Reusing BIST Logic for Intelligent DRAM

Tzi-cker Chiueh   Tulika Mitra

Computer Science Department
State University of New York at Stony Brook
{*chiueh, mitra*}*@cs.sunysb.edu*

1 Introduction

The performance gap between processor and memory is increasing at an alarming rate. As a result, a wide variety of data-intensive applications are running at the memory speed despite the presence of ultra-high processor performance. The idea of IRAM [5] attempts to exploit the huge internal bandwidth within memory chips by integrating a processor core on a DRAM chip. Less ambitious proposals include incorporating specialized logic into DRAM to perform some simple operation like Read-Modify-Write [1], or integrating reconfigurable logic to implement primitive operations [4]. While these proposals could result in significant performance improvements in certain cases, they all require significant changes to the DRAM interface and possibly DRAM core array. Consequently, it is difficult to convince the DRAM vendors, which are necessarily conservative given the scale of capital investments involved, to implement these proposals inside commodity DRAM chips.

However, as DRAM density increases, most DRAM chips in the future are expected to include some sort of *Built-In-Self-Test (BIST)* circuits to alleviate the testing effort required to detect manufacturing/field defects. Even though the BIST logic consumes precious silicon area (about 1%) of the DRAM chip, it is used only during the power-up testing sequence and essentially sits idle for the rest of the time. In this abstract, we propose a simple extension to the standard BIST circuit to support simple memory block copy and memory block zeroing operations, so that BIST logic can also perform useful function even during normal operation. Moreover, these "intelligent DRAM" features are accessible to software without any changes to the pin interface of existing DRAM chips or the processor's instruction set architecture. Finally we show how these new features can improve the performance of several important operations frequently arising inside the operating system. As implementing BIST is in the best interest of DRAM manufacturers, the resistance against "IRAM using BIST" is expected to be much less compared to previous IRAM proposals, especially when the interface of DRAM chips can remain intact.

2 Proposed Design

Figure 1 shows the block diagram of a memory device with an on-chip test circuit [2]. It consists of parallel write and compare circuits. Parallel write circuit is used to write data into memory cells connected to a word-line. The parallel comparator circuit is used to compare all data from the cells with the expected data and to raise pass/fail flag. This BIST circuit requires modification to the column decoder and page buffers. Typically the BIST logic supports multiple test algorithms using microcode to detect various defects. A generic template of memory testing algorithms is to write specific bit patterns to a given row, read the row back, and compare the read and written bit values. This means that the BIST logic must support the following primitives: row read, row write, two row buffers and exclusive OR between the two row buffers. Given these primitives, one can support the following "intelligent DRAM" functions:

- **Page Copy:** copy one memory page to another memory page
- **Page Clear:** zero a memory page
- **Partial Page Copy:** copy a selective portion of one memory page to a possibly different portion of another memory page
- **Partial Page Zero:** zero a selective portion of a memory page.

Partial Page Copy and **Partial Page Zero** require the ability to shift the contents of row buffers in both direction. This shifting capability is the extension required to standard BIST logic.

To maintain the same DRAM chip interface, we have to use unique time sequences of existing pins to encode the "op-code" of IRAM operations and their operands. Fortunately, this does not impose any serious problems [3]. It is the memory controller's responsibility to generate the right bit patterns to instruct the DRAM chips to perform the intended operations. However, the software on the CPU has complete control over which IRAM operations should be performed on which rows and when. The way that the processor expresses these

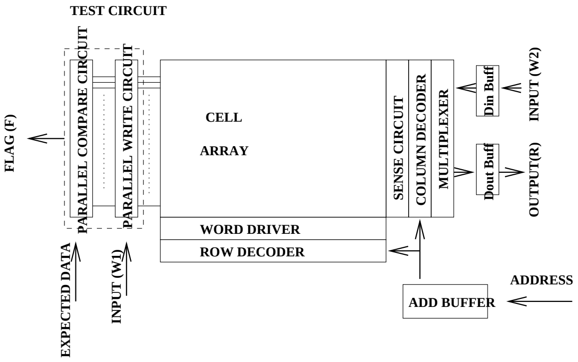

Figure 1: *Block Diagram of a Single-array Multiple-bit BIST Circuit*

intentions is through additional memory-mapped control registers on the memory controller. This is similar to how standard processors instruct disk controllers to perform certain operation by writing to their associated control registers.

3 Applications

There have been a long stream of research effort that aimed to reduce the amount of data copying in networking protocol software as well as in inter-process communication on the same machine. While these research produced impressive performance improvements, they invariably introduced additional software complexities, sometimes to the extent of breaking modularity and hurt code reusability. An obvious example is the *integrated layer processing* approach to protocol stack implementation, which reduces data copying and movement overhead by coalescing software modules that perform logically distinct functions, e.g., compression/decompression and encryption/decryption, into one physical entity. From the software life-cycle standpoint, it is not clear that trading additional software complexity and potentially more expensive maintenance cost for run-time efficiency is a sound engineering decision. Similar problem arises in the case of manipulating virtual memory page table to avoid data copying in inter-process communication. The proposed IRAM architecture is able to perform memory to memory copying at 4 KBytes per 100 nsec, which we believe will completely eliminate the complexities of all software optimizations that minimize data copying.

Zeroing out the memory regions returned by `malloc` is essential to improving application robustness and to facilitating application debugging. With memory zeroing, it is more likely that that applications can detect the use of un-initialized memory regions (e.g., pointers are NULL) earlier; moreover applications are easier to debug because results of application runs are deterministic and would not be affected by random contents of allocated memory regions, which would have been the case if memory regions were not zeroed. Another important use of memory zeroing is the support of secure file system, which, among other things, requires all de-allocated disk blocks to be zeroed. Instead of physically writing zeros to disk blocks when they are de-allocated, one can modify the disk driver in such a way that it records all de-allocated disk blocks, and returns zeroed pages to all read accesses to these de-allocated blocks. This mechanism avoids the expensive operation of zeroing disk blocks while satisfying the requirement that all de-allocated disk blocks should contain 0's when they are accessed. The proposed IRAM architecture can perform memory zeroing also at the speed of 4 KBytes per 100 nsec, which offers significant performance improvement to m̂alloc and disk block deallocation in secure file system.

Scatter/Gather DMA (Direct Memory Access) allows software to specify multiple destination memory locations or multiple source memory locations in a DMA transaction. Although Scatter/Gather DMA performs better than executing each component DMA transaction independently, it still physically performs multiple DMA transactions over the I/O bus such as PCI. With fast memory copying, a Scatter/Gather DMA transaction can be implemented with only *one* I/O bus transaction, because scatter incoming data among multiple memory locations and gathering outgoing data from multiple memory locations can all be converted to memory copy operations. As a result, the entire payload of a Scatter/Gather DMA transaction is always transferred over the I/O bus as a single data burst running at full sped.

References

[1] A. Brown et al. Using MML to Simulate Multiple Dual-Ported SRAMs: Parallel Routing Lookups in an ATM Switch Controller. In *Workshop on Mixing Logic and DRAM*, 1997.

[2] J. Inoue et al. Parallel Testing Technology for VLSI Memories. In *IEEE International Test Conference*, 1987.

[3] Tulika Mitra. Dynamic access memory – a survey. *RPE Report, Computer Science Department, SUNY at Stony Brook, March 1999, http://www.ecsl.cs.sunysb.edu/rpe.html.*

[4] M. Oskin, F.T. Chong, and T. Sherwood. Active Pages: A Computation Model for Intelligent Memory. In *International Symposium on Computer Architecture*, 1998.

[5] D. Patterson et al. A Case for Intelligent RAM. *IEEE MICRO*, 17(2), 1997.