Address and Data Scrambling for Memory Systems

Karen Amirkhanyan
Synopsys
Yerevan, Armenia
e-mail: kamirkha@synopsys.com

Valery Vardanian
Synopsys
Yerevan, Armenia
e-mail: vvardani@synopsys.com

ABSTRACT

In this paper, we defined a new logical organization of a group of memory instances of the same size and structure, called a memory system (MS). A few scrambling types defined earlier for conventional SRAM memory instances are extended for MS that are widely used nowadays in semiconductor industry. It is observed that the defined scrambling types may increase the efficiency of solving several problems of test and repair connected with memory systems.

Keywords

Memory system, scrambling type, address/data scrambling.

1. INTRODUCTION

Nowadays System-on-Chips (SoCs) contain numerous functional blocks and the most prevailing among them are the memories. The memories in SoC can have different sizes from a small size (a few words) up to big sizes (a few Gigabytes) of cells (data storages). The working speed (frequency) of an MS is also different and depends on the functionality of the memory block in an SoC. They can be small but very fast register files, different level caches (L1 to L3) or big but slow storages for the program data, graphical information, etc. In SoCs, the memories of high performance are usually implemented based on the Static Random-Access Memories (SRAM). In current technologies, an SRAM is the highest frequency memory, and that is the reason of widely using SRAMs in different embedded functional blocks of SoCs.

A memory instance is typically implemented as an array of storage cells of regular structure. Each cell stores a discrete bit “0” or “1” of information. Memory control block and column/row decoding logic map an externally provided address into a selected pattern of cells. The selected cells can be either written into or read from, depending on an external command Write or Read.

To test the memory instance, a test engine writes/reads and then compares the predefined data pattern (DP) with the one at a given address location. Efficiency of the memory instance testing highly depends on the correct usage of the structural information (“scrambling”) of the memory instance [1]. The address and the data scrambling of the memory instance are important approaches for increasing the efficiency of the memory testing [2]. The different addressing methods such as: Regular Up/Down, Incremental Up/Down, Pop-Up/Down etc. and different (mainly 16) Data Background (DB) types: Solid “0”/”1”, Double column, Double rows, Checkerboard, Bit-line Checkerboard, etc. (see in [2]) are used as patterns during the testing of the memory instance [1], [2] with a set of test algorithms. We want to mention that the Bit-line Checkerboard BP is constructed based on the Bit-Line and Row scramble information of the memory instance and that is why topological Bit-line Checkerboard provides the highest coverage of the memory testing [1], [2]. By considering this, extraction of the complete and verified scramble information of the memory compiler becomes a very important task [3]-[5] for testing and many other applications, such as memory repair, fault/defect diagnosis, fault analysis, etc.

Another feature of the structure of embedded memories in SoC is that they are usually constructed not based on one instance only but usually on several instances of the memory. This approach provides [6], high performance of the memory, as well as higher flexibility during the physical implementation (placement and routing) in GDS of an SoC.

In this extended abstract, based on the developments of technology and emergence of new structural/hierarchical organizations of sets of high numbers of memory instances in an SoC, we found it relevant to introduce new definitions for the new structural organizations of memory instances, naming them as memory systems.

2. MEMORY SYSTEMS

The Memory System (MS) is a group of (usually, homogeneous) memory instances (i.e., of the same size), such as two or three levels of cache memory of CPUs (widely known as L1 to L3 caches), the register files of different types of controllers, the huge graphical memories, the local caches between a fast bus and a low performance unit, the reconfigurable Jenga CPU cache memory system [3], [6], etc. All the examples mentioned above are a few examples of embedded MS implementation.

We call a logical memory unit/module that is implemented by means of several memory instances as a Memory System. Figure 1 depicts an example of a simple MS that contains n x m memory instances and has a form of a matrix of regular memory instances of the same size. It is convenient to consider it as a matrix or an array of cells of the size m x n where each cell is a memory instance of a given size. Thus, an MS is a direct and logical extension of the notion of a conventional memory instance. An MS has a regular structure, consisting of a group of memory instances of regular and homogeneous structure. In Figure 1, ADR is the logical Address bus of MS, MCS is the set of MS Control Signals, Power is the set of power signals of MS, DATA is the Data Bus of MS, M_{ij}, i=1,…n, j=1,…,m, are the memory instances included in MS. But, however, it should be noted that, in general, the form of an MS may be of other forms as well, not only a matrix. Also, the buses in MS may be not only functional but also test buses (see [6]).

The functionality of an MS could be observed as a memory logical unit, and it could be presented as a black box as well.

At the same time, availability of the physical structure information (scrambling) of MS is very important for some applications such as: 1. the Memory test and repair, 2. Defect analysis and diagnosis, 3. Redundancy planning and sharing

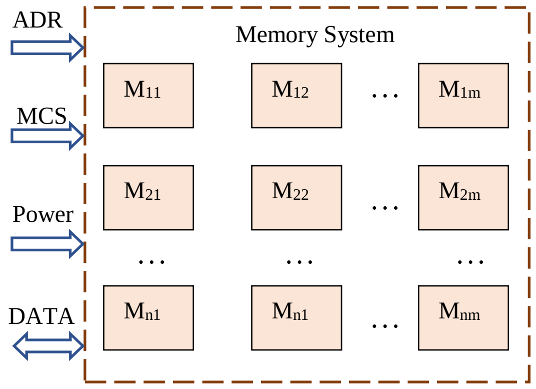

Fig. 1. Definition of the Memory System

in the MS, 5. Register sharing for MS faults, 6. MS Fault analysis, calculation of physical/logical coordinates of a faulty cell in the memory array, etc. As in the case for a memory instance, the physical structure and scramble of MS are very important for the listed above, as well as many other applications (see e.g., [8] - automated extraction of scrambling types, [9] – defect injection flow).

3. SCRAMBLING TYPES (PRIMITIVES) IN SRAM MEMORY COMPILERS

A memory compiler usually contains a GDS (Graphic Data System) file as a geometrical representation of the memory structural model. This model is a detailed description of the memory and naturally has a very large volume of megabytes of information. To develop algorithms for solution of different problems connected with memory design & test, it was previously accepted widely to analyze the GDS model of the memory. Due to the necessity of high computational resources connected with exploration and analysis of the GDS file with detailed and complete information on the memory structure the problems such as development of test algorithms, background pattern extraction for topological checkerboard (or other types), finding physical coordinates or location of defective cells, physical bitmap viewing of defective cells in the memory array, structural changes in the memory after its repair by redundant elements, error correction based on error correction codes, etc., required excessive time for handling of GDS files when considering large memory compilers.

GDS is a binary format for representation of planar geometric shapes, text labels and some other information in a hierarchical form. It reflects the physical layout (Integrated circuit layout), also known as IC layout or IC mask layout, of an integrated circuit in terms of planar geometric shapes that correspond to shapes drawn on photomasks used in semiconductor device fabrication. Originally, GDS was designed as a format of data files used to control integrated circuit photomask. Despite its limited set of features and low data density, it was ubiquitously used for transfer of IC layout data between design tools from different vendors which operated with proprietary data formats. GDS files are considered as final output of the IC design cycle and are passed to IC foundries for IC fabrication.

We proposed (see [3], [4]) a new approach to develop memory models of sufficiently simple structure than the GDS file for consideration of different problems. We have extracted several structural primitives from the layout corresponding to the GDS file constituting a basis for construction of different memory models of different complexity with respect to the information contained in the GDS. By selecting different subsets of structural "primitives" (scrambling types) (see [4]) we constructed different memory models. In dependence on the problem considered, the memory model may be much simple than the GDS and the time complexity for exploration and analysis of the simple structural model of the memory becomes significantly smaller than the one necessary for exploration and analysis of the whole GDS. This approach allows significant reduction of time necessary for extensive simulations.

In [2], the authors described the importance of the usage of address and data scrambling. They indicated the importance of scrambling information for testing issues and defined 16 different Data Backgrounds and suggested to use them with a set of test algorithms applied several times with different DPs to increase the fault coverage of test algorithms. They have done experiments on several March algorithms and experimentally showed that scrambling has significant impact on the fault coverage of the test algorithms by increasing it up to 35%. In [4], we have defined several other scrambling types corresponding to our visual observation of hundreds of GDS files of contemporary memory compilers. Finally, we developed a tool for automated generation of different memory models by selecting the necessary subset of structural primitives from the complete set of structural primitives from the basis. In [3], we defined a procedure called SIV (Scrambling Information Verification) tool for automated verification of the scrambling information generated automatically by means of the Scramble Information Generator (SIG) tool developed in [4]. Shortly, the tool compares the scrambling information with respect to the GDS file of the compiler and thus verifies the correctness of its generation by the SIG tool.

Each memory model has two basic forms for representation: logical and physical. Those transformations can be described in many formats, for example, a TCL procedure. The set of all TCL procedures, corresponding to all transformations from logical to physical and from physical to logical, are described in a database, called the Scrambling Library [4]. When a specific memory model is selected the corresponding scrambling types with their TCL procedures are activated by the tool.

Both forms, the logical and physical, are important and have their range of application. The logical form is used for outer representation of the memory model which can be different from the inner representation of the physical form of the model. In addition to convenience for using the logical form it may have also other purposes as well, e.g., IP protection to veil the main structural features of the memory from customers. The difference between logical structure of the memory seen from outside and internal physical structure is named scrambling. In other words, for example, address scrambling means that logically adjacent cells with logically adjacent addresses may not be physically adjacent. For testing the memory to detect certain faults, the test algorithm which applies read and write operations to the memory cells (words) often needs to allocate specific data on physically neighboring cells (words). Due to the unavailability of the scrambling information, the customer is often forced to apply different data background patterns with the test algorithm to compensate for the effects of scrambling. Often scrambling is introduced intentionally for physical purposes

to improve the electrical characteristics of the design, geometrical optimization, address decoder optimization, for area optimization (sharing cell contacts and well area) (see [3], [4]), speed, robustness improvement by means of bit-line (column) twisting, yield improvement by introducing redundant elements for memory repair, etc. (see [2]-[4]).
The purpose of introducing structural primitives with the corresponding scramble types that provide transformations between logical to physical and physical to logical forms of the memory model is to:

- Provide a mechanism for IP related information transfer between levels of memory embedding hierarchies. Memory Structural Primitives provide interface for using memory structural information without breaking its IP. This allows solving test, diagnosis and repair problems that need memory structural information.
- Provide a variety of memory models with logic and physical forms. The subset of memory structural primitives allows building a memory model that is specific for current application with the same interface.
- Provide possibilities for efficient solution of many problems connected with design, test diagnosis and repair of memories [3], [4]:

1. Design: geometrical optimization, address decoder optimization, area optimization (sharing cell contacts and well area), speed, robustness improvement by means of bit-line (column) twisting;
2. Test: a. Fault coverage increase. Knowledge of scrambling types corresponding to the used structural primitives allows test and detect special classes of realistic functional fault models occurring between neighboring nodes;
   b. Simplification of the memory test algorithm.
3. Data Background patterns: To extract the required data background patterns for the memory test algorithm, certain structural primitives with their corresponding scramble types are necessary.
4. Diagnosis: Knowledge of the physical neighborhood in the memory and usage of the corresponding structural primitives simplifies development of the diagnostic algorithm and increases its efficiency.
5. Fault location: For repair purposes it is important to locate the faulty bits in the memory. The fault detection algorithm locates only the single-cell faults. Two-cell faults, such as coupling faults are sometimes known as realistic and statistically may have high probability of occurrence. To repair such faults, it is necessary to locate the aggressor cells causing the faults.
6. Repair: Reflection of state after memory repair.
7. Coordinate calculation of defective cells: For debugging and silicon validation problems it is necessary to find physical location or coordinates of defective cells in the physical memory.
8. Bitmap viewing. For debugging and silicon validation problems it is necessary to view the distribution of defective cells in the memory array logically and physically.

These structural primitives and the corresponding scramble types do not cover all possible types in memories. Our SIG tool [4] is flexible enough to include new structural primitives and scramble types that can appear in the future memory compilers of new technology and create new memory models by selecting a certain subset of structural primitives.

4. EXAMPLES: ADDRESS AND DATA SCRAMBLE TYPES IN MS

We consider the following list of scrambling types T_{nm} $\{ADR_{nm}, (X,Y)_{nm}, O_{nm}, D_{nm}\}$ for an embedded MS.

The following notations are used:
ADR_{nm} is the logical address of instance T_{nm} in MS:
$(X,Y)_{nm}$ are the ordinary X and Y coordinates of the reference point of an instance T_{nm} in the GDS file of an MS;
O_{nm} is the orientation and mirroring [3], [4] of an instance T_{nm} in the GDS file of an MS;
D_{nm} is the Data range position of I/O bits of an instance T_{nm} in an MS word structure.
All T_{nm} parameters are used in the listed above different applications of the MS test, repair and fault diagnosis.

ADR_{nm} is typically used to describe the address value in the range of logical address bits. For an example MS, the logical address scramble looks like [A10, A9 … A0] but the bits [A10:A8] are addressing instances M_{nm} in an MS. Data range position shows how the Data bus of an MS is constructed by D_{nm} of instances. For example, the Data Bus of MS contains bits [0:31], and it was formed by means of Data Bus of two instances. For example, D_{11} forms the lower [0:15] bits of the word, and D_{12} forms the upper bits [16:31] of the word.

O_{nm} is a string and shows the orientation and mirroring of the instance. A typically used orientation of cells and mirroring in the GDS are: r0, r0mx, mxr0, r90, mxr90, r90mx, r180,

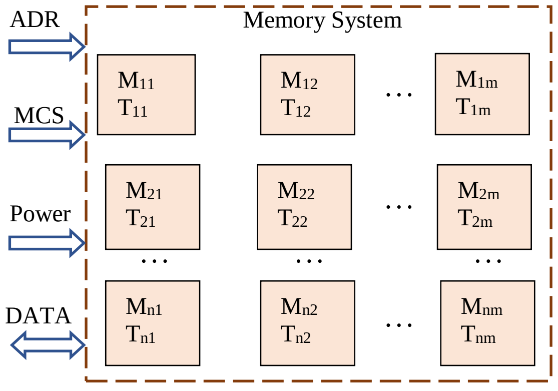

Fig. 2. Scrambling types of embedded MS

mxr180, r180mx, r270, mxr270, r270mx.

4.1 Address scrambling

As we already mentioned, an MS is considered by an MS environment as the logical/functional module that is controlled by the Address, Data busses and the Control signals. In the logical aspect, ADR (see Figure 2) is the set of the address signals with the range [0: ADRmax]. ADR includes two main parts: 1. Instance address and 2 Addresses of certain Cells in the instance (Figure 3). The purpose of Instance address is to select the corresponding memory instances in the MS. By means of this address signal in MS we generate the Memory Enable (ME) signals for the memory instance.

The purpose of the selection address for Instance’s Cells is the addressing and activation of the MS word cells in the memory instance. Those signals include the following addresses of the instance: Columns, Rows and Banks (if there are any) in the memory instance. The instance address signals activate the corresponding rows and columns of the instance memory area cells during the Write and the Read operations of MS. The address scrambling of the instance is an important part of the success in memory testing [3].

MS logic address			
Instance logic address	Cells of the word logic address in the instance		
Instance address	Bank address	Row address	Column address

Fig. 3 MS address structure

In the given example all the parts of the MS address are presented in the regular distribution but in the general case the distribution of the address bits and parts can be irregular or mixed and in that case the address structure will be source of the scrambling for either the logical and the physical address of MS.

4.2 Data scrambling

The next type of the scrambling in MS is the Data scrambling (DS). For example, some cases of DS could be observed for MS with the physical structure with 2 rows and 4 columns (see Figure 4). In the given example DATA range is 64 bits and, the memories in the physically upper row are connected to [0:31]. Data bits and the memories in physically lower row, are respectively connected to [32:63] data bits of DATA bus. That kind of distribution of data bits

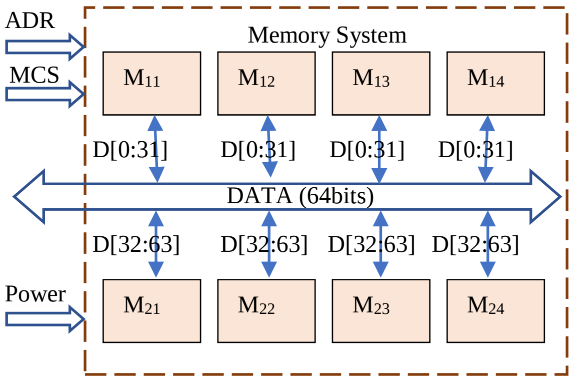

Fig. 4. Data scrambling in the Memory System

assumes that in the MS the Read, and the Write operations of 64 bits are implemented by means of two memory instances (the same ME signal is used), for example by M11 and M24 instances. In the general case, the pairs can be formed by means of any instances from the two rows. Of course, the Data bus can be physically distributed by many other different ways also: each memory instance has 64-bits data bus (64-bit Read/Write operation requires one instance), each memory instance has 16-bit data bus (64-bit Read/Write operation requires four instances) etc. All mentioned distributions of the data bits are the reasons of the logical to the physical scrambling in the MS DATA.

5. MS APPLICATIONS WITH SCRAMBLING TYPES

For the embedded memories, there are some applications which require the knowledge of certain scrambling types for both the MS and the memory instances in MS. Moreover, they can be implemented only based on the scrambling information. Let us list some of them: MS Built-in self-test (BIST) (requires the address and the data scramble information); Diagnosis of the detected defect type; redundancy/register sharing, calculation of the topological X, Y coordinates of a defective cell in a memory instance of MS. We will illustrate the usage of MS scrambling in the defect coordinate calculation. That application usually is used during the Fault Analysis (FA) for a new technology SoC production. FA is required for those complicated cases when the source of the defect should be detected in the wafer. After BIST run and a defect detection in an MS the test algorithm returns the logical address and error bit position in the word of the defective cell. Based on this logical information the physical address (the physical number of the row (denoted N(rows) and the column - N(columns)) of the defective cell must be calculated by means of the address and data scrambling of MS. Then using the scrambling information of physical sizes of the memory bit-cell, sizes of the borders (left, right, bottom and top) of the memory instance and (X;Y) coordinates of the reference point of the instance, then taking into account the instance orientation we can calculate the ordinary X(d) and Y(d) coordinates of the defective cell(s) by the formulas:

X(d)= X(instance) + Size(left/right boarder) +
+N(columns) x X(size of the bit cell);
Y(d)= X(instance) + Size(bottom/top boarder) +
+N(rows) x Y(size of the bit cell);

It is obvious that calculation of coordinates X(d) and Y(d) would be impossible without the knowledge of scrambling information.

6. CONCLUSION

A notion of Memory System (MS) is introduced as an extension of an SRAM instance. The notions of scrambling types known for instances are extended for MS. It is observed that knowledge of certain scrambling types for an MS are important for many problems of test & repair. A few examples of scrambling types for an MS are considered.

REFERENCES

[1] A.J. van de Goor "Testing Semiconductor Memories: Theory & Practice", *ComTex Publishing*, pp. 1-512, 1998.

[2] A.J. van de Goor, "Address and Data Scrambling: Causes and Impact on Memory Tests", *DELTA*, pp. 128-136, 2002.

[3] K. Alexanyan, K. Amirkhanyan, S. Karapetyan, S. Shoukourian, A. Shubat, V. Vardanian, Y. Zorian, "Various methods and apparatuses for memory modeling using a structural primitive verification for memory compilers", *US Patent* No. 8,112,730, pp. 1-22, 2012.

[4] K. Aleksanyan, K. Amirkhanyan, S. Shoukourian, V. Vardanian, Y. Zorian, "Memory Modeling Using an Intermediate Level Structural Description", *US Patent*, No 7768840, USA, pp. 1-17, 2010.

[5] S.-K. Lu, H.-C. Jheng, H.-W. Lin, M. Hashizume_, and S. Kajihar, "Built-in Scrambling Analysis for Yield Enhancement of Embedded Memories", *IEEE 23rd Asian Test Symposium*, pp.137-142, 2014.

[6] M.Tyson, "Reconfigurable 'Jenga' CPU cache memory system proposed https://hexus.net/tech/news/cpu/107725-reconfigurable-jenga-cpu-cache-memory-system-proposed/, pp. 1-3, 2017.

[7] D. Schor, "AMD's Zen CPU Complex, Cache, and SMU", *Architectures, Circuit Design*, ISSCC 2018 pp, 1-2, 2018.

[8] K. Amirkhanyan, K. Darbinyan, A. Davtyan, G. Harutyunyan, S. Shoukourian, V. Vardanian, **Y.** Zorian, "Generation of Memory Structural Model Based on Memory Layou**t",** *US Patent* No. 9,514,258, pp. 1-32, 2016.

[9] *K.* Amirkhanyan "Defect injection and memory test algorithms verification flow" *CSIT*, pp. 283-286, 2011.