or partial failure of an entire physical column (for example, due to a short to ground of one of the bit lines), failure of a single row (for example, due to a short with a horizontal power strap), failure of a pair of columns (for example, due to a short between bit lines of adjacent columns), and failure of a pair of rows (for example, due to a faulty word-line driver).

As memory size increases, it becomes possible that a memory failure results from more than a single point of failure. The second half of Table 2 lists some examples of combinations of failures that could result from pairs of events listed earlier. For example "F and (D or F)" is the combination of a paired column failure with either a single column or another paired column failure. Such a failure cannot be repaired in mux4 designs because it will always effect multiple bits in a word, but has a 50% probability of being repaired in mux 8, and so forth.

Table 2. SRAM Defect Matrix

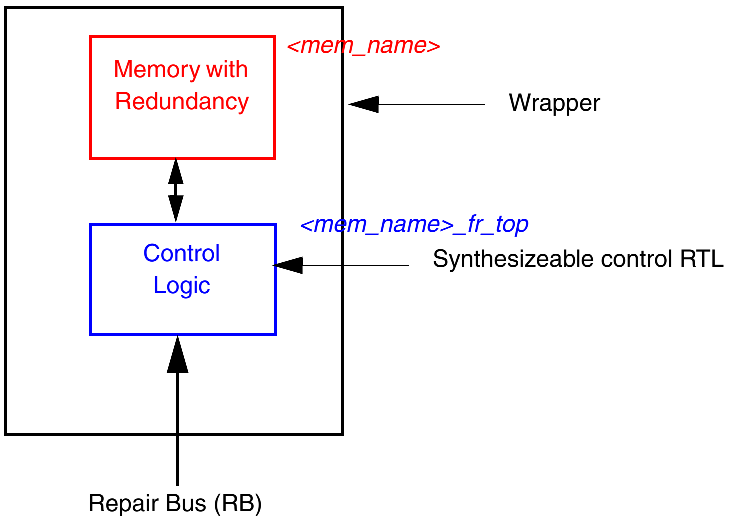
Defect	Designator	Typical Distribution (%)	RA1/RA2 (%)	RF1/RF2 (%)	Note
Single bit only	A	45	100	100	
Paired bit horizontal only	B	13.5	100	K	1
Paired bit vertical only	C	13.5	100	100	
Single column	D	9	100	100	
Single row	E	4.5	100	0	
Paired column	F	<4.5	100	K	1
Paired row	G	<0.9	100	0	
(A, B, C, or E) and (A, B, C, D, or E)	---	8.2	100	L	2, 3
(D or F) and (A, B, C, E, or G)	---	1.5	100	0	2
G and (A, B, C, D, or F)	---	0.1	100	0	2
D and D	---	0.1	M	0	2, 4
F and (D or F)	---	0.1	N	0	2, 5
G and E	---	<0.1	0	0	2
Three or more independent defects	---	---	---	---	6

Notes:
1. K = 0% for mux1, 50% for mux2, and 75% for mux4
2. For realistic yield values, two independent defects in a single array are significantly less likely than one.
3. L = 1/bits
4. M = 50% for mux4, 75% for mux8, 87% for mux16, and 95% for mux 32
5. N = 0% for mux4, 50% for mux8, 75% for mux16, and 87% for mux 32
6. Three or more defects in a single array are very unlikely but some combinations are still repairable.

4.2 Register File Flex-Repair/Redundancy

The main components of the high density single-port Register File Flex-Repair solution are provided in Figure 1. Component definitions are provided below the diagram. The memory compiler creates the repair RTL in Verilog and VHDL languages that combine the different components. You can use either, depending on your RTL design language.

Figure 1. Components of High Density Single-Port Register File Flex-Repair



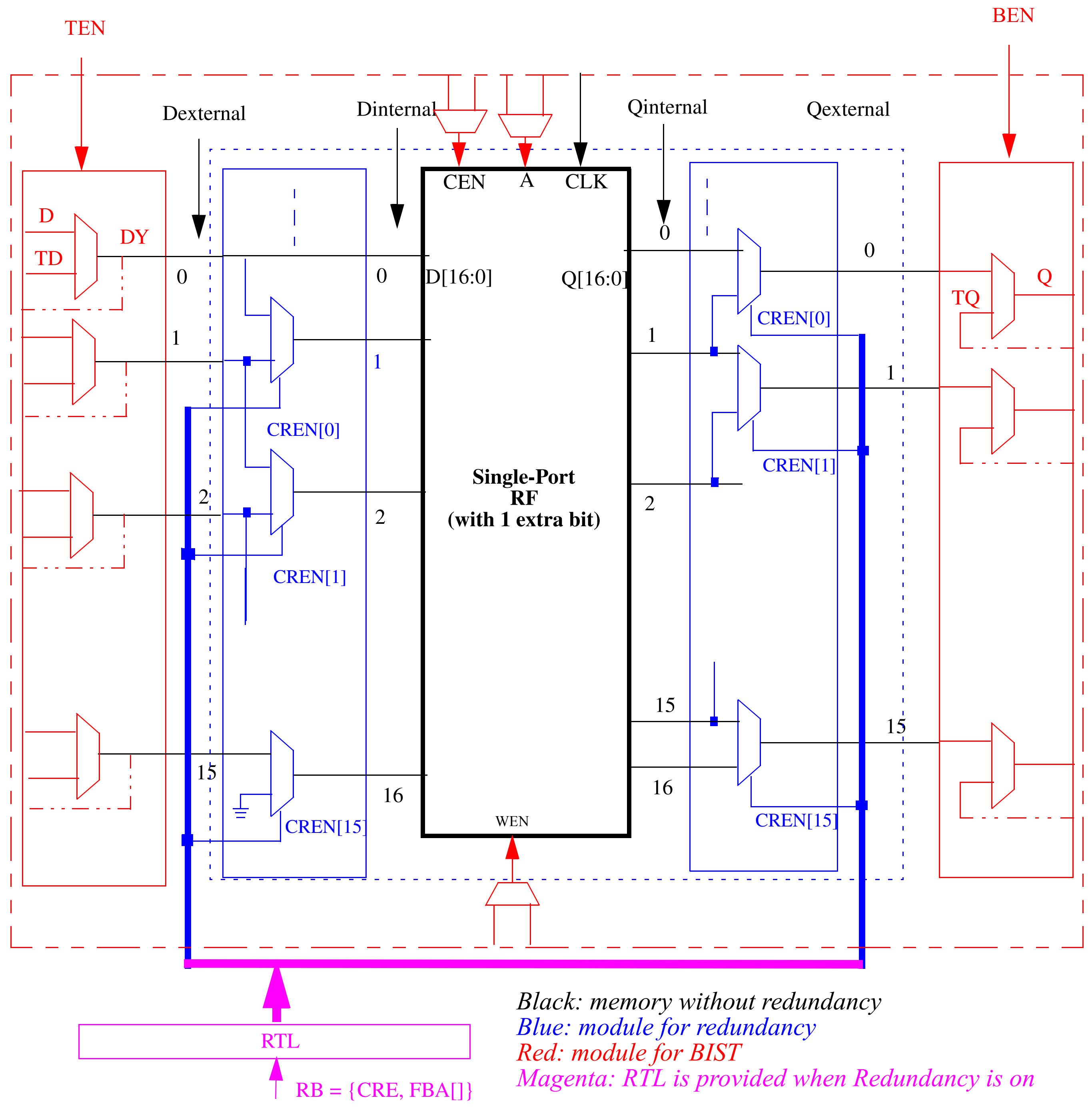
- **Memory:** The single-port register file compiler creates a memory that includes the total number of bits per word (the sum of the number of user-defined bits and the number of redundant bits).
- **Fuse Box:** The fuse box is required if you are using the laser repair methodology. The fuses are used to store the faulty bit location.
- **Control RTL:** The RTL code describes the logic for replacement of faulty columns by redundant columns during chip operation, as defined by the fuse programming. The logic is implemented in standard cells by synthesizing the RTL code. The components of the repair solution are contained within a wrapper module. The Repair RTL for this control block is described in "Repair RTL Description" on page 9.

Note

See the “Using the Repair VHDL Model” and “Using the Repair Verilog Model” sections in the memory compiler user guide for information about using ARM’s Artisan-provided repair models.

The detailed block diagram shown in Figure 2 provides a block-level schematic of the repair RTL wrapper.

Figure 2. Block Diagram with High Density Single-Port Register File Flex-Repair



4.3 RTL-Supported Fault Categories

RTL in the high density SRAM and Register File Flex-Repair memory compilers support repair for the following fault types:

- A random, single bit within the memory array fails, such as when a contact is not formed.

- An entire physical column fails, such as when a short occurs between bit line wires.

- Adjacent bits or columns within the same bit-column fails, such as multiple bit line shorts.

These memories can be repaired whether one fault or combinations of faults occur in a *single* bit column. See "Sample Manufacturing Flow for Laser Repair" on page 22 for more information.

The RTL provided with this memory can also be used as an example for creating RTLs to implement other types of repair. For example, you could create either row or single-bit repair using extra storage within the memory. You could also create a repairable memory using storage external to the memory.

4.4 Repair RTL Description

Repair RTL creates the control logic that redirects access from the faulty bit to the redundant bit. The logic is synthesized into standard cells. As shown in Figure 1, the RTL also has an enclosing cell, or wrapper, that instantiates the memory and the control logic.

4.4.1 Faulty Bit Locations

This section describes how the control block uses the Repair Bus (RB[ ]) pins to repair the memory for fuse based repair scenarios. The RB[ ] pins are connected to the fuse box. For BISR, the bus is connected to the repair register.

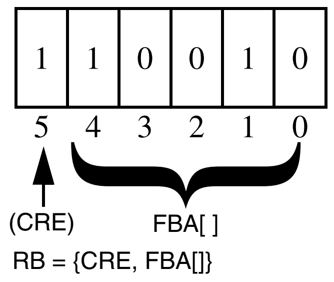
When using fuse box based repair, you can use the fuse box compiler to create a fuse box instance with a desired number of fuses. The memory bit location represented by the fuse settings is replaced by the redundant bit.

The faulty bit location should be stored in *encoded* format with the fuse box or the repair register. Both types of faulty bit location formats are shown in Figure 3.

Note

The outputs from the fuse box are logic "0". When a fuse is blown, the corresponding output is set to "1".

Figure 3. Faulty Bit Location for a 32-Bit SRAM



Encoded

Figure 3 shows that the FBA inputs to the control logic at bit location 18 of a 32-bit memory [bits 0 -31] is faulty, for *encoded* locations.

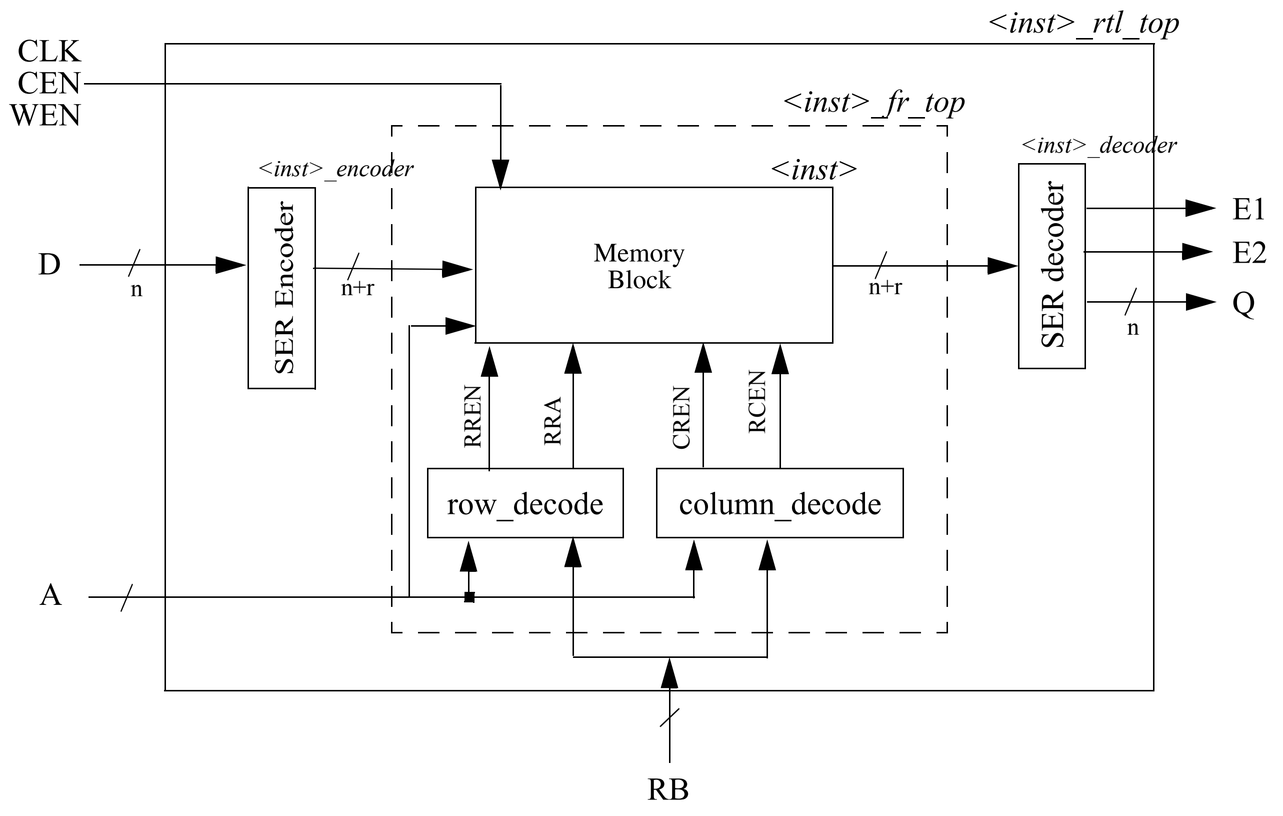
In the *encoded* format, the faulty bit location is binary encoded. For example, a memory with 32 bits needs a 5-digit binary number to represent the faulty bit location (round up $\log_2 32$). The *encoded* format requires one additional input signal to indicate whether repair is required or not. If there are no faults within the memory, this signal should be "0", or low.

If the memory needs to be repaired, this signal CRE should be set to "1", or high. Within the control logic, this signal is the MSB of the RB[ ] inputs. If you are using fuse programming, the fuse corresponding to the RB[MSB] should be blown. For BISR, the corresponding bit in the repair register should be set to "1".

4.5 Repair and SER RTL

The SRAM and register file compilers create the RTL when SER or Redundancy are enabled. The block diagram for the RTL is shown in Figure 4.

Figure 4. Typical Repair and SER RTL



4.5.1 Repair RTL

RTL has a top level module called "`<instance_name>_rtl_top`". An extra repair bus, RB, is added to the interface.

Note

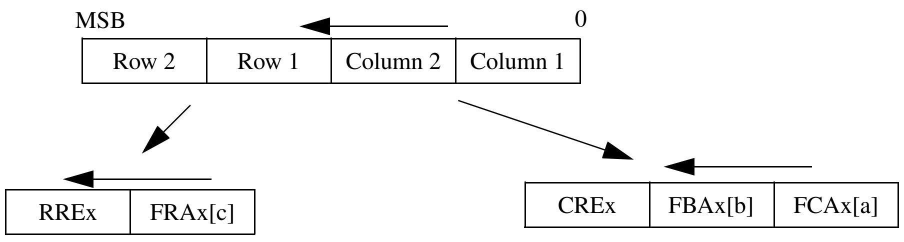
The dual-port SRAM has two sets of repair pins on the memory block for redundancy, even though a similar fault is seen by both ports.

The same repair bus, RB, is used for the top level RTL wrapper that encodes the fault locations. However, memory block has redundancy pins for each port, for example, CRENA, CRENB. The logic that determines if the currently accessed address is faulty resides in the RTL. This logic enables the redundancy pins on the memory block whenever the faulty address is detected. Since two ports of a dual-port SRAM can access different addresses, we need two different sets of repair pins: one for each port.

4.5.1.1 Repair-Bus (RB) Encoding

The repair bus, RB, represents all types of faults the specific memory instance can repair. Repair RTL uses the following bits in the repair bus. This information is necessary when enabling the repair.

Figure 5. Repair Bus Encoding for Instance With Two Rows and Two Columns



a: $\log_2$(MUX) number of fuses needed to select a column in a MUX
b: $\log_2$(bits), number of fuses needed for selecting one of a user-bits
c: $\log_2$(words/MUX), number of fuses needed to select a word-lines

4.5.1.2 Row Redundancy

- •RRE1 (RRE2): Row redundancy enable for 1st (2nd) redundant row, if set to logic 1 then row redundancy is ON for 1st (2nd) row.
- •FRA1 (FRA2): Faulty row address for 1st (2nd) redundant row. This has the logical address of the row.

4.5.1.3 Column Redundancy

- •CRE1 (CRE2): Column redundancy enable, if set to logic 1, then 1st (2nd) column redundancy is ON.
- •FBA1 (FBA2): Faulty Bit Address for 1st (2nd) column, this bus encodes the index of the BIT in the faulty word.
- •FCA1 (FCA2): Faulty Column Address for 1st (2nd) column, this bus encodes the logical address of the faulty column in the faulty bit.

4.6 BIST MUXes

Embedded memories exist as part of an overall SoC design and, as a result, it is not sufficient for memory Design for Test (DFT) features to support only memory testing. To provide an interface with industry standard BIST and ATPG tools, the BIST MUXes option was added to the high density SRAM, register file, and ROM compilers. Figure 6 shows a block diagram of the BIST MUXes.

Figure 6. BIST MUXes Block Diagram for Single-Port SRAMs

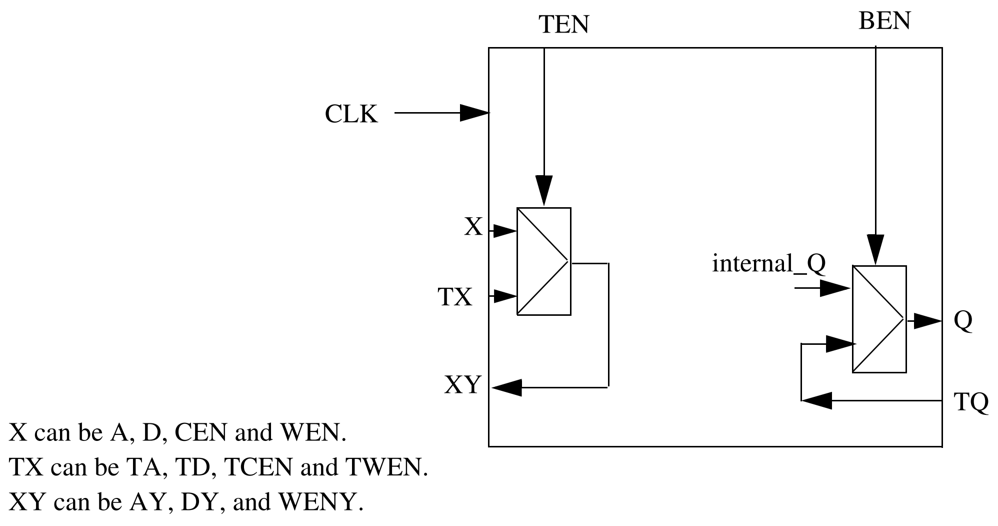

TEN
BEN
CLK
X
TX
XY
internal_Q
Q
TQ
X can be A, D, CEN and WEN.
TX can be TA, TD, TCEN and TWEN.
XY can be AY, DY, and WENY.

Table 3 provides pin descriptions for the BIST MUXes.

Table 3. BIST MUX Pin Descriptions for Single-Port SRAMs

Name	Type	Description
A[m-1:0]	Input	Addresses (A[0] = LSB)
D[n-1:0]	Input	Data inputs (D[0] = LSB)
CEN	Input	Chip Enable, active low
WEN [*]	Input	Write Enable, active low. *If word-write mask is enabled, this becomes a bus.
CLK	Input	Clock
Q[n-1:0]	Output	Data outputs (Q[0] = LSB)
TEN	Input	Test Mode Enable, active low. 0=Test operation, 1=Normal operation
TA[m-1:0]	Input	Test Address (TA[0] = LSB)
AY[m-1:0]	Output	Address MUX output (TA[0] = LSB)
TD[n-1:0]	Input	Test Data inputs (TD[0] = LSB)
DY[n-1:0]	Output	Data MUX output (TD[0] = LSB)
TCEN	Input	Test Chip Enable, active low
CENY	Output	Chip Enable MUX output
TWEN [*]	Input	Test Write Enable, active low. *If word-write mask is enabled, this becomes a bus.
WENY[*]	Output	Write Enable MUX output. *If word-write mask is enabled, this becomes a bus.
BEN	Input	Bypass Mode Enable, active low. 0=Bypass operation, 1=Normal operation
TQ[n-1:0]	Input	Bypass Q input, in write mode. If BEN=0, Q[n-1:0]=TQ[n-1:0].

Address, data input, chip-enable, write-enable signals get MUXes at their inputs, which are controlled by a single test enable signal, TEN. The memory output has a MUX to bypass the memory, which is controlled by a bypass control signal, BEN.

The BIST engine can create its own BIST MUX collar. However, as these multiplexers are external to the memory, normal data timing can be uncertain. Embedding the multiplexers in the memory itself achieves the following goals:

•Delay associated with test in the normal data path is minimized (approximately 100ps is added to overall access time), and uncertainty due to variable placement of test multiplexers is eliminated.

•ATPG-based path delay testing of logic and wires leading up to and away from the memory physical boundary is enabled (not possible when muxes are placed elsewhere).

•Input and output portions of the MUXes are controllable independently, allowing for advanced debug capability (for example, functional load followed by scan unload, etc.).

•ARM is working with industry standard BIST vendors to ensure that their automation tools can take advantage of these MUXes.

While MUXes simplify the testing process, they will not make sense in every design, and hence have been made optional. Some of the costs of using them include additional memory area, about 1.5% on a 64k SRAM block. Also, additional wiring is needed (the number of ports on an SRAM increases by nearly 3 times), leading to potential congestion around small memories and to larger routing channels for memories located far from the standard cell logic driving them.

This logic differs from scan wrappers proposed in the IEEE P1500 standard. Most memory BIST algorithms require parallel access to the memory ports, rather than serial. Also, the overhead of a register bit in addition to the muxes already present would be excessive for small memories. If such a wrapper is required, designers are free to add it separately, outside the physical instance of the memory.

The following procedures are to be followed to access website listed BIST algorithms:

1. Login to http://www.artisan.com
2. Click on "My Account" on the left hand side
3. Click on "Browse The Knowledge Base" in the center column
4. Click on "Technical FAQs" from the right hand side
5. Click on "DFM and Test" from the right hand side
6. Click on the individual questions to access the appropriate algorithms

5. Extra Margin Adjustment (EMA)

All ARM Artisan memories go through a rigorous margining process as part of their design. The memories will operate correctly with any EMA setting across their specified PVT (process, voltage, temperature) range. There can be cases, however, where this is not enough, and extra margin would be helpful. Examples include process excursions beyond the SPICE model, excessive IR drop to the memory, and localized temperature hot spots. Such situations can usually be corrected in time for volume production, but in early chip bring-up and debug it is helpful to have the capability to add extra memory margin to compensate for the environment change.

Extra Margin Adjustment provides the option of adding delays to the internal self-timing pulse. This delay provides extra time for memory read and write operations by slowing down the memory access. There are three input pins, named EMA[2], EMA[1], EMA[0], for each instance. The access time and cycle times are progressively increased as the pins are driven from 000 to 111 respectively. The EMA[2:0] pins are always visible. Margin sequentially increases as EMA sequentially increments from 000 through 111. Setting 000 is the fastest setting and 111 is the slowest setting. The minimum EMA setting for given operating range will be documented in the model .lib file.

The actual amount of extra margin (and increased access time) is compiler dependent, but is on the order of 200ps from setting 000 to 001. For single voltage applications, providing the ability to switch from one EMA setting to a higher one acts as a sort of insurance against unexpected deviations that put the memory outside its margined range. Because access time is increased when using EMA, it will often be necessary to increase the clock period by a similar amount.

5.1 EMA options

5.1.1 Permanently set

Tying the EMA pins to a fixed value (for example, all EMA pins tied to GND) ensures that the memory will always power up to the desired EMA value, but this value cannot be changed without physically altering a chip (for example, with a FIB during debug or bring-up, or through mask changes in production). Because of EMA, primary use is as a debug feature. Providing easy FIB access to the EMA signals is often a simple solution. This could be accomplished, for example, by routing an EMA bus to all memories, and driving the bus bits with a tie cell, providing a single point where it could be changed.

5.1.2 Programmable

This is the most flexible approach, and provides the greatest coverage of contingencies, but it requires careful design. If the EMA pins are driven from a register, then they may be changed by whatever means are used to access the register (JTAG, software, etc.). Care must be taken during design to ensure that the register will be set correctly on power up and during production test, and that the mapping between registers and memories is well understood (for example, does a single register control a single memory, a group of memories, or even all memories).

5.1.3 Hybrid

It is also possible to combine permanently set and programmable approaches, by having two values selectable (for example, 000 and 001, 000 and 111, or 001 and 010) via a register, a mux, or some combination. This method can reduce the wiring and register cost of programmable methods, while retaining some flexibility. Again, care must be taken during design to ensure that power-up behavior is correct.

5.2 Retention

Retention applies to SRAM and Register File compilers only. Retention testing is performed in order to verify that the two P transistors in a bit cell are present and functioning. If one is missing, the resulting fault is called an *asymmetric fault*. If both are missing (for example, due to a missing VDD contact), then the fault is called *symmetric*. It is important to check for retention faults, because a cell without P transistors can operate correctly over a short period of time due to the charge storage on node capacitance.

The most common retention test method is simply to load the memory with a known data value (checkerboard), wait for some period of time (this is empirically determined and can be up to hundreds of milliseconds), read the data from the memory, and then repeat with the inverse value.

It might be argued that increasing background leakage currents should allow for reduced retention test time as technology advances, but the data do not support this. While the fast process leakage increases dramatically for smaller process geometrics, the typical leakage increases more slowly, and the slow leakage is somewhat constant. Current-based (IDDQ) testing must consider the fast process corner, but retention testing needs to focus more on the slow corner. It can be seen that the ratio between fast and slow is increasing dramatically as transistors shrink, and while this makes memory design more complex, it cannot be used to reduce data retention testing time.

5.3 Soft Error Repair (SER)

As circuit geometries continue to shrink, they become increasing susceptible to “soft errors,” which is data loss caused by external radiation. The radiation can come from numerous local sources such as alpha particles emitted by solder or from the outside environment (for example, high energy neutrons from cosmic rays).

Although Artisan SRAM memories are designed to withstand soft errors, the SER feature was created for applications that require extra soft error protection. The SER feature applies to only 90nm single- and dual-port SRAM compilers.

The SER adds extra bits per word into the memory block automatically and creates accompanying RTL logic needed to implement error check and correction (ECC). RTL enables a single bit correction, and single or double bit error detection Hsiao code [11] on one word at a time in the memory. Ordinarily, the largest number of physical columns that are generated is 1024 (1026 with redundancy), but with this option additional columns are generated to accommodate the code bits ($\log_2 n + 2$ for single correct, double detect, where n is the number of bits in the word). The combination of extra memory and logic area typically increases by 20-30%, while cycle time typically doubles.

SER has two options:

- Single bit error correction and single bit error detection
- Single bit error correction and double bit error detection

These two options are usually adequate for most applications. For designs where higher performance or lower area is necessary, other approaches that make use of the memory compiler are available commercially.

5.3.1 SER RTL

When a single error is detected, RTL sets a signal pin called E1 to high. The data on RTL's output is corrected: you must decide if the corrected data needs to be stored back in the memory using another write operation. That may be useful if the same word may be read long afterwards without being overwritten with new data. In its absence, another SER error on same word may make the word unrepairable.

Pin Interface for the RTL is shown in Figure 4.

- •E1: Output: present when SER is enabled. It is normally at logic-0. When a single bit or two bit error is detected (when 2bd1bc is selected), it is set to logic-1.
- •E2: Output: present when SER is selected with two bit error detection, when 2bd1bc is selected. It is normally at logic-0. When a two or more bit error is detected, it is set to logic-1.

In an unlikely case of a large number of errors in a word, user-bits and code-bits can change in such a way, that the coding scheme may think that no error has occurred.

The SER has the following limitations:

- •The SER is not supported when word-write mask is enabled.
- •The user-bits and code-bits must meet the memory block-size limitations. In other words, SER is not supported on the bits/word, as it is at the higher-end of the range for that MUX value.

6. Additional Requirements for Memory Repair

This section provides additional requirements for implementing a memory repair using the Flex-Repair solution.

- Select a distinct name for each fuse box that contains a different number of fuses, in order to avoid conflicts when instantiated in a design. Select an encoding option and fuse box inclusion options to match your design plan.

- If using laser fuse repair, create a fuse box with the distinct name you determined for the memory instance and the correct number of fuses. See the "Sample Manufacturing Flow for Laser Repair" on page 22 to determine how many fuses are needed.

- Instantiate the Repair RTL top cell named <*`inst_name`*>*`_top`* inside your netlist. You must instantiate the fuse box or the repair register and make the necessary connections to the repair bus (RB).

- Link the fuse box models and memory models for the rest of the design steps.

- In addition to laser fuses, some foundries may offer an e-Fuse. You can discuss these fuses with your foundry representative to determine if these would be beneficial for your chip production.

6.1 Sample Manufacturing Repair Flow for Laser Repair

For repair, *each* memory with redundancy should be tested for faults. Depending on the results, one of the cases listed below will apply. See Figure 7 for a high-level manufacturing flow for fuse-based memory repair.

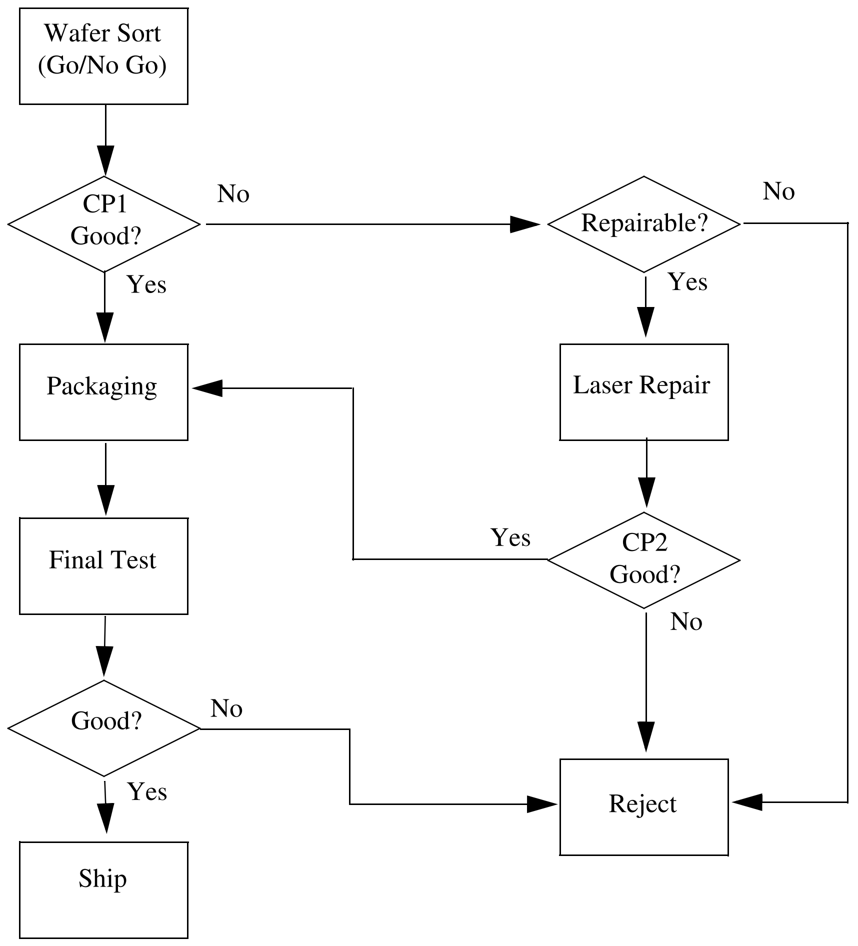
- All single-port SRAMs are working properly.

- One or more single-port SRAMs are faulty. Determine which of the following cases apply:

-Are all the faulty single-port SRAMs repairable? If the number of faulty bit-columns is greater than the number of redundant bit-columns, the single-port SRAM cannot be repaired.

-If all faulty single-port SRAMs are repairable, the chip can be repaired by blowing the appropriate fuses to replace the faulty bit(s) with the redundant bit(s).

Figure 7. Sample Manufacturing Flow for Laser Repair



Appendix A- Revisions

This appendix describes the technical changes between released issues of this book.

Table A-1. Issue A

Change	Location
Replaced Metro product name with High Density	Entire book