A Set-Associative, Fault-Tolerant Cache Design [1]

Dan Lamet and James F. Frenzel
Department of Electrical Engineering
University of Idaho, Moscow, ID 83843
208-885-7888 jfrenzel@groucho.mrc.uidaho.edu

Abstract - **The design of a defect-tolerant control circuit for a set-associative cache memory is presented. The circuit maintains the stack ordering necessary for implementing the LRU replacement algorithm. A discussion of programming techniques for bypassing defective blocks is included.**

1 Introduction

The dramatic increase in cache memory size and diminishing geometries has resulted in lower yields. Today's high performance processors often have on-chip cache and consequently the yield of these memories can be a significant factor in determining the ultimate cost of the processor. One way of increasing yields is to provide defect-tolerance through the use of redundant resources. The two methods for achieving defect-tolerance most commonly employed, error correcting codes and spare rows and columns [4], require additional hardware and introduce additional access delays. This paper describes a cache design which uses the redundancy inherent in set-associative caches to operate correctly in the presence of manufacturing defects with minimal additional hardware or propagation delays.

Caches are small memories which operate between the processor and main memory. Their goal is to reduce average memory access time. This is accomplished by using very fast memory, storing only the most used bytes (instructions and/or data) in the cache, and copying multiple word units of main memory called blocks. Block usually contain 4 to 16 bytes from main storage and are organized into groups called sets. In a direct-mapped cache, each set consists of only one block, whereas in an n-way, set-associative (SA) cache, each set contains n blocks, where n is the associativity. The total cache size is the product of the block size, the number of sets, and the associativity [2].

Sohi observed that a cache memory does not have to be defect free to meet its objective, namely reduce the average memory access time of a hierarchical memory system [6]. A direct-mapped cache memory with a defective block will never be able to hold items from main memory which map to that set in the cache. For a cache to operate properly under this condition two things are necessary: one, the cache must be able to recognize a defective block and generate a miss and two, must have the capability of performing a load-through, so that the processor can access the item. An associative cache has alternate locations within a set which can be used when there is a defective block present. Ideally, the circuitry which implements the replacement algorithm would be modified at test time to exclude defective blocks from selection during replacement. Provided each set has at least one good block, all items from main memory can map to a good location in the cache. This exclusion would be implemented in the replacement algorithm circuitry.

[1]This research was supported by NASA under Space Engineering Research Center Grant NAGW-1406

	A	B	C	D
A	X	X	X	X
B	X	X	X	X
C	X	X	X	X
D	X	X	X	X

a) The original matrix.

	A	B	C	D
A		X	X	X
B	X		X	X
C	X	X		X
D	X	X	X	

b)Without the diagonal.

	A	B	C	D
A		X	X	X
B			X	X
C				X
D				

c) The non-redundant information.

Figure 1: The Reference Matrix

When the processor requests a piece of data not currently in the cache, then a miss is said to occur. The cache must decide which block's data to remove, making space for the new data. For a direct-mapped cache, the decision is trivial, as each block from main storage maps to a single block in the cache. However, with a set-associative cache, assuming the referenced set is full, there are n possible blocks to replace. One of the best replacement algorithms is referred to as least recently used (LRU), where the set is treated as a stack and accessing a particular block moves that block to the top of the stack. The least recently used block is always at the bottom of the stack and a miss will load the data into this block and move it to the top of the stack. A block is also moved to the top of the stack on a cache hit because it now the "most recently used." Efficient implementations of the LRU replacement algorithm require $n(n-1)/2$ bits of storage per set to maintain the $n!$ possible stack configurations. Consequently, a 4-way SA cache requires 6 bits of storage per set, while an 8-way SA cache requires 28 bits per set [3]. Alternative replacement strategies are first in, first out (FIFO), and random [2].

2 LRU Replacement Circuit Operation

The set-associative, fault-tolerant cache uses the algorithm described by Maruyama to implement LRU replacement [3]. This method revolves around the concept of a reference matrix, which stores the stack order. The matrix is square, with a column and row for each block. Each bit represents whether or not one block has been used more recently than another block. A 1 indicates that the row block has been used more recently than the column block. For example, if there is a 1 in the B row and C column, then C block has been used *less* recently than B block.

The entire square matrix is not needed to store the stack order. First, the diagonal can be removed, because B column and B row has no meaning. B block can only occupy one space in the stack. Also, one of the remaining triangular sections may be removed since stack order is symmetrical, i.e., if A row, C column indicates the relationship between A and C blocks, then we don't need C row, A column to do the same thing. This leaves only the upper right hand half of the matrix, a triangle with $n(n-1)/2$ elements, where n is the associativity of the cache. Figure 1 shows the reduction of the matrix. Note that in part c), the first block has no column and the last block has no row.

Figure 2 shows the circuit implementation of the defect-tolerant LRU replacement algorithm. The circuit has n inputs and n outputs, one of each for every block. Of the n outputs,

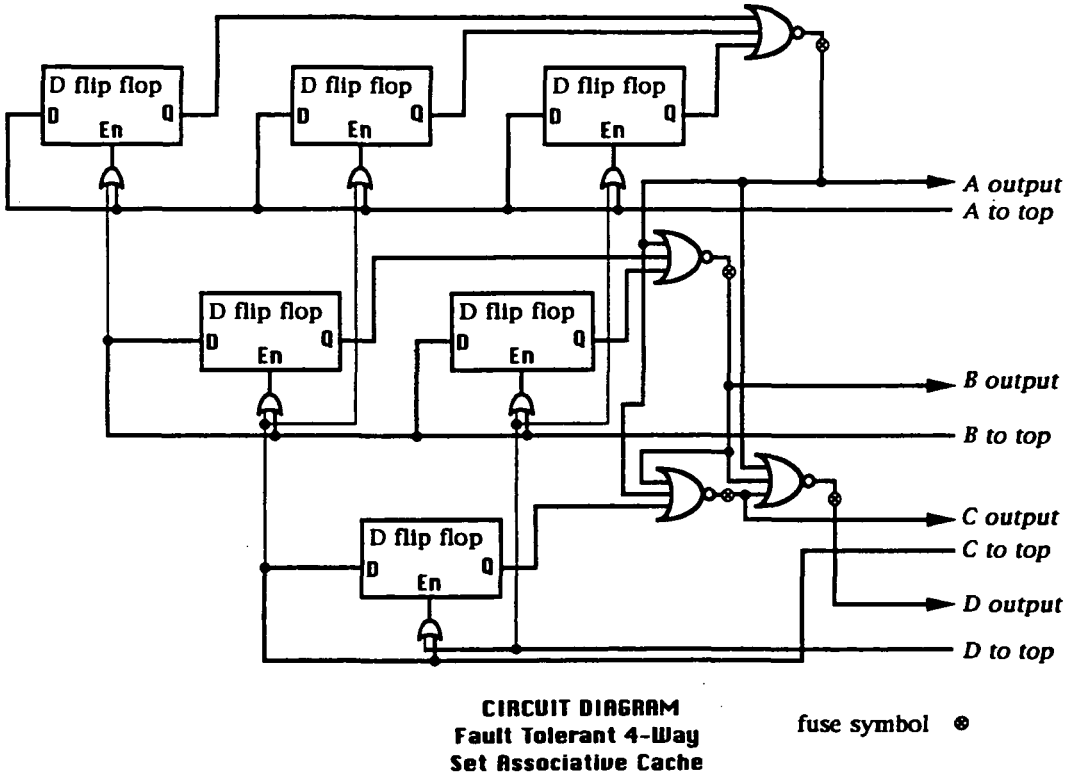

Figure 2: LRU Control Circuitry

one and only one will be high at any one time. This output indicates the block which has been used least recently. The order of the other three blocks is stored correctly within the circuit, but cannot be discerned from the outputs. When a block is faulty, a fuse is blown at its output, permanently fixing that output low. In the case where all blocks have been marked, all outputs will remain 0. This is the only case where at least one output is not high.

The inputs are used to notify the control circuit when a certain block has been referenced. In other words, when the CPU makes a memory call and the cache receives it, either the address being called is already in the cache or it needs to be loaded into the cache. In both cases, that block must now be placed at the top of the stack, indicating that it is the most recently used. This is done by bringing the input line called "'block' to top" high. For example, suppose that after normal operation the stack comes to rest in the order BCDA, where A is at the bottom of the stack (least recently used) and B is at the top of the stack. The CPU sends a request for an address which happens to be located in block D. The cache circuitry routes the appropriate data to the CPU, but at the same time sets the "D to top" line to a 1, causing the stack to place D at the top. The final state of the stack is DBCA. Note that although the stack order has changed, A is still at the bottom of the stack. The "D to top" line must be set regardless of D's position in the stack as it is now the most recently used block. This causes D to go to the top of the stack.

There are several patterns which describe the internal operation of the LRU control circuit. Each row has a certain permanent priority among the blocks. The A block has the

	A	B	C	D
A		1	1	0
B			0	1
C				0
D				

a) The initial state.

	A	B	C	D
A		1	0	0
B			0	1
C				1
D				

b)After C block accessed.

	A	B	C	D
A		1	0	0
B			0	0
C				0
D				

c)After D block accessed.

Figure 3: Normal Circuit Operation

lowest priority and priority increases to the highest, which is D for a 4-way cache. This priority is invoked only when there are two or more blocks with zero rows. Normally, if all the bits in a row are 0, then that block is on the bottom of the stack. If there is more than one row with all zeros, then priority breaks the tie. An example of this is the initial state. At startup all bits are set to zero and because the A block has the lowest priority, it is selected as being on the bottom of the stack.

When a block is selected, that is a memory reference to it is made, it must be moved to the top of the stack. This is accomplished by setting all the bits in the block's row to 1 and all the bits in the block's column to 0. For example, take the stack state as shown in Figure 3, part a). This is for the stack order ABDC, where C is at the bottom. When C is referenced, C row (only a single bit) is set to 1 and C column is set to 0. This case is shown in part C. D then comes to the bottom. Once D is referenced, B comes to the bottom.

To use the fault-tolerant properties of the circuit, fuses must be installed at the output of each NAND gate. The fuses have the characteristic that once they are blown, the incoming signal is left floating and the outgoing signal is drawn to ground. This forces the faulty block to always indicate that it is above the bottom of the stack even though it will eventually float to the top and its row bits will saturate to zeros. Consequently, the faulty block will never be selected for replacement during a cache miss.

3 Programming Techniques

There are several methods for implementing memory reconfiguration in the presence of defects. These are electrically programmable links, electron-beam programmable fuses, and laser cutting/welding [5]. These techniques are employed to bypass faulty resources and activate spare units in common applications, but special considerations must be made for the circuit described in this paper.

Typically, spare rows are initially connected as well as the primary rows. Whether or not any repair takes place, there must be some fuses blown. This is acceptable when the number of rows and columns is limited. In our design, however, there is a fuse for every block. For an 8K cache with a block size of 4, 2K fuses are implemented. It is not feasible to burn all of them on every chip. For this reason, in the absence of defects the circuitry must operate properly with the fuses in-place, without requiring laser programming. Past redundancy techniques typically use only one type of programming. For example, many RAM chip designs are programmed by opening certain connections. A few close certain connections, but none use both technologies. This is acceptable for their purposes, but not

for the LRU control circuit.

The LRU circuit requires that for one output line, two input lines are originally connected so that one is dominant until the fuse is burnt, in which case the second line becomes dominant. Circuitry to implement this requirement is troublesome. One of the simplest solutions is to put a resistor between the second input and the output so that originally, the difference in signals becomes a voltage drop across the resistor and after the fuse is opened, the second signal can be passed through the resistor. This solution is costly. Resistors, or their VLSI equivalent, must be added to 2K fuse locations in the cache, consuming area and some considerable power will be dissipated through the resistors whether or not the fuse is blown. Other more complicated solutions become even more cumbersome. The ideal solution would be to simply open the first line and close the second. This may be possible.

P.W. Cook discovered that connections could be made through an oxide layer on silicon using the same technology which is used to open connections [1]. He documents this in a 1975 paper using a 6 micron technology.

4 Future Work

As stated above, P.W. Cook's method of combining open-fuse and closed-fuse technology may be the best possibility for the cache design presented. One area of research will be to determine if it is still feasible in today's smaller geometries. Electron-beam methods will also be examined. More testing will continue on the operation of the LRU circuit itself as well as defining its surrounding circuitry.

References

[1] P.W. Cook, S.E. Schuster, and R.J. von Gutfeld. Connections and disconnections on integrated circuits using nanosecond laser pulses. *Applied Physics Letters*, February 1975.

[2] J. F. Frenzel. Performance of defect-tolerant set-associative cache memories. In *3rd NASA Symposium on VLSI Design*, pages 3.2.1–3.2.9. University of Idaho, October 1991.

[3] K. Maruyama. mLRU page replacement algorithm in terms of the reference matrix. *IBM Technical Disclosure Bulletin*, pages 3101–3103, March 1975.

[4] Will R. Moore. A review of fault-tolerant techniques for the enhancement of integrated circuit yield. *Proceedings of the IEEE*, pages 684–698, May 1986.

[5] R. Negrini et al. *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays*, chapter 4. The MIT Press, June 1983.

[6] Gurindar S. Sohi. Cache memory organization to enhance the yield of high-performance VLSI processors. *IEEE Transactions on Computers*, April 1989.