Redundancy Analysis 11

Introduction

ECR devices with both redundant rows and columns use redundancy analysis to determine whether replacing a row or a column would conserve redundant elements. These devices also include segmented devices, devices with multiple-row or multiple-column redundant elements, and devices with redundant elements shared between segments. You can set up a preference for either row or column repair, exclude one type of element, regardless of efficiency, and restrict any part of the device for partial testing and repair; refer to *Flags* on page 11-15. After a test pattern finishes executing, the redundancy algorithm analyzes the contents of the Error Catch RAM and returns to the user a list of replaceable rows and columns for repairing the device. Note that the redundancy algorithm does not actually make the repairs.

FLASH750 also supports parallel redundancy analysis because each ECR has its own redundancy processor. Each FLASH750 system holds up to 32 ECRs.

Topics:

Principles of FLASH750 Redundancy Analysis

FLASH750 redundancy analysis consists of two operations:

- Redundancy algorithm generates a list of must-replace rows and columns, assigning the necessary redundant elements to replace them.
- Redundancy algorithm tries to cover any remaining bad locations with the remaining spares.

Each segment in the defined analysis region is analyzed, and repairs are assigned. Or, if a segment is unrepairable, the redundancy analysis routine returns immediately.

+ Unless otherwise specified in the test program, the redundancy algorithm considers columns first. If the test program specifies rows first, the terms column and row in the following description should be reversed.

The redundancy algorithm begins by scanning the columns of the device. Starting with column 0, it scans each column whose column error RAM has one or more errors, looking for the rows failing that column.

As a failing column is scanned, each failing row is recorded in a bad-points list. As each failing row is added to the list, the redundancy algorithm verifies the additional point and makes the column a must-replace column. A must-replace contains more errors than the number of spare rows available. When a column is replaced, the column and all failing rows in that column are removed from the bad-points list.

The redundancy routine considers a repair made once it has recommended the repair.

As each failing column is scanned, a consecutive-column-failures counter is incremented. This counter is reset when a non-failing column is skipped. Non-failing columns are not scanned.

After each failing column is scanned, the consecutive-column-failures counter is compared to the number of spare columns. If the consecutive-column-failures counter is greater than the number of spare column, the bad-points may contain a must-replace row. The algorithm examines the bad-points list for a must-replace row; if it finds one, all occurrences of that row in the bad-points list are removed.

After redundant elements have been assigned to all must-replace rows and columns, one of three conditions exists:

1. No bad locations; segment is repairable; this segment analysis is completed.

2. Bad locations; no spares remain; segment is unrepairable.
3. Bad locations; spares remain; repair pattern plan must try to finish.

In the third case, the second phase of the analysis reads the bad-points list and finds the row with the most column failures and the column with the most row failures. Of those two, the one with the most errors is replaced first, then the other. If the worst row and column have the same number of errors, replacement depends on user preference and availability of spares. This phase is repeated until all spares in the segment or all errors have been covered.

If redundancy analysis finds an unrepairable segment, it returns immediately, even if some segments have not been examined.

If all segments are repairable, redundancy analysis returns the rows and columns to be replaced in two arrays—srb and scb (Segment Row Buffer and Segment Column Buffer) in C++ or by calling the **tl_EcrGetRepairResults()** in **VB**.

Redundancy Analysis Process

The redundancy analysis process in the following examples assumes:

- 16-by-16 device with 4 redundant column elements and 2 redundant row elements.
- A functional test finds 14 errors, indicated by the black diamond symbols in the following diagrams.
- Test program specifies that columns are repaired first.

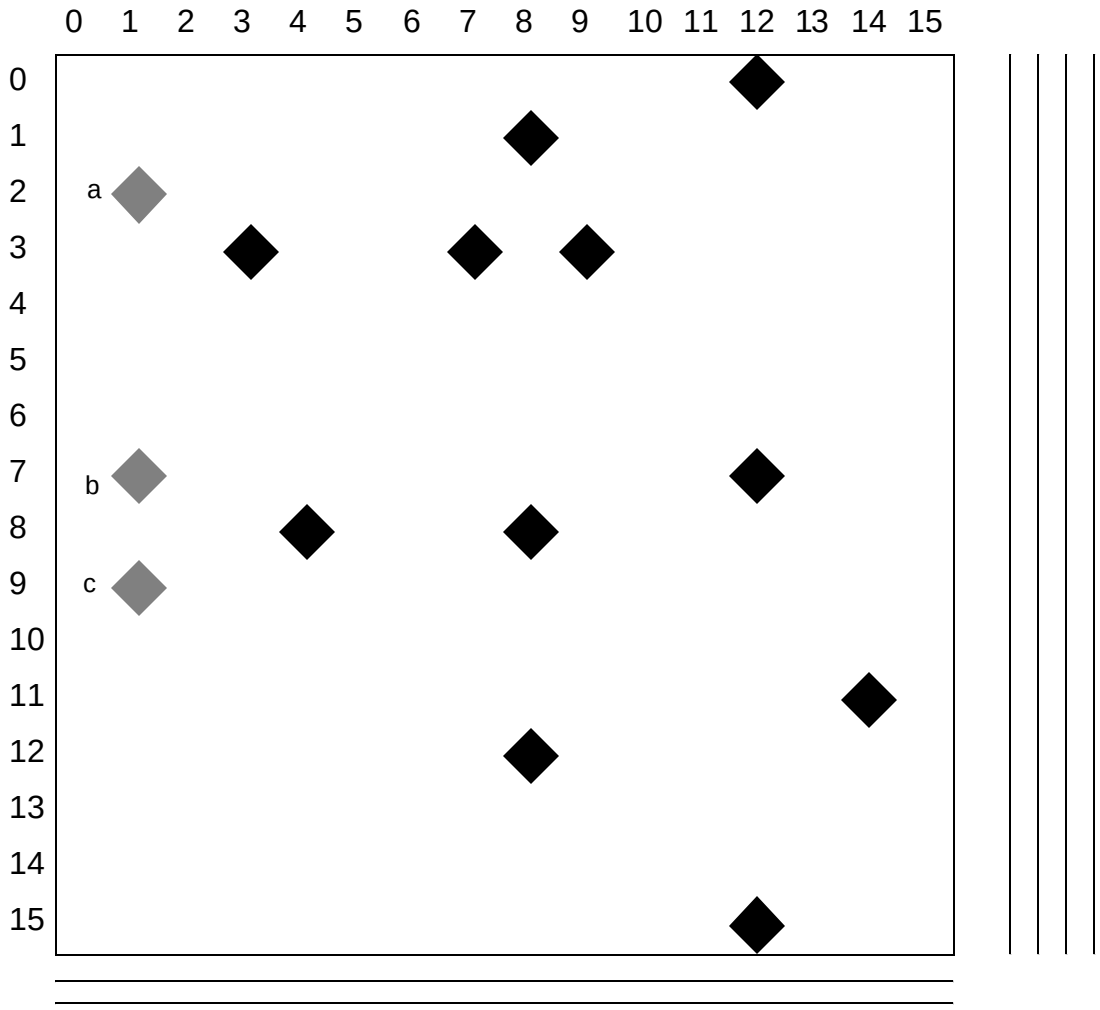


Figure 11-1 Redundancy Analysis Begins, 16-by-16 Device

Redundancy algorithm begins analyzing the device shown in Figure 11-1 on page 11-4. Starting with column 0, it scans each column whose column error RAM has one or more errors, looking for the rows that failed in that column.

Column 0 is not scanned, since its column error RAM indicates no errors in that column.

Column 1 is a must-replace column because its three errors (a, b, c) are more than the number of spare rows available (two).

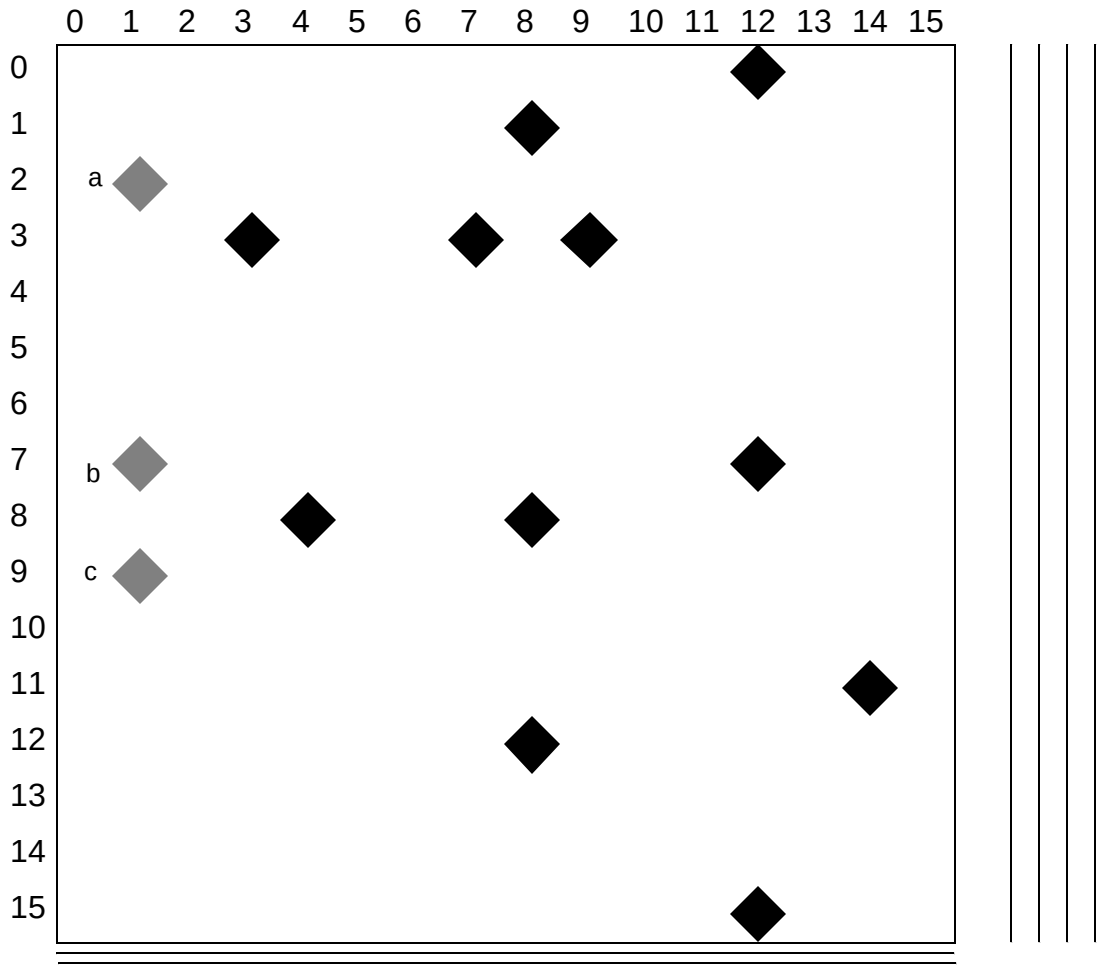


Figure 11-2 Redundancy Analysis Begins, 16-by-16 Device

One of the four spare columns is assigned to replace Column 1. As shown in Figure 11-2 on page 11-5, three spare columns remain. Redundancy analysis resumes scanning columns whose error RAMs indicate errors:

- Column 2 is skipped—its column error RAM indicates no errors.
- Columns 3 and 4 are scanned—their column error RAMs indicate errors. Column 3 has error *d*; Column 4 has error *e*. These errors are logged to the bad-points list, but they are not repaired yet.
- Columns 5 and 6 are skipped—their column error RAMs indicate no errors.
- Columns 7 and 8 are scanned. In Column 7, its column error RAM indicates an error (*f*). This error is logged to the bad-points list, but it is not repaired yet. Column 8 has three errors (*g*, *h*, and *i*) that qualify it as a must-replace column.

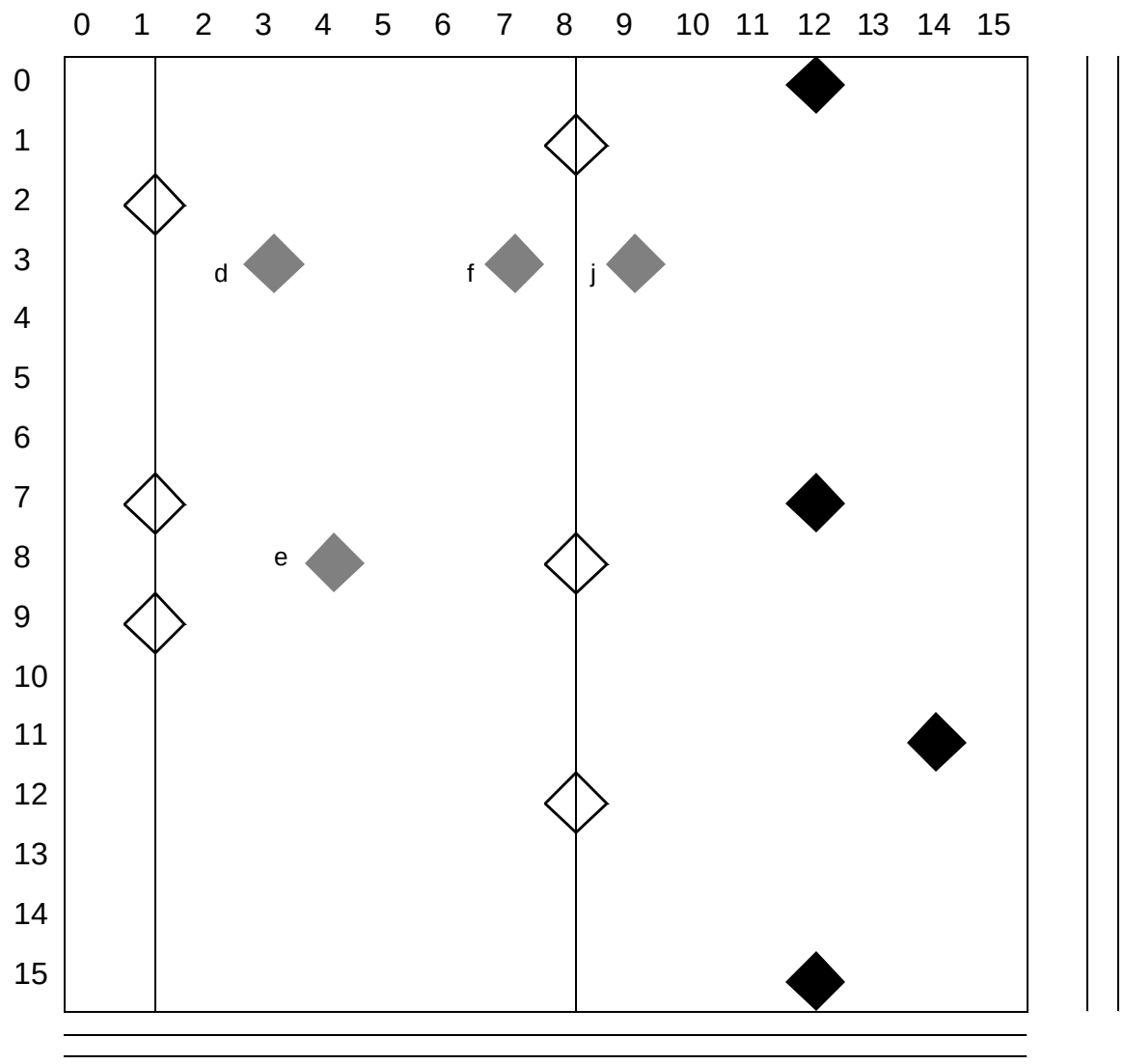


Figure 11-3 Redundancy Analysis Continues Scanning Columns, 16-by-16 Device

As shown in Figure 11-3 on page 11-6, one of the 3 remaining spare columns replaces Column 8; 2 spare columns remain.

The algorithm continues scanning columns whose error RAMs indicate errors until it reaches Column 9, which contains error *j*. Because errors have now been logged in more consecutive columns (7, 8, and 9) than there are spare row elements, the algorithm searches the bad-points list for a must-replace row. It finds row 3 has three errors (*d*, *f*, and *j*), which is more than the remaining number of spare columns; thus, it is a must-replace row.

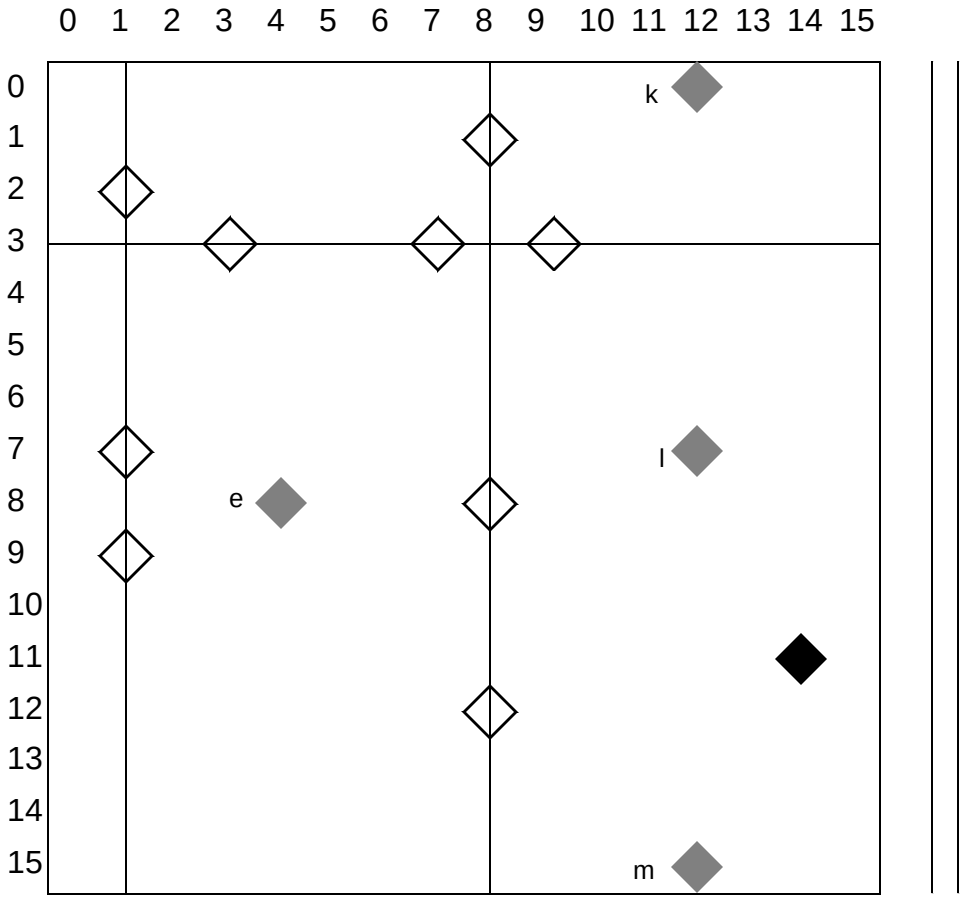


Figure 11-4 Redundancy Analysis Continues Scanning Columns, 16-by-16 Device

As shown in Figure 11-4 on page 11-7, one of the 2 spare rows is assigned to replace row 3. One spare row and two spare columns remain.

The algorithm resumes scanning columns whose column error RAMs indicate errors. Columns 10 and 11 are skipped.

Column 12 is the next column with errors (*k*, l, and *m*). Since the errors in Column12 (3) is greater than the spare rows available, it is a must-replace column.

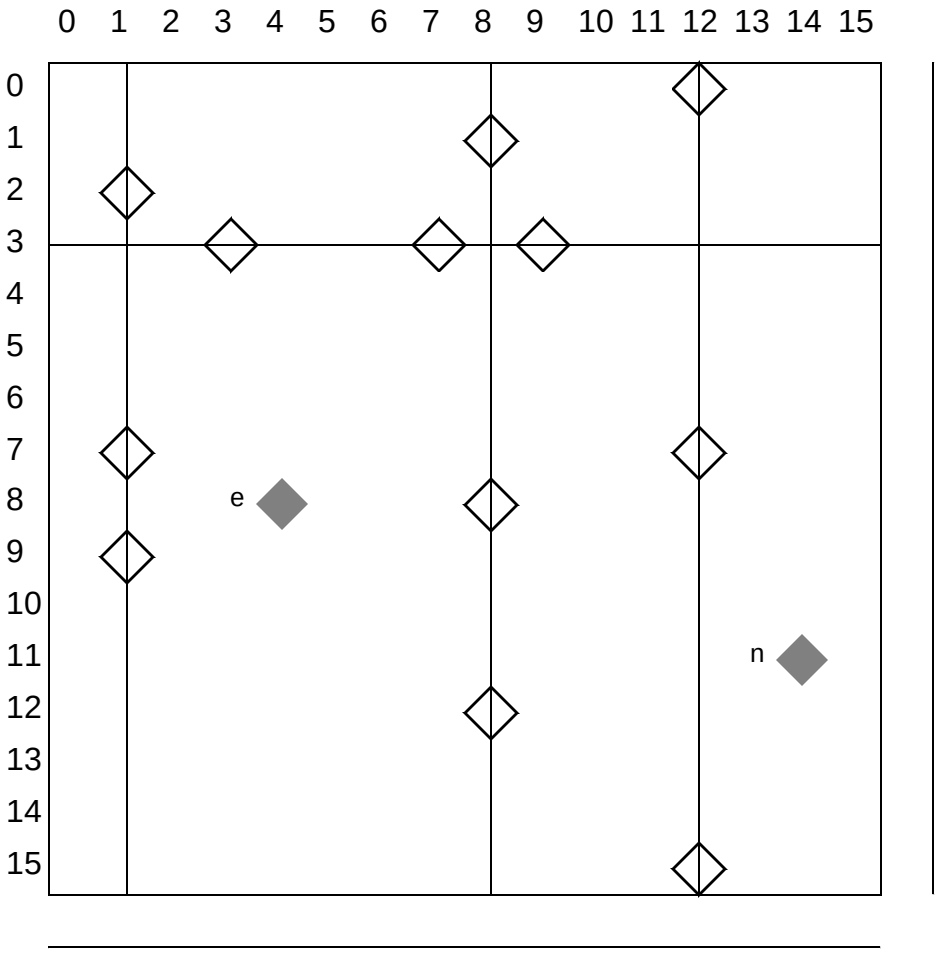


Figure 11-5 Redundancy Analysis Continues Scanning Columns, 16-by-16 Device

As shown in Figure 11-5 on page 11-8, one of the two remaining spare columns is assigned to replace Column 12. One spare column (and one spare row) remain.

The algorithm resumes scanning columns whose column error RAMs indicate errors. Column 13 is skipped. Column 14 contains an error (*n*), which is logged to the bad-points list, but not repaired yet. Column 15 is skipped.

No must-replace rows or columns are unrepaired now: two errors in the bad-points list (*e* and *n)* remain. One spare row and one spare column remain.

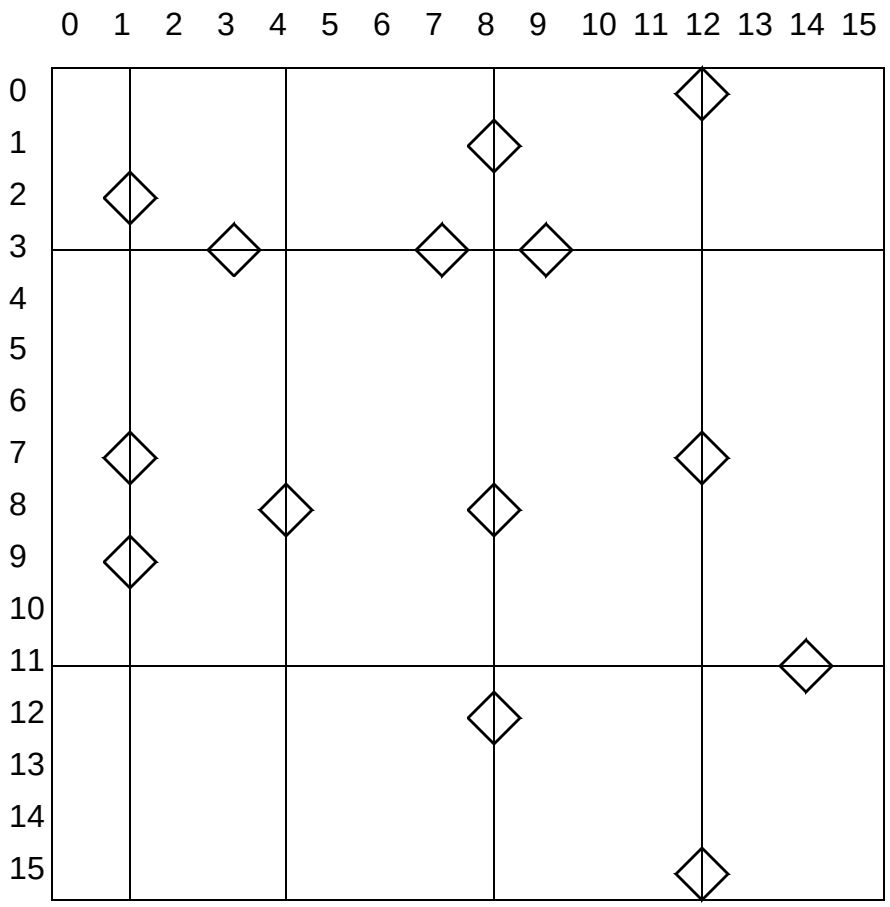


Figure 11-6 Redundancy Analysis Repair Pattern, Sample

As shown in Figure 11-6 on page 11-9, the rows and columns with the remaining bad points are equal—each has one error. Because the test program assigns repairs first to columns, error *e* is repaired by replacing Column 3, and error *n* is repaired by replacing Column 14.

The device is repaired. Figure 11-6 on page 11-9 shows the repair pattern generated by the redundancy algorithm.

Redundancy Table Sheet

Overview

To analyze error data, the redundancy analysis routine must know the device topology. Device parameters are entered by the user in the *Redundancy Table* sheet; refer to *Redundancy Sheets Settings* on page 11-11.

Redundancy Sheets Settings

- *Segment*
- *First Segment Row*
- *Last Segment Row*
- *First Segment Column*
- *Last Segment Column*
- *Data Mask*
- *Row Linkage*
- *Column Linkage*
- *Rows Differential*
- *Columns Differential*
- *First Global Row*
- *Last Global Row*
- *First Global Column*
- *Last Global Column*
- *# of Segments*
- *Number Row Spares*
- *# Used Row Spares*
- *Number Column Spares*
- *# Used Column Spares*
- *Flags;* refer to *Flags on page 11-12*
- *Comment*

For more information, refer to *Redundancy Table Sheet* on page 3-113.

Flags

- *Row Preference*
- *Physical Rows and Columns*
- *Row Results w/o Linkage*
- *Spare Col Per Data*
- *Discretionary*
- *Parity*
- *Col Results w/o Linkage*
- *Unequal R (row)/C (column) resources per segment*
- *Continue Analysis to End*
- *Spare Row Per Data*

For more information, refer to *Redundancy Table Sheet* on page 3-113.

Linkage

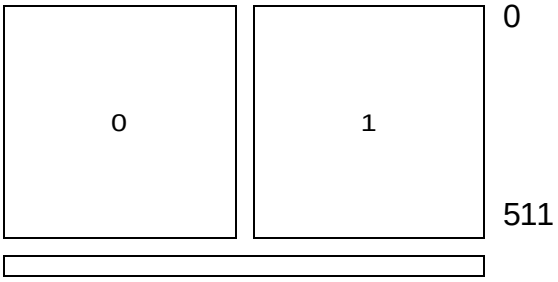
Overview

A redundant element repairing more than one segment also links those segments. Linkage is specified by the linkage options on the *Redundancy Table* sheet: *Row Linkage* and *Column Linkage*.

Two types of linkage are provided: complete and partial. The linkage variables in the redundancy table are the same for both types; the difference is the segment boundary options specified by *First Segment Row*, *Last Segment Row*, *First Segment Column*, and *Last Segment Column*.

Complete Linkage

Complete linkage means a redundant element replaces elements in more than one segment at a time. For example, the redundant row element in the device shown below is shared between both segments. If it is used for a repair, it replaces the same row address in both segments.



Redundancy Sheet Settings

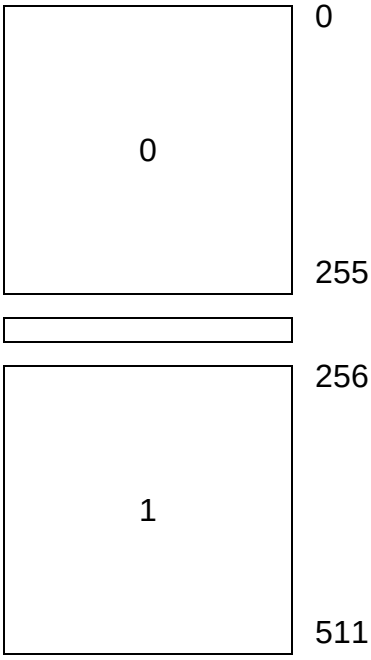
Segment 0
Row Linkage = 1
First Segment Row = 0
Last Segment Row = 511

Segment 1
Row Linkage = 0
First Segment Row = 0
Last Segment Row = 511

In the redundancy table for a device with completely linked rows like this one, the segment-boundary-first-row settings are the same for both segments. The same is true for the segment-boundary-last-row settings.

Partial Linkage

Partial linkage means a redundant element replaces elements in either segment, but not both. For example, the redundant row element in the device shown below replaces one row in segment 0 *or* one row in segment 1; however, it can be used in only one segment, not both.



Redundancy Sheet Settings

Segment 0
Row Linkage = 1
First Segment Row = 0
Last Segment Row = 255

Segment 1
Row Linkage = 0.
First Segment Row = 256
Last Segment Row = 511

In the redundancy table for a device with partially linked rows like this one, the segment-boundary-first-row variable is different for each segment. The same is true for the segment-boundary-last-row variable.

Flags

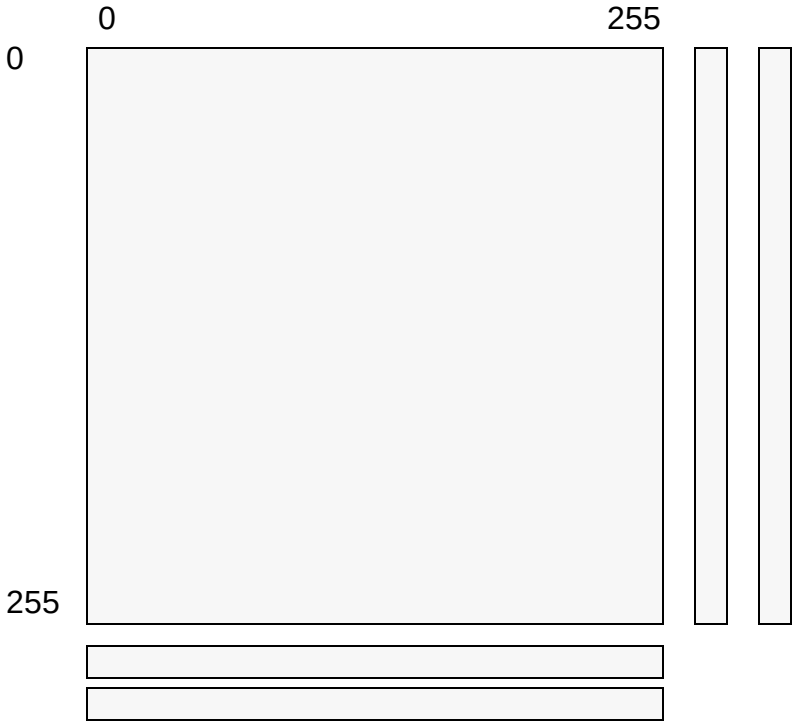
The *Redundancy Table* flags listed in the following table can be OR'ed when setting the user preferences in the *Redundancy Analysis Options* popup.

Flag	if checked	if not checked
Row Preference	Use rows first (if available) in assigning repairs.	Use columns first (if available) in assigning repairs.
Discretionary	After worst row and column are found, replace the worst, assuming both types of spares are available, regardless of the *Row Preference* selection. If worst row and worst column have same number of bad locations, replace type specified by *Row Preference.*	Row preference selection is absolute: use the specified element type (row or column) until no more are available, before using any elements of the other type except for a must-replace element, which is always replaced first.
Physical Rows and Columns	Record the physical rows and physical columns. After the analysis, the Bitmap Scramble RAMs determine the physical row to record. Example 4.	Record the logical rows and logical columns.
Parity	Use segment bit masks for non-parity and parity I/O for non-parity I/Os.	Use segment bit masks for non-parity I/Os.
Row Results w/o Linkage	Record the row of every segment containing the bad row regardless of the row linkage. Refer to *Actions and Results Returned by RT_RLNK or RT_CLNK* on page 11-45 and Example 10.	Record the row of one segment among the linked segments if more than one linked segment contains the same bad row.
Column Results w/o Linkage	Record the column of every segment containing the bad column regardless of the column linkage. Refer to *Actions and Results Returned by RT_RLNK or RT_CLNK* on page 11-45 and Example 10.	Record the column of one segment among the linked segments if more than one linked segment contains the same bad column.

Flag	if checked	if not checked
Unequal R/C Resources	Use the initial differential number of spare rows and columns for the segments for devices having unequal spares per segment; refer to Example 11.	Use the total number of spare rows and columns for devices having the same number of spares per segment; refer to Examples 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10.
Continue Analysis to End	Continue redundancy analysis despite the unrepairable segment until all segments are analyzed; refer to *Actions and Results Returned by RT_CONT* on page 11-44.	Return immediately if the redundancy analysis routine finds an unrepairable segment.
Spare Row Per Data	Limit the recommended row repair to a single segment for linked row segments.	Recommended row is repaired at a given address in a linked row segments.
Spare Column Per Data	Limit the recommended column repair to a single segment for linked column segments.	Recommended column is repaired at a given address in a linked column segments.

Examples

Example 1—One-Segment Device

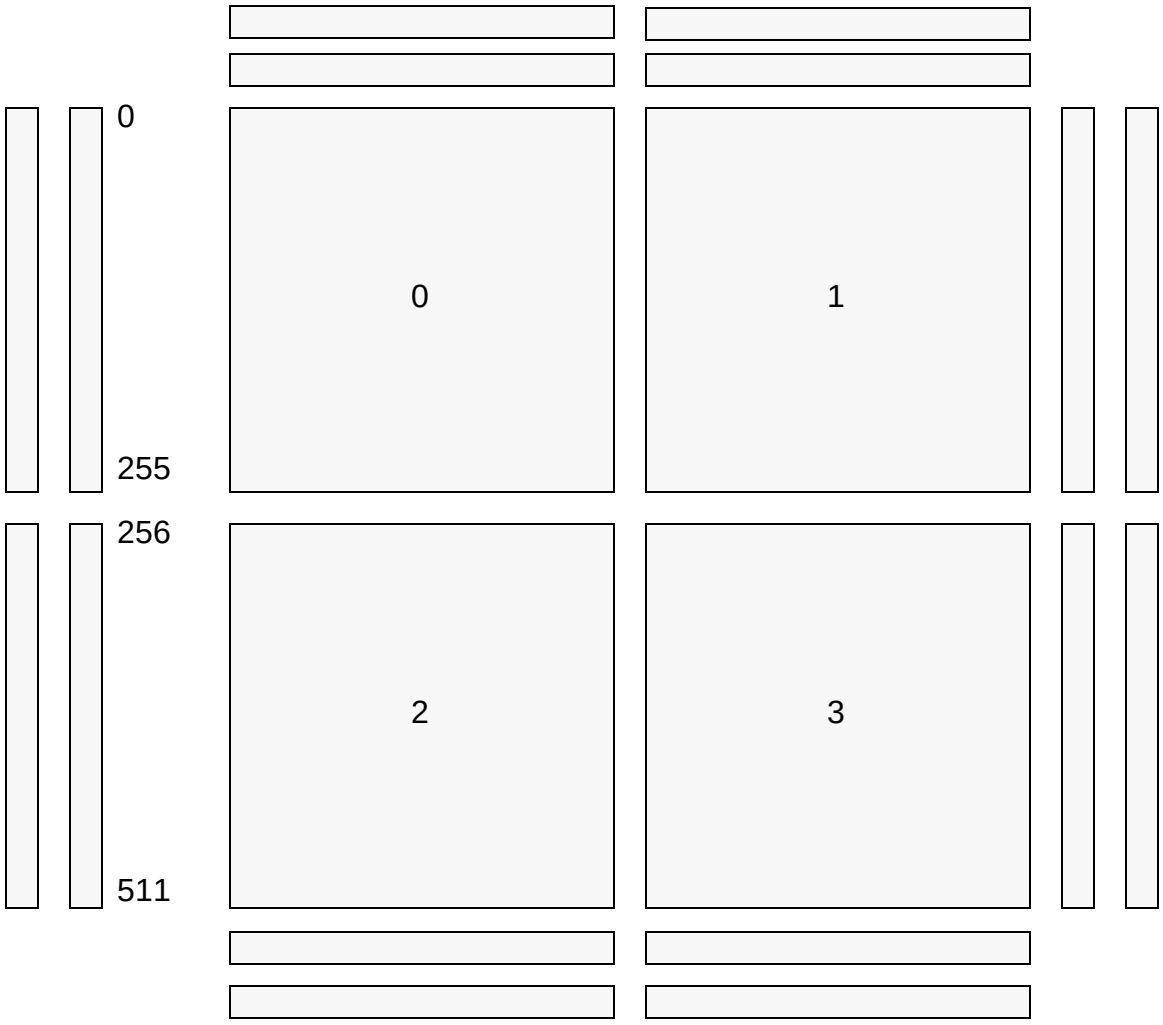


This single-segment device has independent redundant elements (not linked and not coupled) and a single-bit data output. This device has two redundant rows and two redundant columns.

Redundancy Sheet Settings

# of Segments = 1	Column Spares Number = 2
# Used Column Spares = 1	Row Spares Number = 2
# Used Row Spares = 1	Row Linkage = segment number
Column Linkage = segment number	
First Segment Row = 0	Last Segment Row = 255
First Segment Column = 0	Last Segment Column = 255
BitMask = 0x1	Global Row First = 0
Global Row Last = 255	Global Column First = 0
Global Column Last = 255	Flag checked: *Discretionary*

Example 2—Segmentation



This segmented device has four redundancy segments, each of which has two independent redundant rows and two independent redundant columns. None of the redundant elements is linked.

Redundancy Sheet Settings

of Segments = 4

Column Spares Number = 2

Used Column Spares = 1

Row Spares Number = 2

Used Row Spares = 1

Row Linkage = segment number

Column Linkage = segment number

Segments 0 and 1: First Segment Row = 0

Segments 0 and 1: Last Segment Row = 255

Segments 2 and 3: First Segment Row = 256

Segments 2 and 3: Last Segment Row = 511

BitMask = 0x1

Global Row First = 0

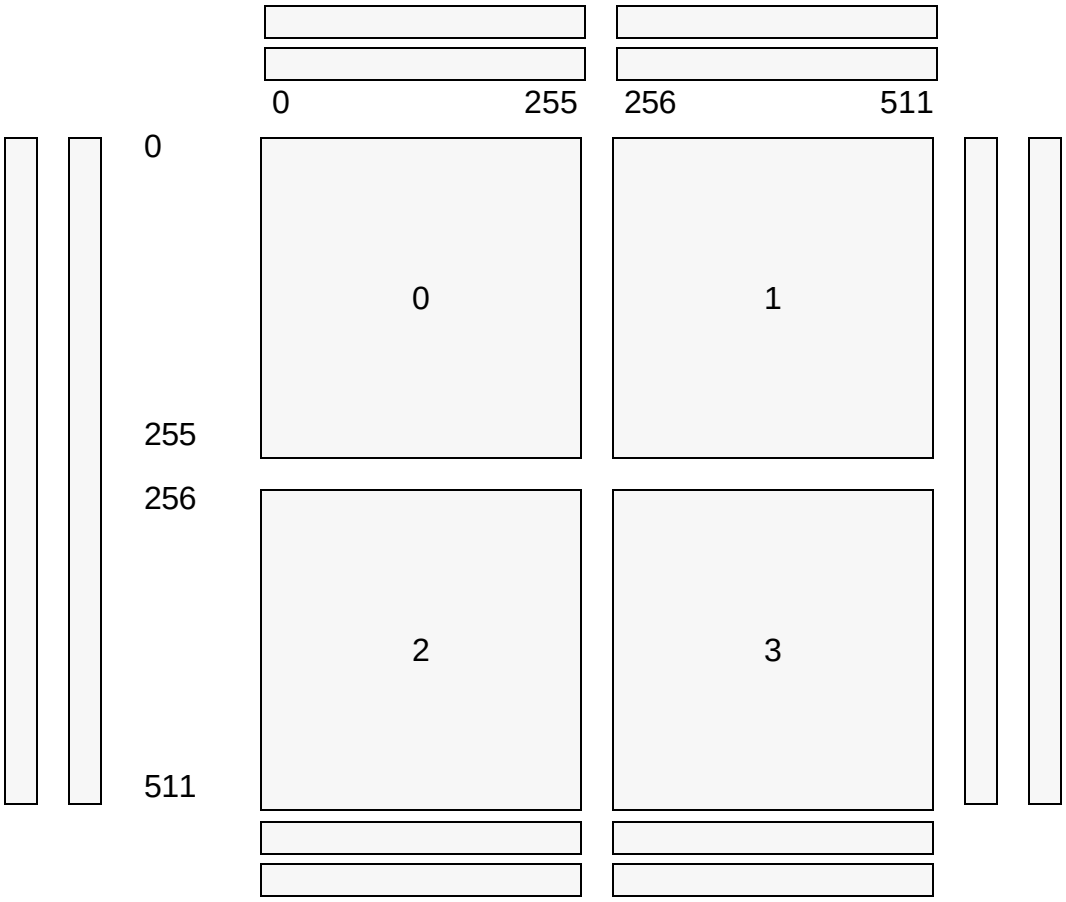
Global Row Last = 511

Global Column First = 0

Global Column Last = 511

Flag checked: *Discretionary*

Example 3—Linkage



In this example a single-data-bit device has linked redundant elements. It has four redundancy segments, each with two independent redundant rows. Each pair of segments shares two independent redundant columns—they are linked by column.

Redundancy Sheet Settings

of Segments = 4

Column Spares Number = 2

Used Column Spares = 1

Row Spares Number = 2

Used Row Spares = 1

Row Linkage = segment *x* linking to segment *x*

Column Link =Segment 0 links to segment 2, Segment 2 links to segment 0, Segment 1 links to segment 3, Segment 3 links to segment 1

Segments 0 and 1: First Segment Row = 0

Segments 0 and 1: Last Segment Row = 255

Segments 2 and 3: First Segment Row = 256

Segments 2 and 3: Last Segment Row = 511

Segment 0:First Segment Column = 0

Segment 0: Last Segment Column = 255

BitMask = 0x1

Global Row First = 0

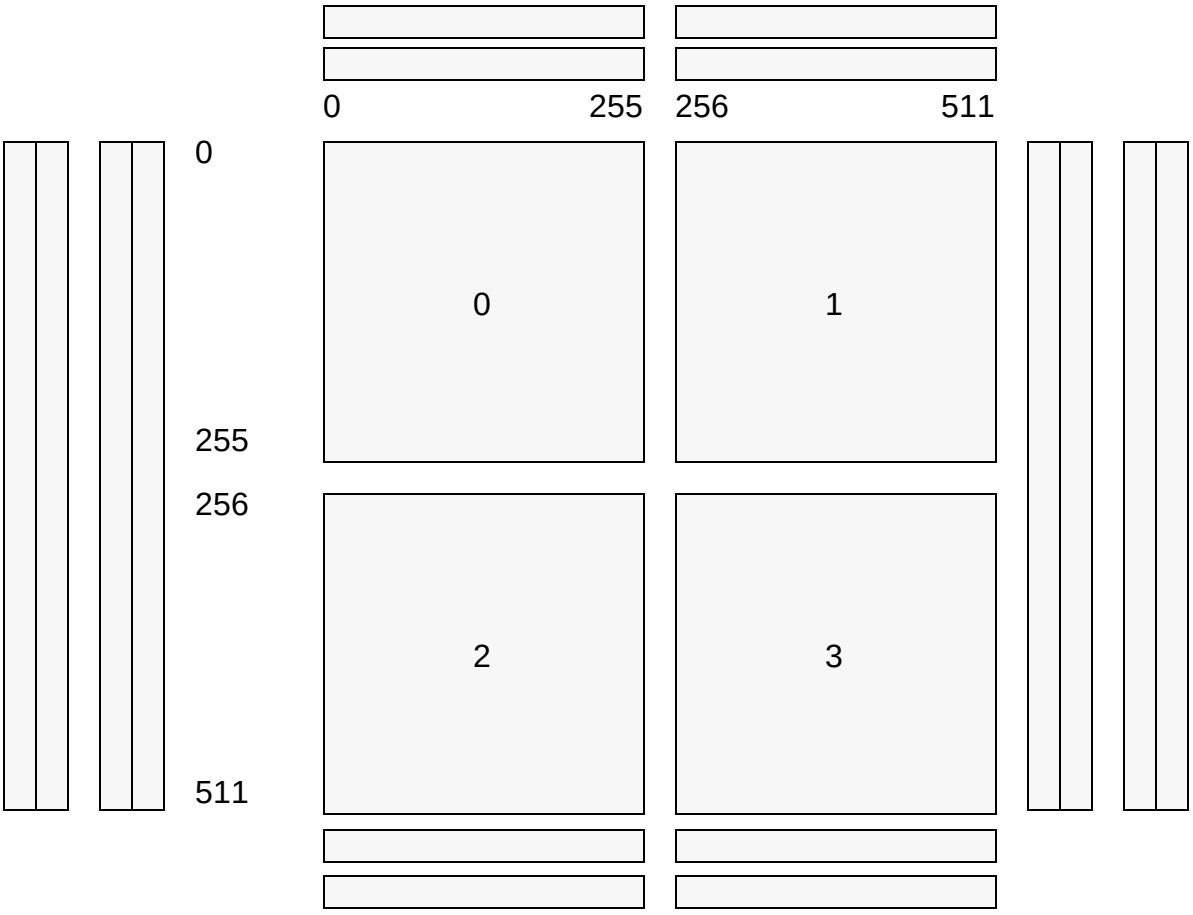
Global Row Last = 511

Global Column First = 0

Global Column Last = 511

Flag checked: *Discretionary*

Example 4—Coupled Redundant Elements



This segmented single-data-bit device has linked and coupled redundant elements. Each of the four segments has two independent redundant rows. Each two segments shares four redundant columns, in groups of two.

Redundancy Sheet Settings

of Segments = 4

Column Spares Number = 2

Used Column Spares = 2

Row Spares Number = 2

Used Row Spares = 1

Row Linkage = segment *x* linking to segment *x*

Column Link =Segment 0 links to segment 2, Segment 2 links to segment 0, Segment 1 links to segment 3, Segment 3 links to segment 1

Segments 0 and 1: First Segment Row = 0

Segments 0 and 1: Last Segment Row = 255

Segments 2 and 3: First Segment Row = 256

Segments 2 and 3: Last Segment Row = 511

BitMask = 0x1

Global Row First = 0

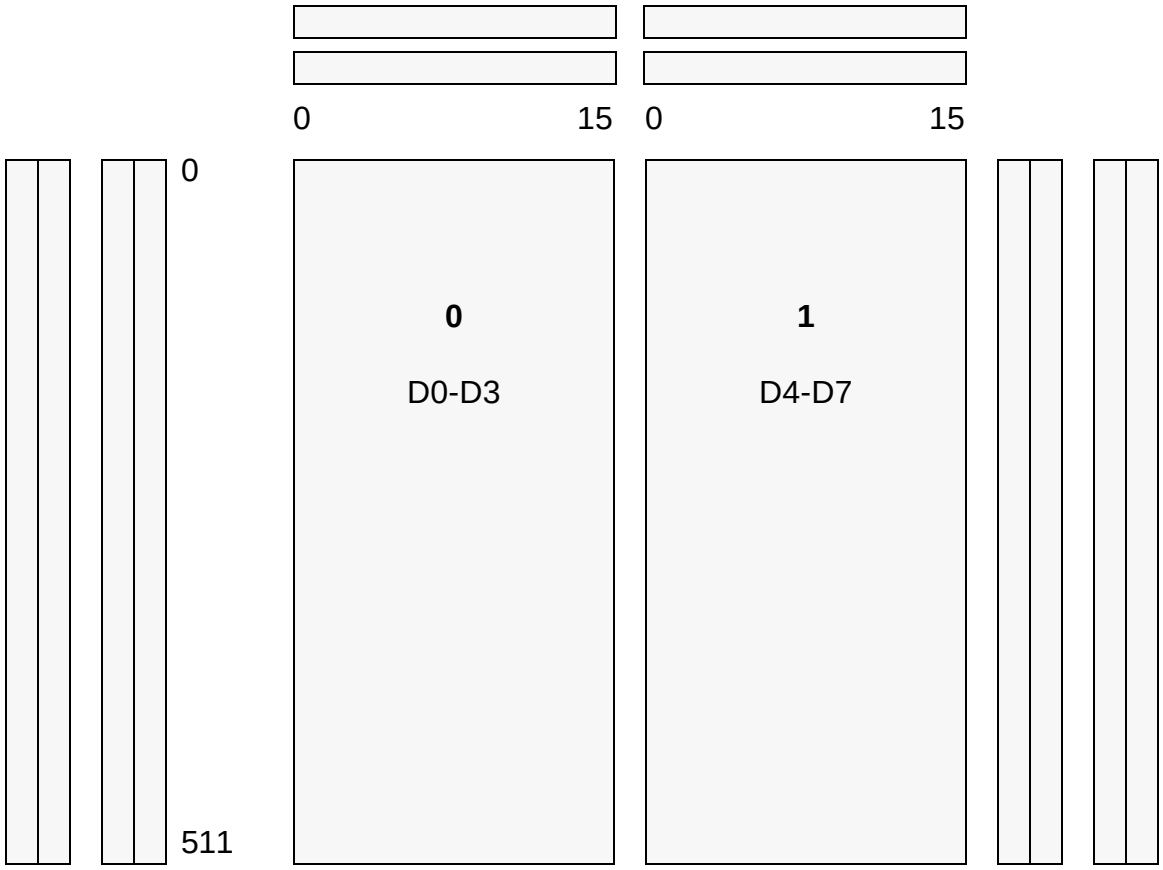
Global Row Last = 511

Global Column First = 0

Global Column Last = 511

Flags checked: *Discretionary* and *Physical Rows and Columns*

Example 5—Segmented Bit Mask



This segmented device has linked and coupled redundant elements, as well as a segmented bit mask. It has eight data outputs.

Each of the four segments has two independent redundant rows. Each pair of segments shares two groups of two redundant columns.

Redundancy Sheet Settings

of Segments = 2

Column Spares Number = 2

Used Column Spares = 2

Row Spares Number = 2

Used Row Spares = 1

Row Linkage = segment *x*

Column Linkage = segment *x*

First Segment Row = 0

Last Segment Row = 255

First Segment Column = 0

Last Segment Column = 255

Segment 0: BitMask = 0xf

Segment 1: BitMask = 0xf0

Global Row First = 0

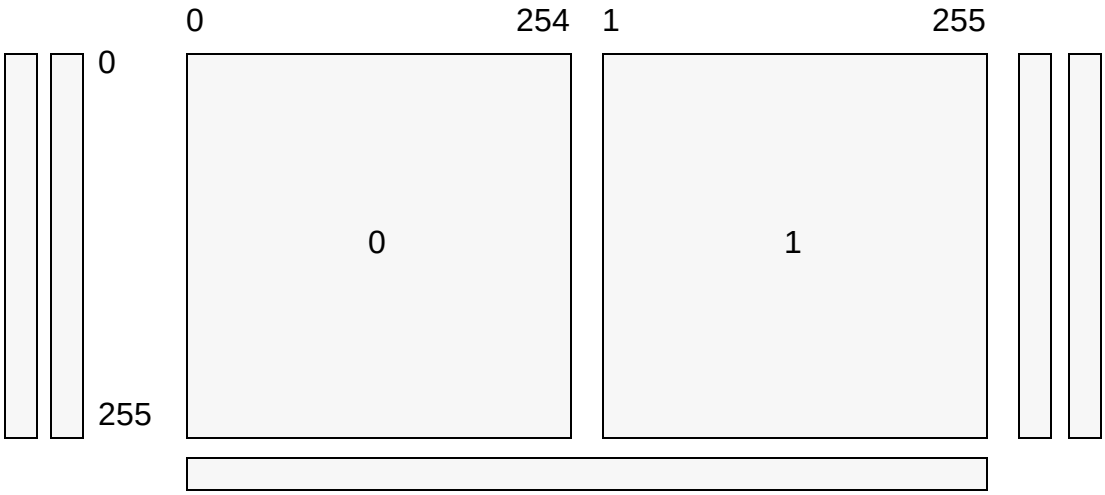
Global Row Last = 511

Global Column First = 0

Global Column Last = 15

Flag checked: *Discretionary*

Example 6—Scrambled Addressing



This example shows a segmented device with linked redundant rows, as well as interleaved addresses.

This device has two segments, each has two independent redundant columns. Segment 0 contains only the even columns; segment 1 contains only the odd columns. The two segments share a single independent redundant row.

Redundancy Sheet Settings

of Segments = 2

Column Spares Number = 2

Used Column Spares = 1

Row Spares Number = 1

Used Row Spares = 1

Row Linkage = (segment x + 1) %2

Column Linkage = segment x

First Segment Row = 0

Last Segment Row = 255

First Segment Column = segment x * 128

Last Segment Column = (segment x + 1)*128 - 1

BitMask = 0x1

Global Row First = 0

Global Row Last = 255

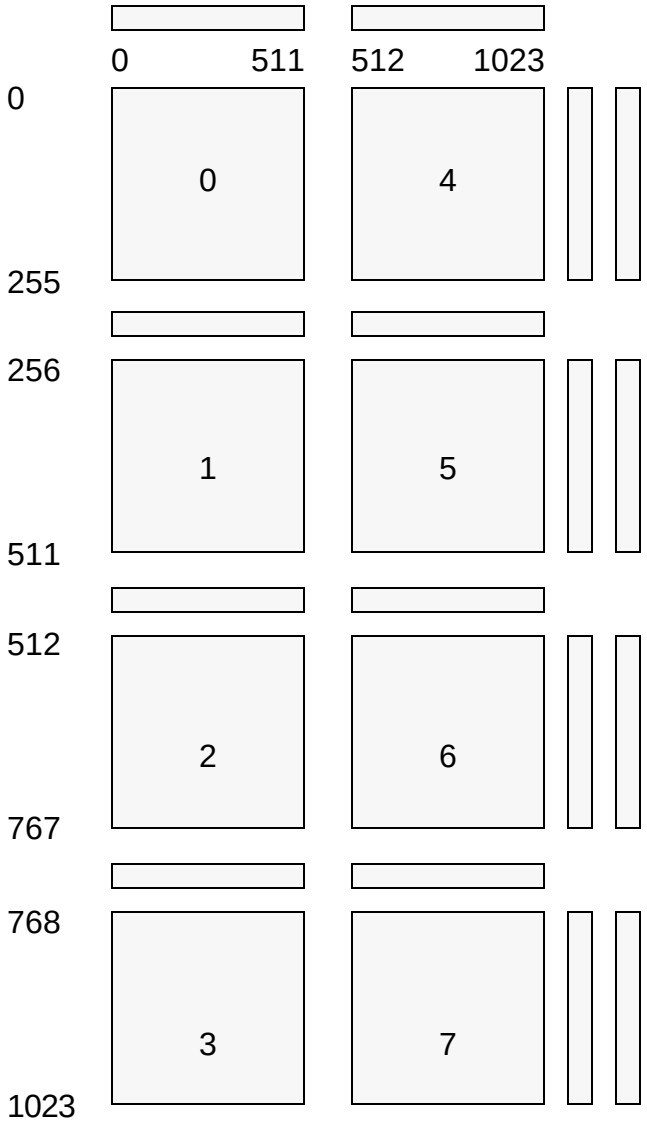
Global Column First = 0

Global Column Last = 255

Flag checked: Discretionary

FLASH750 provides 4 scramble RAMs that are used for either error capturing or bitmapping; use the **tl_EcrScramSelectSet()** routine to select either error capture or bitmapping.

Example 7—DRAM



This example shows a single-I/O bit DRAM.

This DRAM has eight redundancy segments, each has one independent redundant row. Every two segments share two independent redundant columns.

Redundancy Sheet Settings

of Segments = 8

Column Spares Number = 2

Used Column Spares = 1

Row Spares Number = 1

Used Row Spares = 1

Row Linkage = i

Column Linkage = (segment x + 4) %8

First Segment Row = (segment x %4) * 256

Last Segment Row = (segment x + 1) * 256 - 1

Segments 0, 1, 2, and 3: First Segment Column = 0

Segments 0, 1, 2, and 3: Last Segment Column = 511

Segments 5, 6, 7, and 8: First Segment Column = 512

Segments 5, 6, 7, and 8: Last Segment Column = 1023

BitMask = 0x1

Global Row First = 0

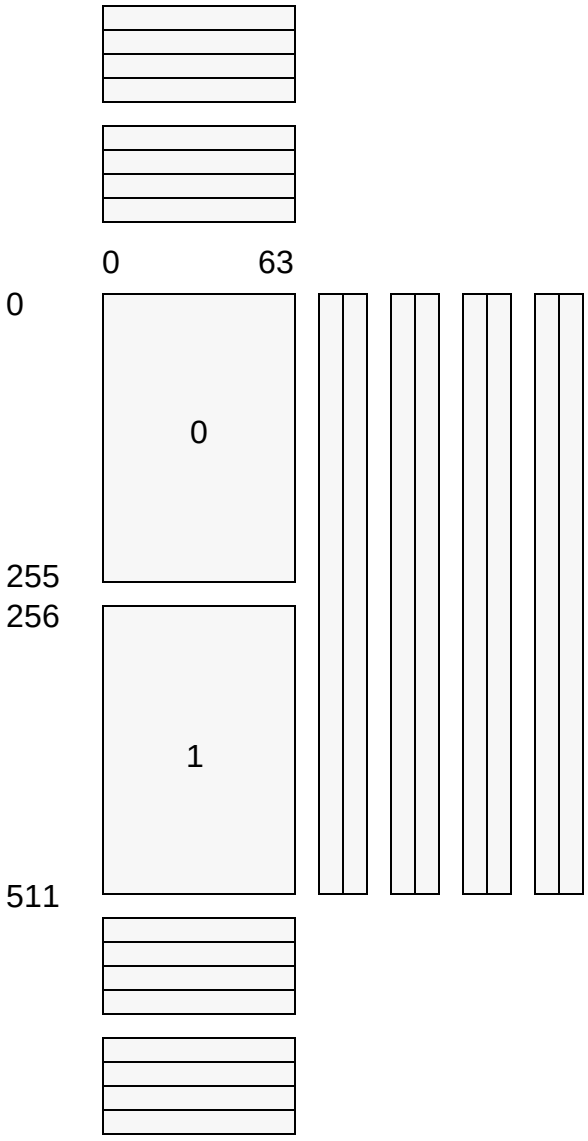
Global Row Last = 1023

Global Column First = 0

Global Column Last = 1023

Flag checked: *Discretionary*

Example 8—EEPROM



This example shows a 256K EEPROM with two segments. Each segment has two four-row redundant row elements. The two segments share four two-column redundant column elements.

Redundancy Sheet Settings

of Segments = 2

Column Spares Number = 4

Used Column Spares = 2

Row Spares Number = 4

Used Row Spares = 4

Row Linkage = segment x

Column Linkage = (segment x + 1) %2

Segment 0: First Segment Row = 0

Segment 0: Last Segment Row = 255

Segment 1: First Segment Row = 256

Segment 1: Last Segment Row = 511

First Segment Column = 0

Last Segment Column = 63

BitMask = 0x1

Global Row First = 0

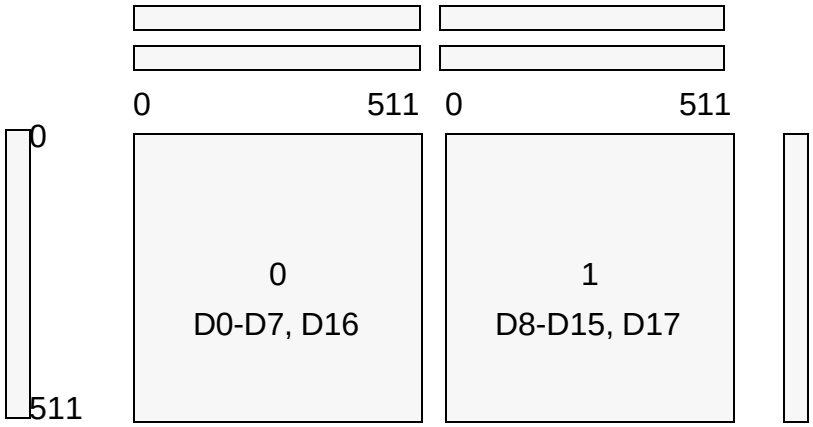
Global Row Last = 511

Global Column First = 0

Global Column Last = 63

Flag checked: *Discretionary*

Example 9—x18 I/O DEVICE



This x18 I/O device has two redundancy segments. Each has two independent, redundant rows and one independent column.

Segment 0 contains errors for D0 through D7 and D16; segment 1 for errors for D8 through D15 and D17.

Redundancy Sheet Settings

of Segments = 2

Column Spares Number = 1

Used Column Spares = 1

Row Spares Number = 2

Used Row Spares = 1

Row Linkage = segment x

Column Linkage = segment x

First Segment Row = 0

Last Segment Row = 511

First Segment Column = 0

Last Segment Column = 511

Segment 0: BitMask = 0xff

Segment 1: BitMask = 0xff00

Global Row First = 0

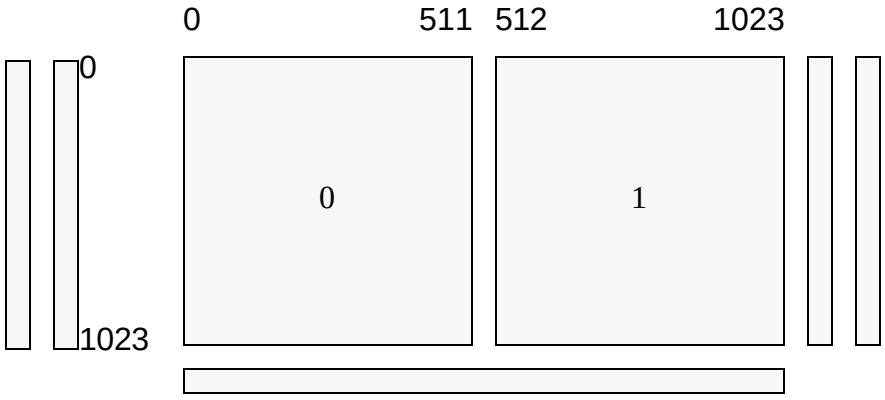
Global Row Last = 511

Global Column First = 0

Global Column Last = 511

Flag checked: none

Example 10—Special Linkage



This device with special linkage has two redundancy segments. Each has two independent, redundant columns and one dependent row.

If one of the two segments contains a bad row, only half of the redundant row replaces the bad row. If both segments contain a bad row, the entire redundant row replaces the bad row. A device with more than one bad row should be unrepairable.

Redundancy Sheet Settings

of Segments = 2

Column Spares Number = 2

Used Column Spares = 1

Row Spares Number = 1

Used Column Spares = 1

Row Linkage = (segment x + 1) % 2

Column Linkage = segment x

First Segment Row = 0

Last Segment Row = 1023

First Segment Column = segment x * 512

Last Segment Column = (segment x * 512) + 511

BitMask = 0x1

Global Row First = 0

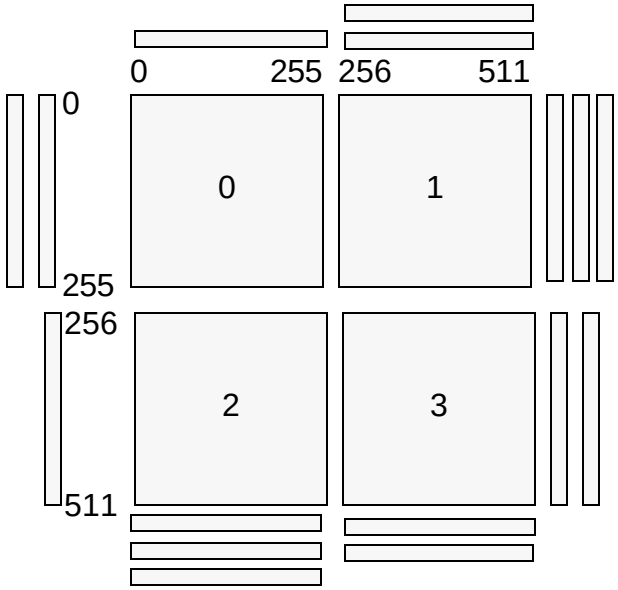
Global Row Last = 1023

Global Column First = 0

Global Column Last = 1023

Flag checked: *Row Results w/o Linkage*

Example 11—Unequal Row and Column Resources



This single data-bit device has unequal row and column resources. It has four redundancy segments:

Redundant Segment	Row	Columns
0	1	2
1	2	3
2	3	1
3	2	2

Redundancy Sheet Settings

of Segments = 4

Column Spares Number = 3

Used Column Spares = 1

Row Spares Number = 3

Used Row Spares = 1

Segment 0: Differential Rows = 1

Segment 0: Differential Columns = 2

Segment 1: Differential Rows = 2

Segment 1: Differential Columns = 3

Segment 2: Differential Rows = 3

Segment 2: Differential Columns = 1

Segment 3: Differential Rows = 2

Segment 3: Differential Columns = 2

Row Linkage = segment x

Column Linkage = segment x

First Segment Row = (i%2) * 256

Last Segment Row = (i%2) * 256 + 255

First Segment Column = (i/2 * 256

Last Segment Column = (i/2 * 256 + 255

BitMask = 0x1

Global Row First = 0

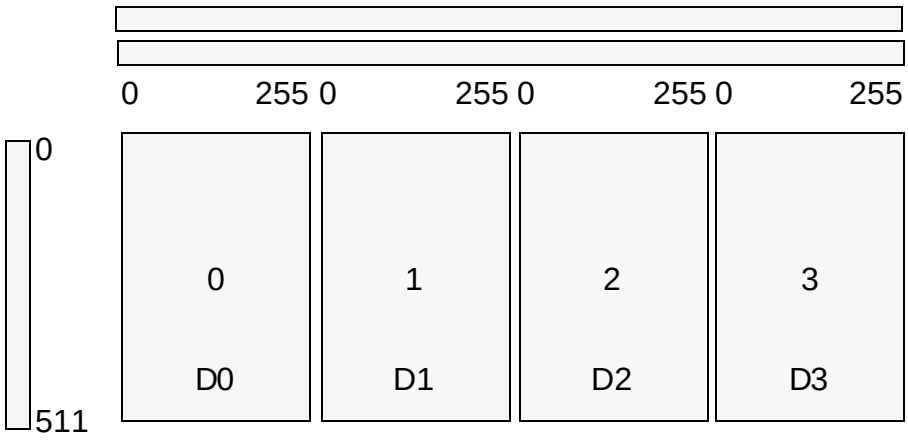
Global Row Last = 511

Global Column First = 0

Global Column Last = 511

Flag checked: *Unequal R (row)/C (column) resources per segment*

Example 12—Special Data Linkage



This device has special column linkages: 2 redundant rows are shared across the 4 segments and 1 redundant column that replaces only 1 I/O out of 4 I/Os.

Redundancy Sheet Settings

of Segments = 4

Column Spares Number = 1

Used Column Spares = 1

Row Spares Number = 2

Used Column Spares = 1

Row Linkage = (segment *x* + 1) % 2

Column Linkage = (segment *x* + 1) % 4

First Segment Row = 0

Last Segment Row = 511

First Segment Column = 0

Last Segment Column = 255

BitMask = 1 << i

Global Row First = 0

Global Row Last = 511

Global Column First = 0

Global Column Last = 255

Flag checked: *Spare Column per data*

Redundancy Analysis Routine: tl_EcrRdRepair

Invoking Redundancy Analysis

TL_RESULT tl_EcrRdRepair(bool init, TVAL timeout);

Description

This routine performs redundancy analysis and saves the per-site results to the

Performs the redundancy analysis (RA) and saves the per-site results to the rd_status[] array where each entry may contain one of these values:

TL_RD_UNKNOWN	Status is unknown.
TL_RD_REP	Errors, but repairable.
TL_RD_NOREP	No errors.
TL_RD_UNREP	Errors, but not repairable.
TL_RD_OVF	Results array overflow.
TL_RD_TIMEOUT	Timed out before RA was completed.

Row repair recommendations are stored in the srb[] array, and the column repair recommendations are stored in the scb[] array.

Parameters

init — **False** specifies RA is not initialized; **True** specifies RA is initialized and is ready for a new part.

timeout — Maximum time limit for redundancy analysis.

The *init* argument is a Boolean variable: 0 specifies redundancy analysis without initialization; 1 initializes the redundancy algorithm before the analysis. Initializing the redundancy algorithm prepares it for a new part; every segment has its full complement of spare row and column elements defined in the ecr_redtable structure. Set the *init* variable to 1 when testing devices that are repaired off-line. Set the *init* variable to 0 only when the algorithm should not be initialized between calls to MTbemRdRepair, such as for devices repaired on-line.

The *timeout* argument sets a maximum time limit for redundancy analysis, allowing you to adjust a wafer sort or other production program for maximum throughput. Typical settings are from 5 to 20 seconds. A timeout occurs, a result of RD_TIMEOUT is returned for the socket in question; the spare-row and spare-column arrays (srb and scb) will both be zero; refer to *Actions and Results Returned by tl_EcrRdRepair()* on page 11-42.

Return Value

TL_SUCCESS is returned for successful operation; **TL_ERROR** for error.

Actions and Results Returned by tl_EcrRdRepair()

tl_EcrRdRepair() analyzes every location that is greater than 0 in the sktl argument. Per-socket results are returned to the rd_status Array, where each location corresponds to a location in the sktl array. Each location has one of six values:

value	socket represented by this location
RD_UNKNOWN	status unknown; all locations initialized to RD_UNKNOWN.
RD_REP	errors, but repairable.
RD_NOREP	no errors.
RD_UNREP	errors, but unrepairable.
RD_OVF	results array overflowed.
RD_TIMEOUT	timed out before the analysis was completed.

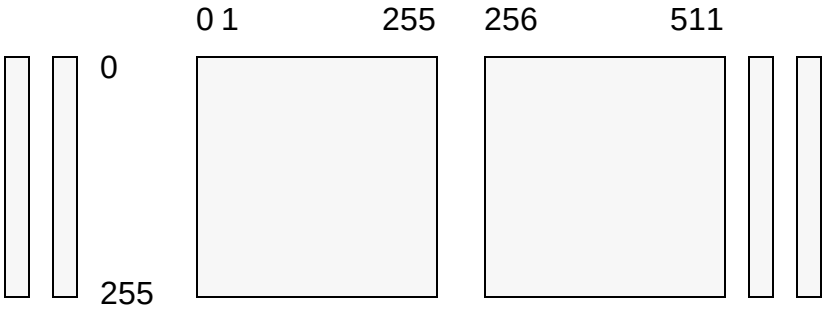
In addition, replaced rows and columns are returned to the segment row and column arrays (srb and scb). Entries are grouped by segments, starting with 0. For non-segmented devices, 0 is the only segment.

The first word of a segment entry is a header word, containing the segment number in the upper byte and the number of lines replaced in the lower byte.

The next one or more words contain the row or column numbers replaced, one per word. The lower byte of the segment header word is the number of words of replacement rows or columns after the header word for that segment.

Immediately following the last line replaced for any segment is the segment header word for the next segment or zero if no more segments or replacements.

Consider the following sample device, which has two redundancy segments, each of which has two redundant columns:



Assume each segment has 2 bad columns: 10 and 187 in the first segment, 265 and 495 in the second segment. After a call to tl_EcrRdRepair, the scb Array contains:

word	hex value	meaning
0	0000	0: identifies segment zero
1	0002	two columns replaced
2	000A	Word specifies first column replaced: hex A (decimal 10).
3	00BB	Word specifies next column replaced: hex BB (decimal 187).
4	0001	1: identifies segment one
5	0002	two columns replaced
6	0009	Word specifies first column replaced in this segment: hex 109 (decimal 265).
7	00EF	Word specifies next column replaced in this segment: hex 1EF (decimal 495).
8	FFFF	End of data.

The maximum number of redundancy segments in a device is 256. Each segment has 16 row and 16 column repairs; thus, the srb and scb Arrays are 4352 words each.

Some devices allow only block-replacement of 2, 4, 8, or 16 rows or columns at a time. You specify the number of rows and columns in a block with the ecr_redtable.rt_rut (rows used together) and ecr_redtable.rt_cut (columns used together) variables in the ecr_redtable. **tl_EcrRdRepair()** routine analyzes errors for these blocks of 1, 2, 4, 8, or 16 physical elements and returns block numbers—not physical line numbers—to the srb and scb Arrays. If ecr_redtable.rt_rut or ecr_redtable.rt_cut is not equal to 1, these block numbers differ from the physical element numbers replaced.

For example, if ecr_redtable.rt_rut = 8, and block 0 was returned to the srb Array, physical rows 0 though 7 would need replacing. If block 9 was returned, physical rows 72 through 79 (decimal) would need replacing.

Actions and Results Returned by RT_CONT

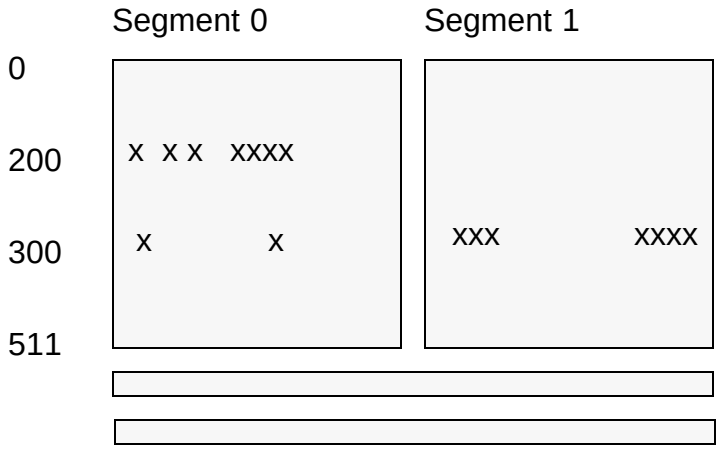
User-preference flag *Continue Analysis to End* continues redundancy analysis in spite of an unrepairable segment until all segments are analyzed. After a call to tl_EcrRdRepair, the RD_UNREP is returned to the rd_status Array as a per-socket result if the device with unrepairable segments. The rows and columns replaced are returned in the srb and scb Arrays for repairable segments and only the header word, which is the segment number in the upper byte and the unrepairable flag (0xFF) in the lower byte, is returned to the srb Array and the scb Array for unrepairable segment.

For example, a device with 4 redundancy segments has 2 bad columns in the first segment: columns 15 and 25; the second segment is unrepairable; the third segment does not require repair; and the fourth segment has 3 bad columns: columns 2, 3, and 4. After a call to tl_EcrRdRepair, the scb Array contains:

word	hex value	meaning
0	0000	0: identifies segment zero
1	0002	2: two columns replaced
2	000F	First column replaced: hex F (decimal 15)
3	0019	Next column replaced: hex 19 (decimal 25)
4	0001	1: identifies segment first
5	0000	2: two columns replaced
6	00FF	FF: unrepairable flag
7	0003	3: identifies segment three
8	0000	2: two columns replaced
9	0002	First column replaced: hex 2 (decimal 2)
10	0003	Second column replaced: hex 3 (decimal 3)
11	0004	Third column replaced: hex 4 (decimal 4)
12	0000	End of data

Actions and Results Returned by RT_RLNK or RT_CLNK

User-preference variable RT_RLNK or RT_CLNK returns the rows or columns replaced to the srb and scb Arrays; however, the rows or columns are linked. For example, the device shown below has two redundancy segments and two redundant row elements shared between the segments. Assume the first segment has 2 bad rows: rows 200 and 300; the second segment has 1 bad row: row 300, and the RT_RLNK bit is set.



After a call to **tl_EcrRdRepair()**, the srb Array contains:

word	hex value	meaning
0	0000	0: identifies segment zero
1	0002	2: two rows replaced
2	00C8	First row replaced: hex C8 (decimal 200)
3	012C	Next row replaced: hex 12C (decimal 300)
4	0001	1: identifies segment one
5	0001	1: one row replaced
6	012C	Row replaced: hex 12C (decimal 200)
7	FFFF	End of data

Without RT_RLNK, the srb Array would not contain the expected result for the second segment because row elements are linked between the segments, and the second bad row in the first segment is the same as the only row in the second segment. Selecting RT_RLNK lets you determine which segment contains the bad row. In the example, row 200 is good in the second segment, and row 300 is bad in both segments.

Related Routines

- **tl_EcrErrCount()**.
- **tl_EcrErrScan()**.

Replacing Elements at the End of Testing

If repairs are at the end of a multiple-pattern test, the redundancy routine should have only one call to tl_EcrRdRepair(). At the end of the series of patterns, the BEM contains the accumulation of all locations that failed during any pattern. After the pattern, the call to **tl_EcrRdRepair()** analyzes the errors and generates either the list of rows and columns to replace to repair the device or segment or indicates by the array rd_status Array that the device is unrepairable. All locations that must be replaced are logged together in the segment row buffer (srb) and segment column buffer (scb) Arrays.

Sample calling sequence invokes redundancy algorithm only at the end the test:

```
.                    /* some functional test */
.
tl_EcrRdRepair (1, SEC(10.0));
```