Fault Tolerance and Yield Improvement of Embedded Memories

Boris Polianskikh, B. Eng. 1994

Department of Electrical and Computer Engineering

McGill University, Montreal

November 2001

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering

Abstract

Recent advances in microelectronics industry allow us to create a System-On-Chip. The embedded memory is one of the vital parts of any system-on-chip. Today it is not enough just to design and fabricate the embedded memory. In order to put the System-On-Chip in mass production, the designer has to be concerned about yield and reliability of the embedded memory. This thesis provides background on fault tolerance improvement theory, and gives several new solutions on how to improve reliability and enhance yield of the embedded memory in efficient ways.

A complete fast embedded SRAM and Control Block for Programmable Clock Manager have been designed, implemented, integrated into a System-On-Chip and tested. The thesis incorporates two novel circuits that significantly improve embedded memory yield and reliability.

This thesis describes new embedded memory architecture for enhanced yield, performance and power consumption. The architecture is able to tolerate major defects including memory kill defects. The mathematical model of the new architecture is presented as well and shows the advantages of new architecture. The new induced Error-Correcting Code (ECC) for Multilevel Dynamic Random Access Memory (MLDRAM) is introduced. The ECC is able to correct 2-bit error and detect 4-bit error. The new ECC also improves reliability and power consumption of the embedded MLDRAM.

Résumé

Les avancées récentes dans le domaine de la microélectronique permettent maintenant de créer un système-sur-puce. La mémoire intégrée est une partie essentielle de n'importe quel système-sur-puce. Aujourd'hui, il ne suffit plus simplement de concevoir et de fabriquer cette mémoire. Afin de produire en série un système-sur-puce le concepteur doit tenir compte du rendement et de la fiabilité de cette mémoire. Cette thèse présente les bases de la théorie d'amélioration de la tolérance des fautes. De plus, elle décrit de façon efficace de nouvelles solutions pour améliorer la fiabilité et mettre en valeur le rendement de la mémoire intégrée.

Une mémoire intégrée de type SRAM ainsi qu'un bloc de commande pour le Gestionnaire de Fréquence Programmable ont été conçus, mis en application, intégrés dans un système-sur-puce et testés. La thèse inclut aussi deux nouveaux circuits qui améliorent de manière significative le rendement et la fiabilité de la mémoire intégrée.

Cette thèse décrit une nouvelle architecture de mémoire intégrée pour mettre en valeur le rendement, la performance et la consommation de puissance. L'architecture de la mémoire peut tolérer des problèmes tel que le défaut de destruction. Le modèle mathématique de la nouvelle architecture présente et montre les avantages de l'architecture. Le nouveau Code de Correction des Erreurs (CCE) induit pour la mémoire dynamique à accès sélectif à multi-niveaux (MLDRAM) est présenté. Le nouveau CCE peut corriger une erreur de 2-bit et détecter une erreur de 4-bit. Le nouveau CCE améliore également la fiabilité et la consommation de puissance de la MLDRAM intégrée.

Acknowledgments

I would like to express my deepest gratitude to all the people who helped me during work on this thesis. The first man whom I would like to thank is my supervisor professor Zeljko Zilic. He provided me with the guidance throughout this thesis preparation and taught me several courses. He also gave me immeasurable amount of advise, not only in the microelectronics domain, but in everyday life as well.

I would like to acknowledge the Canadian Microelectronics Corporation (CMC) for fabricating custom McSOC chips through access to Taiwan Semiconductor's CMOS processes.

My parents, my brothers, my sister and all my family for spiritual support "through the ocean". They were the source of constant support, help and encouragement for me even though they were thousands miles away.

Two men that were always ready to sacrifice their time for me providing valuable advice and help on everything about anything in infinite world of microelectronics: Ian Brynjolfson and Nazmy Daniel-Abaskharoun. Also I would like to thank my two most valuable teaching assistants Ramez Rafla and Geoffrey Duerden. They shared with me a lot of "tricks" that came only with practical experience and could not be found in the text books. I also had the privilege to work days and nights, share ideas with: Henry Chan and Yanai Danan (McSOC team) with whom we created several practical designs.

Professors Nicholas C. Rumin and Gordon W. Roberts deserve special thanks for teaching me "Introduction to Digital VLSI" and "Analog Microelectronics" courses. All the members of Microelectronics and Computer Systems (MACS) laboratory that tolerated me for two years and left deep impressions on me. Katarzyna Radecka, Arshan Aga, Lige Wang, Ahmed Mostafa, Sebastien Laberge, Mourad Oulmane, Mohamed

Hafed, Bardia Pishdad, Tommy Tsang, Antonio Chan, Clarence Kar Lun Tam, Kong Xiaohua, Francis Beaudoin, Mark De Clercq, Stuart M^{c}Cracken, Atanu Chattopadhyay, Rola A.Baki and Mona Safi.

System administrators Ehab Lotayef, Ben Mihailescu and Weiwen Zhu put immeasurable effort to keep the computer system working days and nights.

Contributions of Authors

The core of this manuscript is dedicated to investigation of new methods of fault-tolerance of embedded memories. Chapter 1 presents brief overview of industrial trends in embedded memory domain and shows motivations to future work in fault-tolerant memories field.

Chapter 2 describes main topics that are necessary to understand in fault-tolerant memories domain. This chapter briefly overviews basic memory types widely employed in mass production. It also analyzes novel memory types such as Multilevel Dynamic Random Access Memory (MLDRAM) and Ferroelectric Random Access Memory (FRAM).

The project described in Chapter 3 is a collaboration of 5 people. Head manager of the designed project was prof. Zeljko Zilic and the different parts of the project were implemented by four students: Ian Brynjolfson, Henry Chan and the author of this thesis Boris Polianskikh. The project describes novel System-On-Chip (SOC). The the presented SOC consists of Programmable Clock Manager, Noise Modeling Circuitry, Processor and Embedded Fast Static Random Access Memory (FSRAM). Chapter 3 precisely describes the implementation of Fast SRAM. The results obtained from practical design of this chapter were used for further investigations in embedded memory domain.

Chapter 4 introduces new concept of Induced Error-Correcting Code (ECC) for 2-bit-per-cell Multi-Level DRAM. The ECC is able to correct 2-bit error and detect 4-bit error. The ideas outlined in this chapter were coauthored with professor Zeljko Zilic and published in [1].

Chapter 5 describes new model of Embedded Memory Architecture for Enhanced Yield, Performance and Power Consumption. The model protects the embedded memory from

major types of faults, such as row, column damage, chip-kill defects and cluster defects. The ideas of this chapter were published in [2] and were also coauthored with supervising professor Zeljko Zilic.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction and motivation

The century of the System-On-a-Chip is approaching quickly and it is the moment for the semiconductor industry to analyze the reality. Generally, the SOCs have been defined as the microelectronic systems with embedded memories, processors, analog parts and other specific functions organized on the same chip. A typical System-on-Chip is shown in Figure 1.1

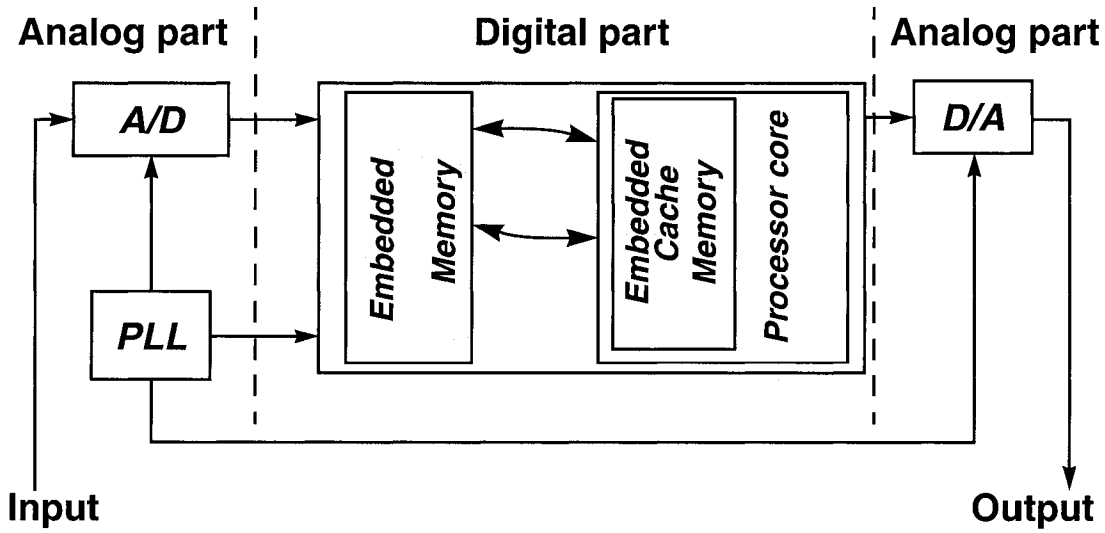


Figure 1.1: Typical structure of a System-On-Chip

The presence of digital and analog components on the same die makes them much more complex than the basic semiconductor blocks. Statistical data says that the System-On-Chip production increased in 1999 up to 345 million units, compared to 160 million units in 1998, which makes it 116% increase over one year. Economic experts predict the number of manufactured SOCs will be as many as 1.3 billion units in year 2004. Recently, the SoC technology has been mostly employed for communications system applications. For example, in 1999, the communication systems were approximately 39% of the whole SOC market. In-Stat foresees [3] that consumer products will jump up to 43% in year 2004, reaching the 310 million units. However, the communication systems still will be the biggest consuming client of the SOCs with approximate gross consumption about 576 million units.

Despite the fact that the SOCs are well defined and managed process, the SOC designers still face a lot of challenges during their design and implementation. The *International Technology Roadmap for Semiconductors (ITRS)* announced in 1999 that one of the most important SOC design challenges is the creation of the unique process design that allows using only standard CMOS flow for SOC implementation. The *ITRS* introduces the new definition of circuit "fabrics". The circuit "fabrics" denotes the microelectronics designs that demand different processes for its manufacturing and have particular yield and functional density. For example, radio frequency design and embedded memory design are two separate "fabrics", because embedded memory can be implemented using simple CMOS process and radio frequency design demands BiCMOS technology. There are approximately nine different design styles that can be used in today's SOCs designs. These are custom static CMOS logic, embedded SRAM, embedded DRAM, custom dynamic CMOS logic, analog circuits, radio frequency circuits, processor core, standard-cell auto place-and-route logic, and regular logic structures (data path). The goal of the modern SOC designer is to be able to combine all these different design styles on the same chip using the same standard CMOS technology.

Another challenge is how to estimate the SOC fabrication cost. Previously, the chip cost was defined only by wafer cost. The wafer cost, in turn, was dependent only on a

particular process technology of a particular design style. Meanwhile, recently the SOC fabrication cost depends on much more components as in Equation (1.1)

Total cost = RF cost + Analog cost + Digital cost + Packaging cost + Cost of testing (1.1)

which includes cost of each design style plus the combined testing and packaging cost.

If a designer wants to integrate additional design style or "fabrics" on the SOC, it is necessary to think about how to make it without overall impact on the power consumption, reliability, yield, performance and cost of the whole SOC. Additional integration of the new styles may require additional processing steps, such as special wiring and metallization for high-performance analog and radio-frequency circuits, as well as trench capacitor with *3D* structure for embedded DRAMs. For example, *International Technology Roadmap for Semiconductors* estimates that the addition of the embedded flash memory or ferroelectric RAM to already existing embedded SRAM will increase the mask levels by four additional levels, and that addition of the embedded flash memory to standard CMOS logic requires 4 extra masking layers. Each additional masking layer creates additional threat to yield and reliability of the whole SOC.

One might ask, if there are so many differences between implementations of the embedded memory and standard logic components that create difficulties, why not retain the old method of fabricating the memory and the logic on different chips?

Routing delays between components are becoming the major factors affecting the speed of the SOC components. Even though these delays play significant role in SOC performance, they are still much shorter than delays that occur if a designer tries to drive signals off-chip to a separate memory bank. Driving signals off-chip also reduces flexibility of the design. For example, widening off-chip bus to increase buswidth negatively impacts the number of I/O pins. The off-chip interconnections require area hungry I/O buffers, in order to be able to overcome package and board-trace impedances. The consequences of addition of I/O buffers are increased power consumption, limited battery life, and reduced reliability.

Embedded memory represents perhaps one of the most important components of the SOC, without which digital part and consecutively the whole SOC will be paralyzed. Sometimes, depending on the embedded memory roles within the SOC, they can occupy approximately 90% of its real estate. Eventually, embedded memories are going to occupy most of the area of the SOC. Today's SOCs characteristics include the following: usually there are more than 30 embedded memories within the SOC, many of which are of different types and sizes. Normally, embedded memories are located all over the SOC, and embedded memory testing is available only from a few I/O pins.

Embedded memory density normally increases as many as four times from one technology generation to another. The change of technology has both positive and negative impacts on the SOC. The main positive impact is the reduction of the occupied area. The main negative impact is the impact on yield. As shown in Figure 1.2, one of the designs is implemented in 0.35 μm technology and another one in 0.18 μm technology. The defect cluster of the same size destroys as many as four times more components in 0.18 μm technology than in 0.35 μm technology.

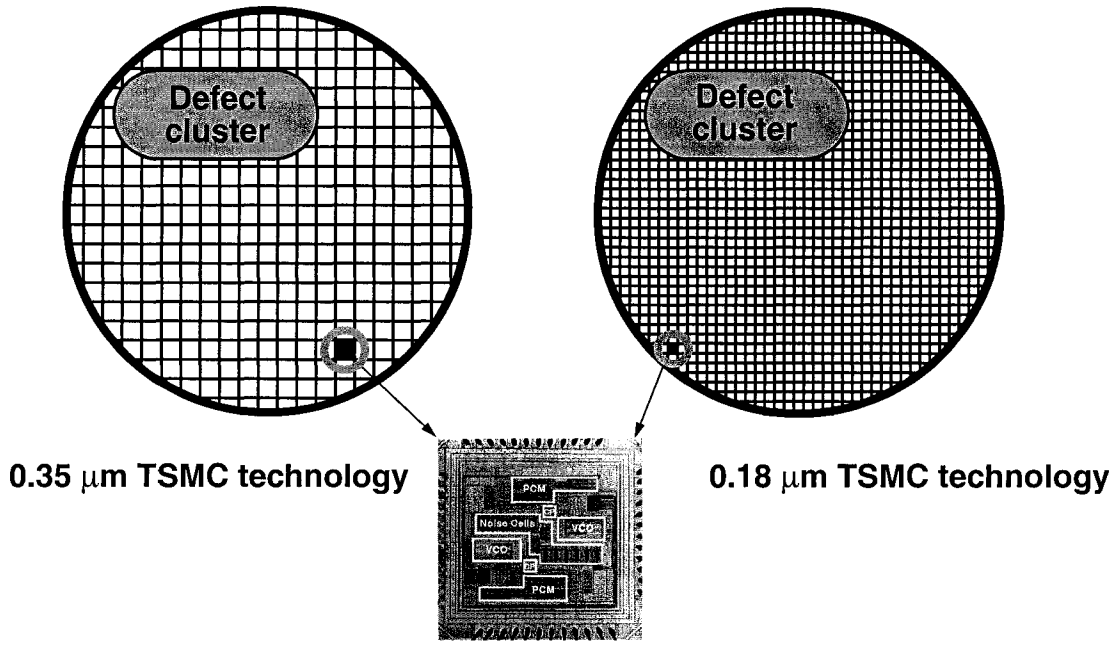


Figure 1.2: Technology shrinkage affects yield of the embedded memory

Embedded memories are the most dense components of the SOC. Modern semiconductor industry makes fast advances towards even more sophisticated and finer embedded memory designs. New generation of *multilevel DRAMs* enables the designer to get even more dense embedded memory. Multilevel DRAM stores 2-bit-per-cell as a different voltage levels on a storage capacitor. At the same time, moving from one technology to another creates a lot of side effects that the designers never encountered before. DRAMs, and even more multilevel DRAMs, are much more sensitive to process variations and manufacturing defects than logic components. Their sensitivity can be compared to the sensitivity of extremely sensitive analog components. It is becoming impossible to design large embedded memory banks without solving the yield problem. Embedded memory bigger than 2-Mbit can not be designed without yield protection circuitry [4].

Usually, the embedded memory without any protection can be only up to 10% of yield or even less. Today, the embedded memory designer has two major methods to increase memory yield and reliability: redundancy protection and error-correcting codes. With fast advances of microelectronics industry towards MLDRAMs and trench three-dimensional capacitor, embedded memory designers have to invent new methods of protection for embedded memory, for example, something like three-dimensional error-correcting codes or three-dimensional redundancy.

All previously described, the modern semiconductor industry is in high demand for well protected, fault-tolerant SOC. The embedded memory is an increasingly important component of any SOC, whose yield and reliability dramatically affect SOC performance.

Recently, many companies continue to perform a research in order to create well-protected, reliable embedded memories for specific application. For example, the company Virage Logic fabricated and put in mass production the STAR Memory System (SMS), which contains one or more memory blocks, processor, and a fuse box. The system allows cost-effective embedding, testing and repairing of multi-megabit memories on any SOC (www.viragelogic.com). That is why the goal of this thesis is to investigate new methods of embedded memory protection.

1.1 - Thesis outline

A brief introduction to embedded memory types and the methods of embedded memory fault-tolerance and yield improvement are provided by Chapter 2. This includes a discussion on main embedded memory types employed in industry, such as SRAM and DRAM memories. Chapter 2 also describes novel Multi-Level Techniques that enable memory to store two bits per cell. The main multi-level memory types are ferroelectric, Flash and multi-level DRAM. Main principle, on the basis of which each multi-level memory type is functioning, was reviewed. Detailed comparison was performed and all pros and cons were summarized. There are two main methods of embedded memory protection: redundancy and error-correcting code. Both methods are described and benefits, drawbacks, and technical challenges are summarized.

The practical exploration of the embedded memory designs begins with Chapter 3. The chapter explains the design and implementation of System-On-Chip. There were several implementations of the System-on-Chip. Each implementation is called *pilot chip*. The Control-Block was designed for the first pilot chip. The Control Block serves to adjust a Programmable Clock Manager (PCM), which is also implemented on this chip. The second pilot chip is more sophisticated and includes an embedded memory core, a processor, external communication subsystems, an internal asynchronous bus and the PCM. The fast Static Random Access Memory is designed and embedded on the second chip. The embedded memory consists of the memory core, row and column decoders, precharge circuitry, sense amplifiers and peripheral circuitry. After fabrication, we developed a research platform, and the embedded memory was tested on the basis of the developed platform.

The research performed in Chapter 3 leads to the new ideas. One of the ideas is how to improve reliability and performance of the Multi-Level DRAMs. Although some designs of the MLDRAM exist they still can not be fabricated for mass production. The improvement of the reliability of the MLDRAM is one step further towards mass production. Thanks to designed Error-Correcting Code (ECC), Multi-Level DRAM is able to tolerate two-bit errors and detect four-bit errors. The proposed ECC is implemented on

the basis of the popular Hamming ECC. Practical implementation is described in Chapter 4.

The problem of yield improvement has been known to memory designer for years. One solution on how to improve yield of the embedded memory is given in Chapter 5. New memory architecture significantly increases embedded memory yield. The architecture enables memory to tolerate not only common types of faults, such as row/column failure and sense amplifier failure, but also such types as chip kill defects. Performed research resulted in mathematical model designed for the new embedded memory architecture. The model shows that skillful redundancy allocation improves yield much better than just increasing the number of redundant rows or columns.

Chapter 2 - Fault-tolerant embedded memories

2.1 - Types of embedded memories

There are two types of CMOS memories that are mostly employed in modern industry. Static RAM (SRAM) uses a latch configuration to store data and Dynamic RAM (DRAM) stores binary data as a charge on a capacitor.

2.1.1 - Static Random Access Memory

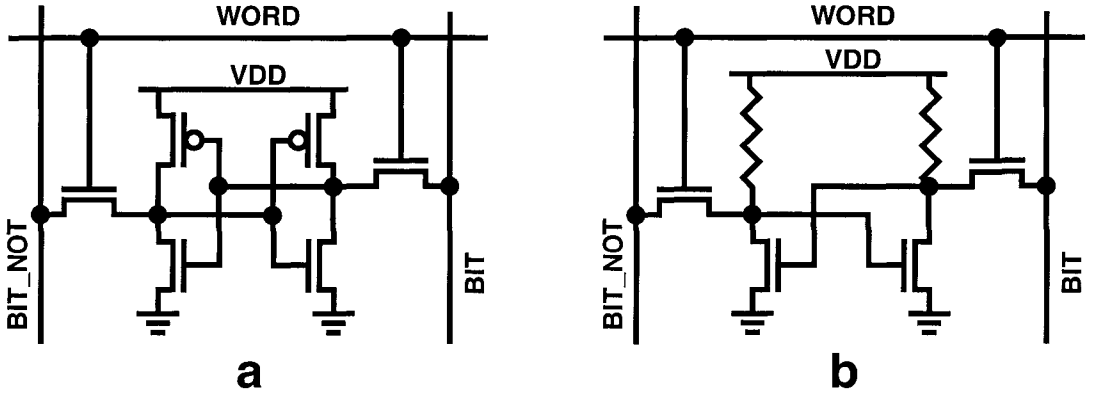


Figure 2.1: Static RAM memory cells,
(a) six-transistor cell, (b) four-transistor cell

Figure 2.1 (a) shows the generic SRAM cell which consists of six transistors. This cell is actually two cross-coupled inverters and two pass transistors. In order to write data to a cell, the information bit and its complement must be put on bit lines. The complementary value is not really necessary, but doing so improves noise margins and makes memory more reliable.

Another type of basic SRAM cell, very often employed by memory designers, is that of the four-transistor cell. The architecture of this cell is similar to the previously explained cell with the only difference that PMOS transistors are replaced with two resistors. In comparison to the six-transistor cell, the four transistor cell occupies less space, but its access time is longer and is more susceptible to various damaging effects due to lower noise margins.

2.1.2 - Dynamic Random Access Memory

Another type of conventional memories is the Dynamic Random Access Memory. The two main configurations are shown in Figure 2.2.

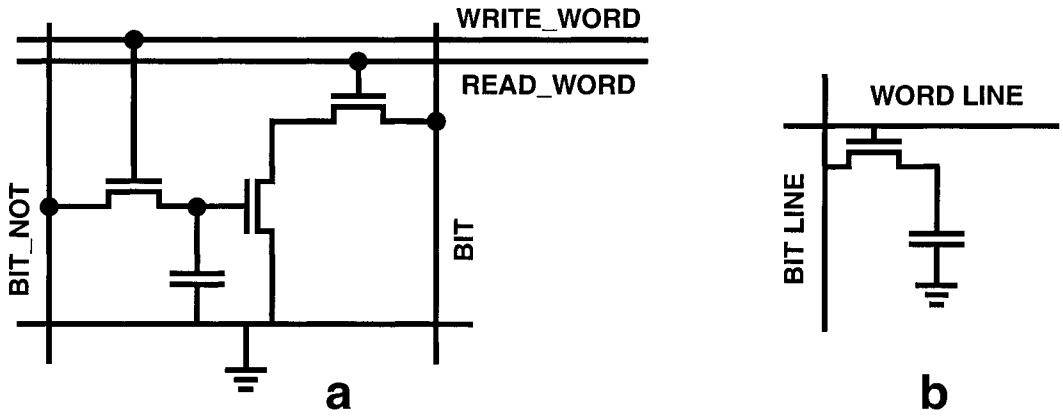


Figure 2.2: Dynamic RAM memory cells,
(a) three-transistor cell, (b) one-transistor cell

DRAM configurations are especially attractive to the industry because of their high densities and easy implementation in CMOS technology.

Three-transistors DRAM cell (Figure 2.2(a)) is employed in circuits which usually contain logic and the memory to support the logic. They are very rarely used for the memory-only chips. They can be easily implemented with simple CMOS technology and have faster read and write operation times than those of one-transistor DRAM. As usual, there are the pros and cons. Since the storage capacitance in a three-transistor cell is much smaller than the storage capacitance in a one-transistor cell, the three-transistor DRAMs have lower noise margins in comparison to one-transistor implementations.

The densest memories that are employed by semiconductor industry are the memories implemented on the basis of the one-transistor cell. As shown in Figure 2.2 (b) the cell consists only of one pass transistor, which serves as a switch for data access, and one capacitor, which serves as a data storage device. The main features that attract embedded memory designers to one-transistor DRAM are:

- Small area occupied on a silicon surface.
- A lot of cells can be tied to the same bit line.
- Large noise margins.
- Long time between refresh cycles.
- Low power dissipation.

2.1.3 - Multi-Level Dynamic Random Access Memory

Another very interesting type of embedded memory is the *Multi-Level DRAM (MLDRAM)*. MLDRAM differs from conventional DRAM by storing more than one bit per storage cell. The main idea behind the MLDRAM principle is that the binary data is stored on the capacitor as different voltage levels. For example, the two bit data is stored by using four different voltage levels GND, one third of VDD, two third of VDD and VDD. MLDRAM can be implemented on the basis of the same technology as usual DRAM. As a trade-off for higher density, the read and write times are much higher for MLDRAM than for conventional DRAMs. Another drawback is that due to even more

reduced noise margins, MLDRAM is very susceptible to faults. MLDRAM has very good outlook, since the only way to increase density of the memory right now is to reduce transistor size. Recently it has become possible to implement transistor with the size of ten atomic layers. Eventually, it is reasonable to assume that transistor size might be reduced up to one atomic layer. The only way to increase density will be using MLDRAM principle. Although MLDRAM is not in a mass production due to reliability concerns, the full scale research is performed in order to be able to implement MLDRAM for mass production in industry.

2.1.4 - Ferroelectric Random Access Memory

Ferroelectric memories (*FERAM or FRAM*) can also store several bits in one cell. Basic architecture (Figure 2.3) of the FRAM resembles one-transistor DRAM structure and consists of a ferroelectric capacitor and a pass transistor. The information is stored as a direction of the ferroelectric polarization. The polarization *P* versus electric field *E* shows hysteresis behavior, with P having two stable values (plus and minus) at 0 field conditions. The (reverse) electrical field that must be applied to annihilate the existing polarization (P=0) are termed the *coercive* fields.

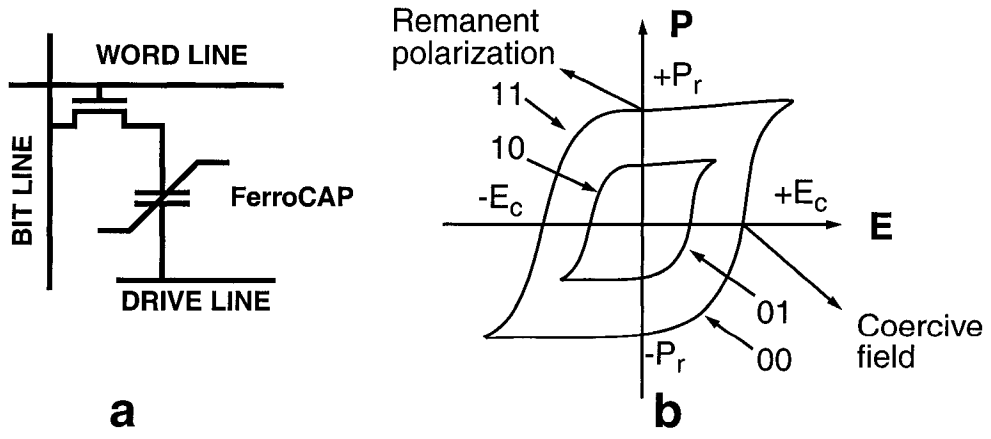


Figure 2.3: Ferroelectric memory structure and operation,
(a) basic cell structure, (b) polarization vs. electric field.

Ferroelectric memory seems to be a good alternative to CMOS MLDRAMs to store several bits in one cell. However, the main reason that prevents designers from using ferroelectric memories on a wide scale is a fatigue problem. Electrical fatigue is defined as the reduction of the switchable polarization with increasing the number of switching cycles. This fatigue limits the endurance of the memory, i.e. the maximum number of read/write cycles that can be applied to a memory element still allowing discrimination between the four memory states (i.e. polarization "up" or "down"), as shown in Figure 2.3.

2.1.5 - Flash memory

Multi-level Flash memory is based on the ETOX^{TM} process. The simplified diagram of the ETOX cell, that stores multiple bits in floating gate, is shown in Figure 2.4. Data is written into the memory with *programming operation,* by exposing the floating gate to sufficiently high voltage. The different levels of voltage are stored as the number of electrons on the floating polysilicon level or storage poly. As soon as data is written, it will be stored in the cell, with or without power supply. The only way the data can be removed from the cell is with *erase operation,* by applying high voltage to the gate and discharging the floating gate. Data is stored on the separate mask layer called *storage polysilicon.*

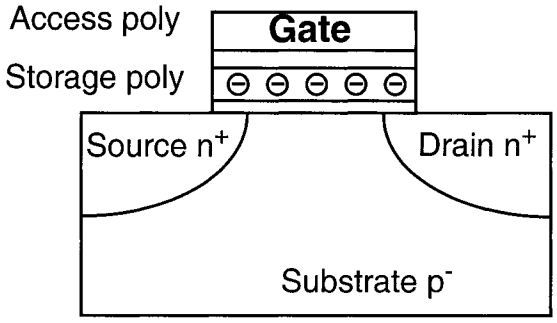


Figure 2.4: Multilevel Flash memory

The advantages of the Flash technology are the direct access to the cell, and reliable charge placement. Although the Flash memory is now widely employed in industry, problems still exist.

One of the major problems occurs when data in Flash memory needs to be changed dur-

ing its operation; memory can not be read out until the write cycle finishes. The microprocessor can not read the data from the flash memory while a word was programmed or one of its blocks was erased. Thus the processor that uses the flash memory needs additional SRAM block in order to operate during flash memory update. There are also other issues such as high susceptibility to noise, because the flash memory cell in its concept is mainly an analog device.

Memory types	Flash memory	Ferroelectric memory	MLDRAM
Storing device	Floating gate	Ferroelectric capacitor	CMOS capacitor
Methods of data storing	Varying threshold voltages	Varying polarization directions	Different level voltages
Volatility	Nonvolatile	Nonvolatile	Volatile
Technology	ETOX^{TM}	Commercial ferroelectric process	Standard CMOS DRAM process
Commercialization	Yes	Yes	No
Additional high voltage needed	Yes	Yes	No
Typical capacity	8-256 Mbit	~256 kBit	~4 Gbit
Typical lifetime	100.000 erase cycles	100.000 write/read cycles	∞

Table 1: Multi-level memory characteristics

The summary of main characteristics of multilevel memories are presented in Table 1. The table shows that multilevel DRAM is the largest of all the multilevel memories. It also has the longest lifetime, and it does not require additional processing steps to implement in CMOS process.

2.2 - Common embedded memory architectures

Basic memory architecture generally contains memory core and peripheral circuitry. As shown in Figure 2.5, memory core is organized as a matrix of rows and columns. Depending on the memory type, peripheral circuitry contains amplifying buffers, row and column decoders, sense amplifiers, precharge circuitry, memory controller and refresh

circuitry. For example, if sense amplifiers and precharge circuitry are used for SRAM memory only to improve the speed of the read operation, for DRAM they are the essential blocks, without which the read operation is impossible. Also, refresh circuitry is necessary only for DRAM types of embedded memories, due to capacitor leakage. As shown in Figure 2.5, sometimes the destruction of one of the vital peripheral blocks means the destruction of the whole memory. For example, if amplifying buffers are damaged, the signals from row decoder can not propagate to the memory core and the data stored in memory core cells can not be accessed. As a result, the memory is completely dysfunctional.

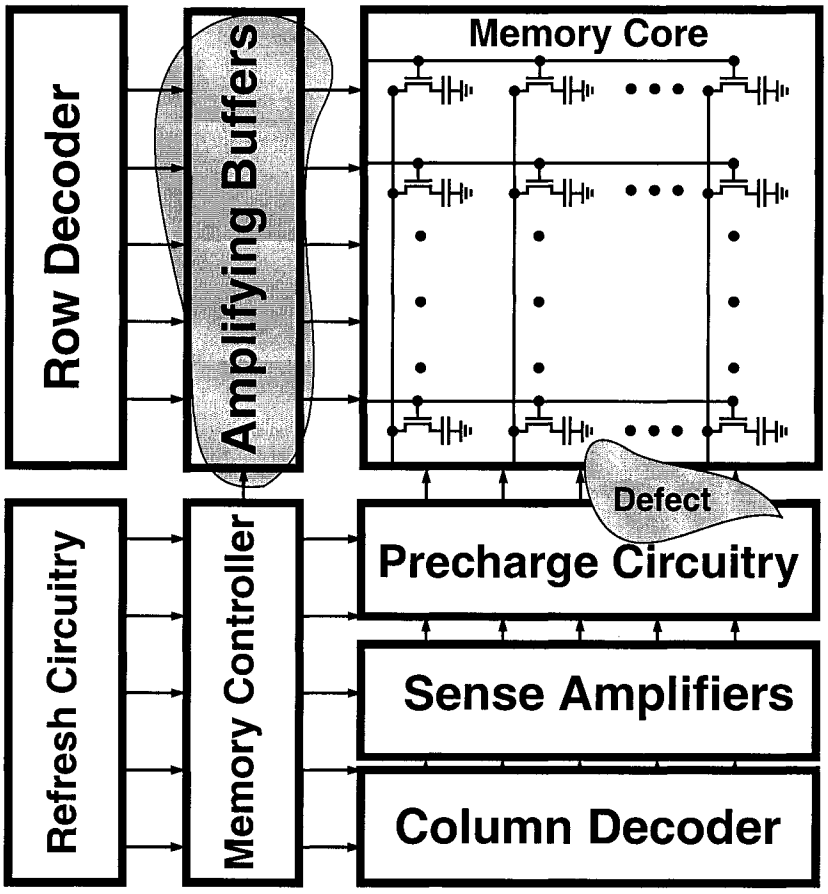


Figure 2.5: Basic Memory Architecture

Another example shows partial destruction of the precharge circuitry and its interconnections. The result of this damage is the partial memory core loss. Of course, it is

impossible to predict where the damage cluster will occur and how big it will be. It is evident from previous examples that during embedded memory design process it is necessary to think not only about memory speed, area and overall performance, but also about memory's ability to tolerate any type of damage that is possible for given process.

2.3 - Embedded Memories and Fault-Tolerance

At this point it is reasonable to ask, why the fault-tolerance is so important for embedded memories and consequently for System-On-Chip. Both non-fault-tolerant design and the fault-tolerant design work properly while being designed in the computer environment.

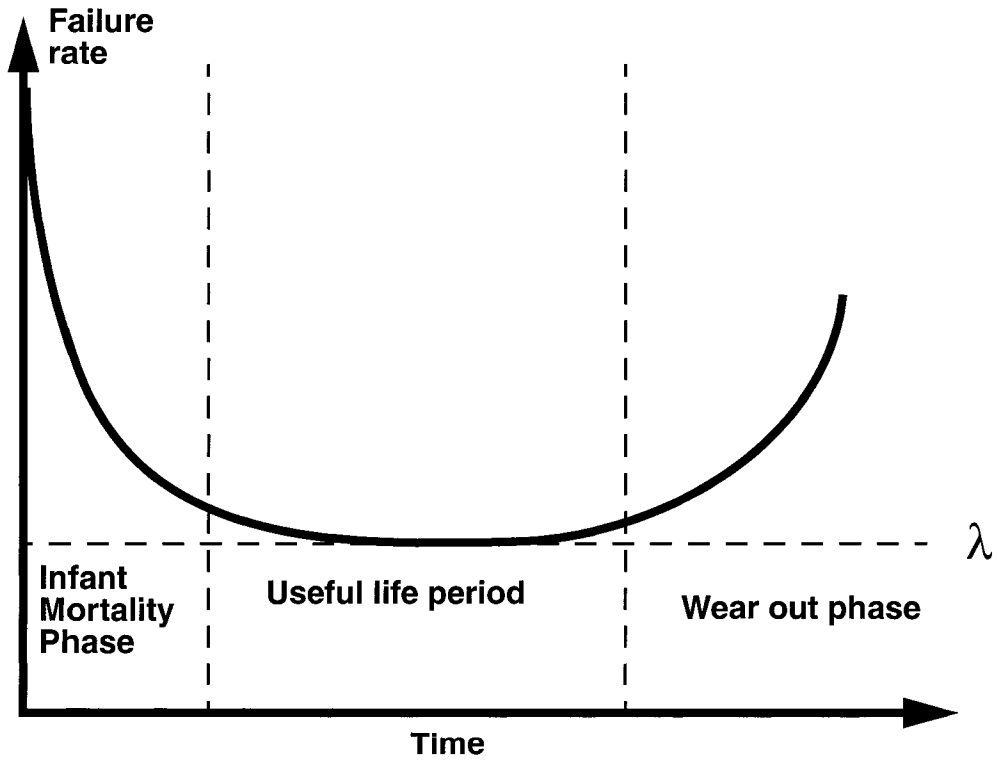


Figure 2.6: Relationship between failure rate and product lifetime

After fabrication stage, the non-fault-tolerant system tends to fail due to a lot of secondary effects; meanwhile, the fault-tolerant system is able to tolerate all side effects

and still perform correctly. From this point, fault-tolerance of the embedded memory can be defined as the ability of the memory to perform correctly after the occurrence of faults. Yield and reliability are the two main quantitative measures that describe the level of fault-tolerance of the given system. The probability of the system failure over the system lifetime can be described with the curve shown in Figure 2.6.

The curve is divided in three parts. The first part shows very high probability of memory failure at the beginning of the embedded memory performance, which is right after chip fabrication. This part is called *infant mortality phase*. The high rate of the memory failures at this stage is explained by several reasons, such as weak components, missing metal layers and so on.

The second part shows *useful lifetime phase*. At this stage, all possible failures that could happen during the first stage are eliminated, and the system functioning depends only on λ. The λ parameter is often referred to as a failure rate, i. e. number of failures per unit of time. Eventually, the lower λ parameter is, the less probability that the memory is going to fail. This means that the level of fault-tolerance is inversely proportional to the failure rate λ and the goal of any designer is to make λ as small as possible.

The third part shows *wearout phase*. The high rate of failures during this phase happens due to fatigue of electronic and mechanical components of the system. The good example for this phase will be the restricted number of read/write cycles for ferroelectric memory.

2.3.1 - Faults, errors and failures classification

Now is good moment to introduce three terms that are very useful for describing fault-tolerant system. These terms are *fault*, *error* and *failure*.

Figure 2.7 shows basic types of faults, errors and failures that occur in embedded memories and relationships between them. Fault is a primary damage that leads to an error. And error, in turn, provokes failure.

A *fault* is a physical flaw or physical defect that occurs due to several factors, such as fabrication impurity, process imperfections, fatigues, deteriorations, ionizing radiations, humidity, electromagnetic interference, internal and external noise and so on. Common types of faults are missing layers, shorts, and damages of semiconductor devices.

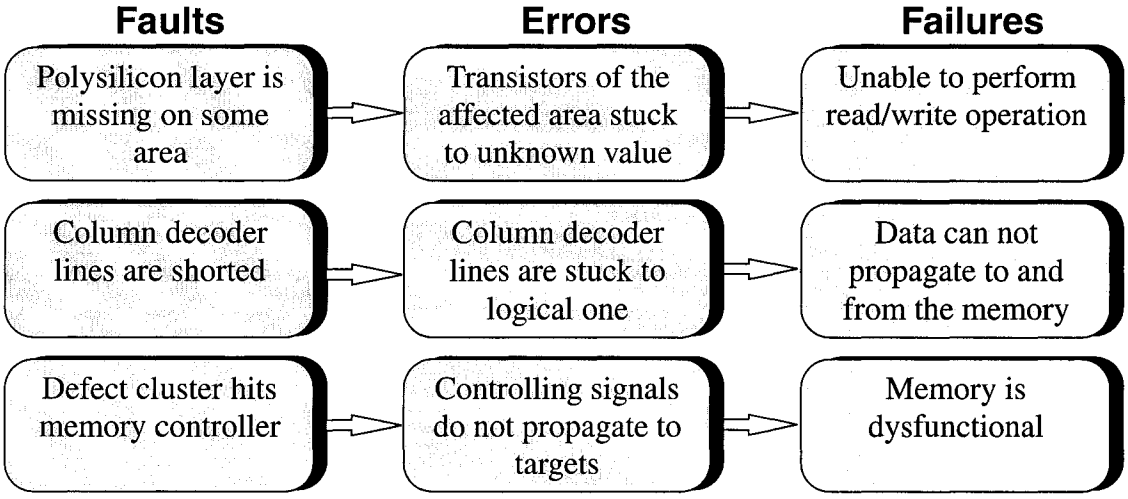


Figure 2.7: Links between faults, errors and failures in embedded memories

An *error* is a consequence of a fault. It is a deviation from a normal state. For example, the logical state of the cell or peripheral elements is different from its intended state. Errors are classified in two categories: hard and soft errors. A hard, or permanent error is a result of the damage that persists through the whole life-time of the embedded memory. A soft error is a temporary error. It does not persists after fault-causing phenomenon is disappeared and needs to be repaired only for a short amount of time.

A *failure* is the inability of the memory to perform required function. Failure affects the whole system on a high level of abstraction. The failures itself can be divided in three groups. *Soft failure* consequences may be negligible, for example, the memory failure of the personal computer. *Firm failure* is the failure that affects the system but can be tolerable by the system for some period of time needed to repair it. And finally, *hard failure* is a failure that is not tolerable by the system at any time. A good example for hard failure can be the total memory failure in the control block of the airplane, during flight.

Since small faults can lead to catastrophic consequences, such as total system failure, the primary goal of the modern fault-tolerant embedded memory designer narrows down to a single goal: to make the embedded memory as tolerant as possible to any kind of faults.

2.4 - Yield and yield modeling

The definition of the word *yield* came to English language from the Old High German language word *geltan* which literally means to pay. This definition implies that yield is the benefit the designer obtains after actual fabrication of the System-On-Chip. Mathematical descriptions of the yield for embedded memories are varying in different sources, but common idea can be expressed as follows. Yield, in our case, is the number of fabricated embedded memories that perform correctly (fully functional memory), divided by the total number of fabricated embedded memories. Fully functional memory means that all memory cells and all peripheral circuitry perform correctly all operations, i.e. read, write, access, storage, as planned by a designer before fabrication stage.

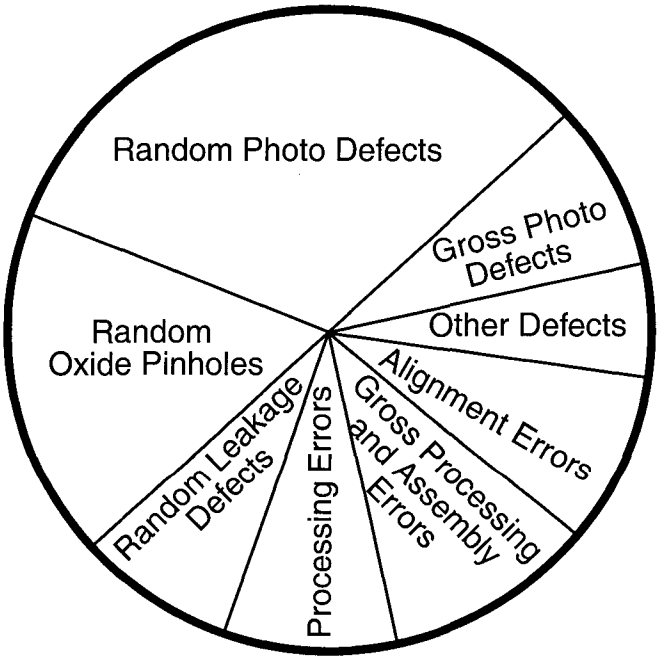


Figure 2.8: Factors that affect yield after fabrication of the memory

There are several factors that affect yield of embedded memory during the fabrication stage [5]. The major factors are random photo defects, random oxide pinholes, random leakage defects, gross processing and assembling faults, specific processing faults, misalignments, gross photolithography defects and other minor defects (Figure 2.8).

2.4.1 - Yield modeling

Yield of any embedded memory can be described using various mathematical models. Each model is constructed on a basis of empirical data obtained during previous fabrications for the given process parameters. The yield model describes yield as a probability of defects happening. Some models try to describe yield assuming cluster distribution of the defects throughout the area of the embedded memory ([6], [7], [8]). Conventional models describe yield of a memory in which the defects randomly and uniformly distributed throughout the area of the memory.

The yield models differ in degrees of complexity, usually the greater the number of factors included in a model, the better the model describes yield. The simplest model describes yield of embedded memory as a probability that embedded memory will function correctly as follows:

$$P(memory\ works) = e^{-A_{memory} \cdot D_0} \tag{2.1}$$

This model takes in account only A_{memory} - the whole embedded memory area with peripheral circuitry and D_0: defect density. Defect density is the expected number of defects per unit area, estimated on the basis of empirical data or previously fabricated chips. It is clear from Equation (2.1) that, in order to get better yield, one has to keep area of the memory as small as possible. With the growing demands of the VLSI industry, it is almost unreal to operate with small memories. This means that the embedded memory designers find themselves faced with the fact that in order to increase yield, they have to decrease D_0 or to find other ways to improve yield.

The Equation (2.1) simply gives us the probability of a specific memory with A_{memory} and D_0 being completely operational. But when fabricating several chips with the same

area A_{memory}, usually they all have different defect densities. To account for this effect, a new function is introduced. This function $F(D_0)$ is the distribution function of the defect density D_0. With new function $F(D_0)$ the obtained yield is going to be more precise and can be calculated as in Equation (2.2).

$$Y = \int_0^{\infty} e^{-D_0 A_{memory}} F(D_0) dD_0 \tag{2.2}$$

The main defect distributions used for yield description are shown in Figure 2.9.

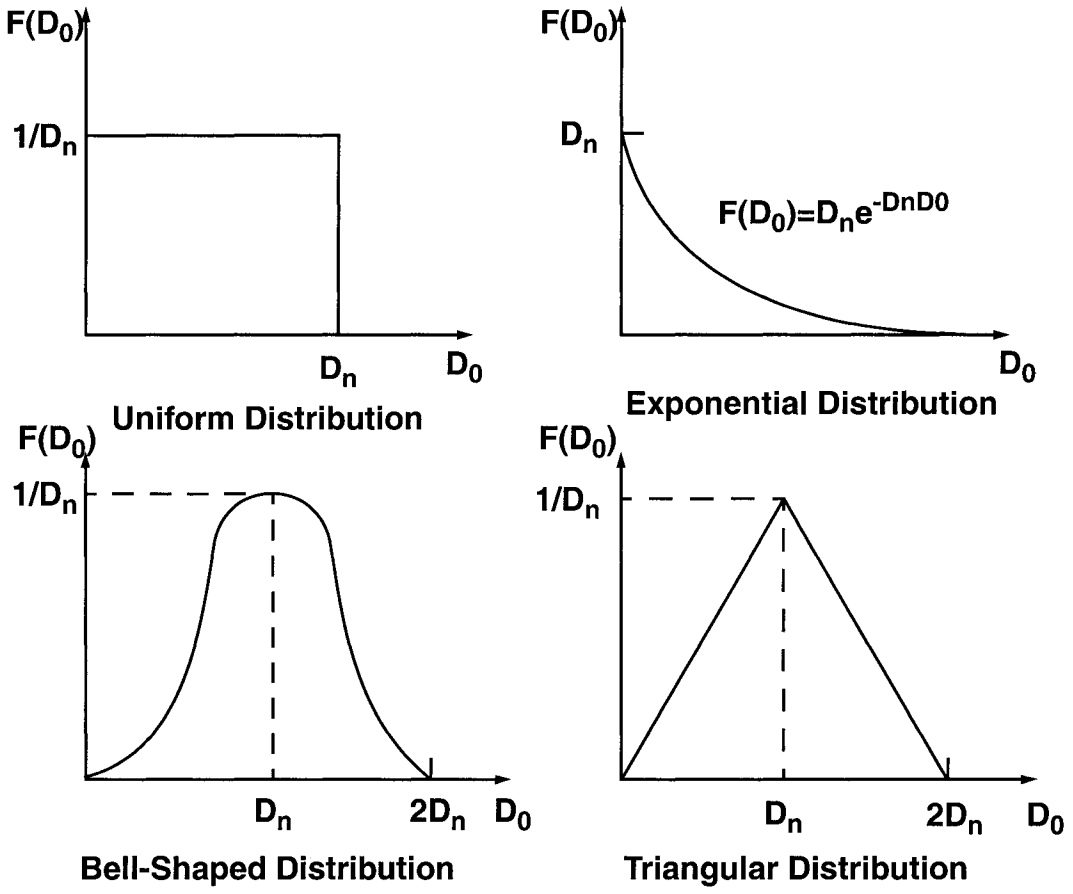


Figure 2.9: Common defect distributions used in yield analysis

Integral of the $F(D_0)$ over all possible values of D_0 must be one. Uniform distribution from Figure 2.9 considers $F(D_0)$ to be uniform between D_0 from 0 to some certain value D_n. The yield for uniform distribution will be given as in Equation (2.3)

$$Y = \int_0^{D_n} e^{-A_{memory}D_0} \frac{1}{D_n} dD_0 = \frac{1}{D_n} \int_0^{D_n} e^{-A_{memory}D_0} dD_0 = \frac{1}{A_{memory}D_0}(1 - e^{-A_{memory}D_0}) \quad (2.3)$$

The uniform distribution is the simplest yield model. If we want to get the more realistic yield model, we should use more complex model such as triangular and bell-shaped distributions models (Figure 2.9). Practically, these distributions are very similar to each other. Because triangular distribution is simpler and easier to operate with, it is very often used as a fast approximation to the bell-shaped distribution. The triangular distribution function is given as following:

$$F(D_0) = \begin{cases} \dfrac{D_0}{D_n^2} & if \quad 0 \le D_0 \le D_n \\ \dfrac{2D_n - D_0}{D_n^2} & if \quad D_n \le D_0 \le 2D_n \end{cases} \quad (2.4)$$

The yield for triangular distribution function is given by

$$Y = \int_0^{D_n} e^{-A_{memory}D_0} \frac{D_0}{D_n^2} dD_0 + \int_{D_n}^{2D_n} e^{-A_{memory}D_0} \frac{(2D_n - D_0)}{D_n^2} dD_0 = \left(\frac{1 - e^{-A_{memory}D_n}}{A_{memory}D_n} \right)^2 \quad (2.5)$$

All previously explained models are simplified models. In order to predict yield as precisely as possible, a lot of factors have to be considered for these equations. As shown in Figure 2.5, the embedded memory consists of blocks, such as embedded memory core, refresh circuitry, row/column decoders and many additional blocks. Of course, it is reasonable to assume that some blocks of the memory are more susceptible to defects and others are less. For example, sense amplifiers are very susceptible to manufacturing variations than other parts of the embedded memory, since all of the transistors inside of the sense amplifier have to be matched precisely. Since each part has different sensibility to the factors that affect yield after fabrication, it is reasonable to express yield of each block independently ($Y_{sense\ amlifiers}$, $Y_{refresh}$, $Y_{memory\ controller}$, $Y_{column\ decoder}$, $Y_{row\ decoder}$). Because yield of each block is completely independent of other blocks, the total yield of

the embedded memory must be expressed as a product of yields of all blocks of the embedded memory and is going to be as follows

$$Y_{total} = Y_{core} \cdot Y_{decoder} \cdot Y_{refresh} \cdot Y_{precharge} \cdot Y_{senseamp} \cdot Y_{controller} \quad (2.6)$$

2.5 - Reliability and reliability modeling

Reliability is another major factor that describes embedded memory fault-tolerance. Reliability *R(t)* is defined as a probability that the embedded memory will function properly throughout the interval of time $[t_0,t]$ with the condition that it was performing correctly at time t_0. Mathematically, reliability is better understood as follows. Assume that there are $M(t_0)$ embedded memories of identical size and characteristics, all of them simultaneously placed under the test at given time t_0. After time *t*, there are *N(t)* embedded memories that still perform correctly and *F(t)* embedded memories that failed since time t_0 ($M(t_0)=N(t)+F(t)$). The reliability of embedded memory is given by

$$R(t) = \frac{N(t)}{M(t_0)} = \frac{N(t)}{N(t)+F(t)} \quad (2.7)$$

Equation (2.7) simply expresses the probability that the embedded memory performs its designed functions under specified conditions in time interval $[t_0, t]$. The reliability can be characterized with λ failure rate and the *Mean-Time-To-Failure* (*MTTF*). Failure rate λ is defined in the same way as in Figure 2.6. MTTF is defined as an average time to failure. If there are $M(t_0)$ identical memories placed into functioning at time t_0 and we measure the time t_i each time one of the memories fails, the *MTTF* will be given by Equation (2.8)

$$MTTF = \frac{\sum_{i=1}^{M(t_0)} t_i}{M(t_0)} \quad (2.8)$$

If we assume that $t_0=0$ and $R(t)|_{t=\infty}=R(\infty)=0$ (there is no system that functions forever without failure) the relationship between *MTTF* and reliability will be given by Equation (2.9)

$$MTTF = \int_0^\infty R(t)dt \tag{2.9}$$

The MTTF can be also calculated with given λ failure rate. Since failure rate can also be expressed through reliability as in Equation (2.10)

$$\lambda(t) = \frac{R(t_0) - R(t_1)}{\Delta t R(t_0)} \tag{2.10}$$

where t_0 and t_1 are the start and the end times of the time interval $\Delta t = t_1 - t_0$. $R(t_0)$ and $R(t_1)$ are reliability at these times. Since the failure rate λ is the number of failures over a time interval Δt the *MTTF* can be derived as following

$$MTTF = \frac{1}{\lambda \Delta t} \tag{2.11}$$

Normally the time t_0 is equal to 0. Under this assumption the Equation (2.11) transforms to Equation (2.12)

$$MTTF(\Delta t) = \frac{1}{\lambda(t)} \tag{2.12}$$

2.5.1 - Reliability modeling

The reliability modeling can be done in two ways: either by using combinatorial model, or by Markov model. The main difference between these two is that combinatorial model describes reliability as a set of operational states such that the probabilities of each of these states can be described with combinatorial means. Meanwhile, Markov model describes the reliability as a probability of faulty transitions amongst all of the possible states of the system.

A good example of using the combinatorial model is a small memory consisting of 4 cells and protected with 2 extra cells. Having 2 cells in the memory means that memory can tolerate up-to 2 failed cells. Defining *P(t)* as the probability that 1 cell will perform

correctly until time t, and the $R(t)$ probability that the whole system (4 cells + 2 cells) will function correctly will be

$$R(t) = \sum_{i=0}^{2} \binom{4}{i} (P(t))^{4-i} (1-P(t))^{i} = 6P^2(t) + 3P^4(t) - 8P^3(t) \qquad (2.13)$$

The Markov model is shown in Figure 2.10. The system consists of four states A, B, C and D. The full lines show allowed transitions in the system and dashed lines show forbidden transitions between states. Basically, model operates with the probabilities of all possible transitions between all four states.

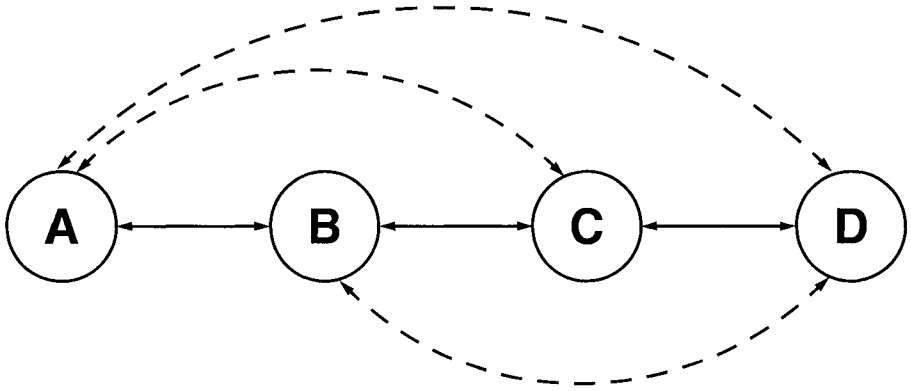


Figure 2.10: The structure of a Markov model of a 4 state system

2.6 - Methods of fault-tolerance improvement in embedded memories

Recently, fault-tolerance became a vital part of any embedded memory design. The definition of fault-tolerance includes protection of the embedded memory from any type of damages that may occur during normal memory performance. Before designing the embedded memory, the designer has to create the initial plan which takes care of the fault-tolerant side of the memory. The initial plan has to include prognosis of all the possible faults that may occur for a specific type of memory and how to reduce the effects that the fault may produce. The designer has to decide what type of memory he/she is going to use,

what types of faults may happen, and which types of protection circuitry he/she is going to use. The major factors affecting yield and reliability to consider at the initial stage of the embedded memory design are shown in Table 2.

Factors affecting yield	**Factors affecting reliability**	
Fabrication factors	**Hard error sources**	**Soft error sources**
Cleanness	Hot carrier emission	
Materials	Electromigration	
Oxides	Surface charge spreading	Surface charge spreading
Parameter spread	Ionic contamination	
Complexity	Spurious currents	Spurious currents
Control	Time dependent breakdown	
Temperature		Package α radiation
Mechanical shock	Static charge and discharge	
Design factors	Electromechanical corrosion	
Feature size	Electrochemical corrosion	Electrochemical corrosion
Chip size	Electromagnetic interference	Electromagnetic interference
Packing density	Temperature	Temperature
Cell type	Cosmic particle impacts	Cosmic particle impacts
Layout	Radiation total dose	Radiation total dose
Parameter variation tolerance	Transient radiation	Transient radiation
Pattern sensitivity	Mechanical shock	Mechanical shock
	Electrical shock	

Table 2: Errors, fabrication and design issues affecting yield and reliability

In the next stage, the designer has to consider is how to repair occurred damage. Basic repair procedure consists of four steps. The first step is to detect a fault or damage. The

second step is to locate it. The third step is the decision-making step, to decide which type of protection to use according to the event. The fourth step is assigning spare elements. And the last, fifth, step is to disconnect the faulty elements from the embedded memory.

The two main methods of protection can be used in order to insure the fault-tolerance of embedded memories. These methods are redundancy application and *Error-Correcting Code (ECC)* application.

2.6.1 - Redundancy application for embedded memories

The word *redundancy* in information theory was used for the first time by Nyquist around 1920. But for Nyquist the meaning of the word was negative, since he saw unnecessary sinusoidal component signal as a redundant one and carrying no sense for computing theory. For us, it signifies positive effect, the protection of the memory with spare elements. There are three types of redundancy used.

The first type is the *passive redundancy.* In the case of a fault, passive redundancy just tries to hide the fault and prevent the occurrence of error. The passive redundancy is designed in such a way that it does not require any activity on the part of the system or interference from the side of designer.

The second type is the *active* redundancy. Active redundancy performs some actions on the system in order to detect, locate and eliminate the fault. Active redundancy interferes in the system and changes its characteristics in order to provide fault tolerance.

The third type is the *hybrid* redundancy techniques. *Hybrid* redundancy uses both features of passive and active redundancies. It attempts to hide occurring faults and prevent the errors as passive redundancy does and it eliminates faults as active redundancy does. Hybrid redundancy is the most expensive of all of the previously described types of redundancies, because it demands more extra space on a die compared to passive and active redundancies. At the same time, it is the most powerful technique to provide fault-tolerance. Hybrid technique is very widely employed in designing the fault-tolerant embedded memories.

There are several types of hybrid redundancies that can be used to protect embedded memories. The one-dimensional redundancy employs only redundant columns or rows, but not both. The two-dimensional redundancy uses both redundant columns and rows. There is also the periphery redundancy to protect embedded memory from so called memory-kill defects. An extra memory controller, or refresh circuitry may be considered as a periphery redundancy.

Depending on the specific goals, such as yield, space availability, and guarding against possible defects, the designer has to choose appropriate type of redundancy.

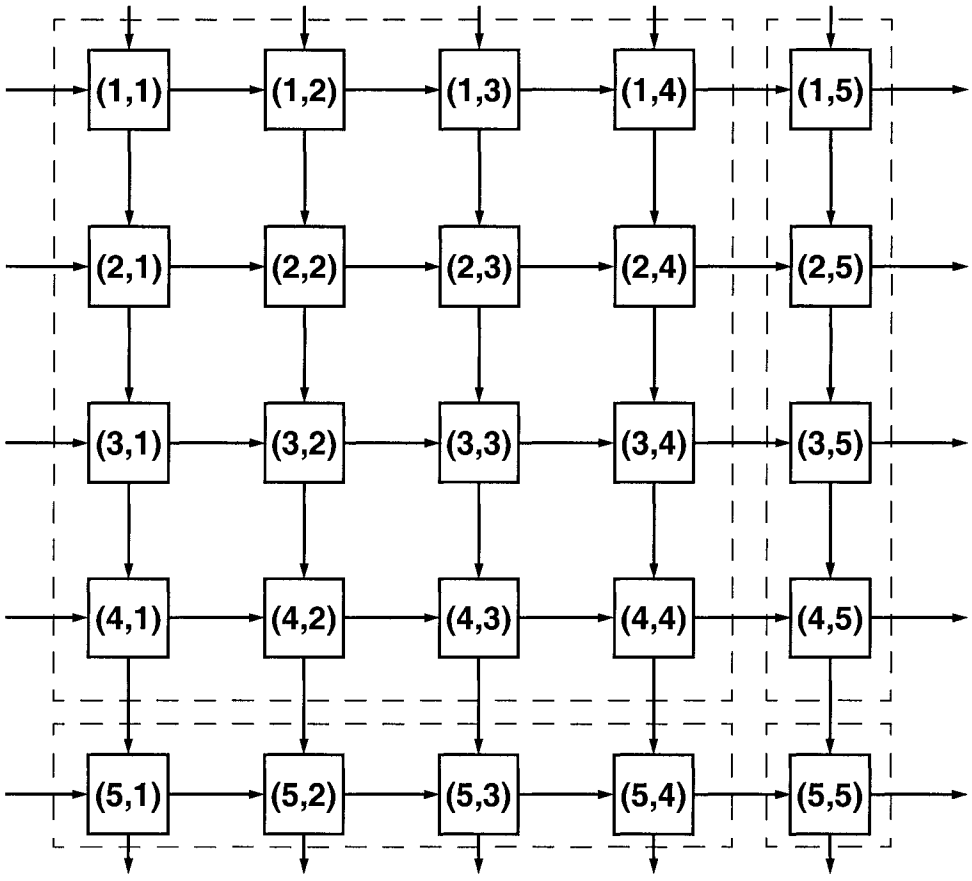


Figure 2.11: Two dimensional redundancy to support 4-by-4 memory core

Normally, the two dimensional redundancy is very popular, because it offers protection for rows and columns at the same time. Figure 2.11 shows a 16-bit memory core free of

any defects and added two dimensional redundancy, consisting of one column and one row.

During normal functioning of the embedded memory all cells are connected directly and no reconfiguration is applied.

One option of repairing the memory core is the *fault stealing replacement* algorithm. The algorithm allows to repair more than one faulty element in the same row or column [9]. The fault stealing algorithm is explained in Figure 2.11 and Figure 2.12.

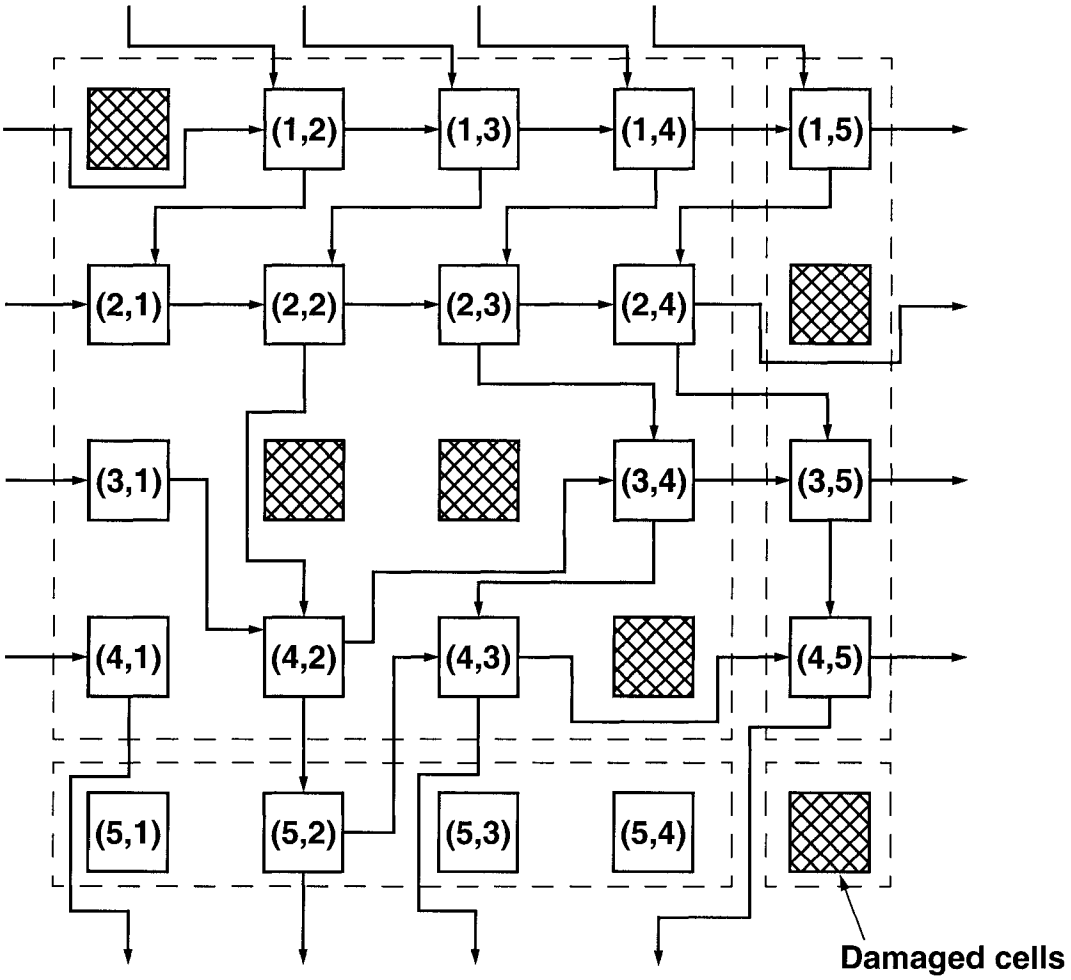


Figure 2.12: Fault-stealing algorithm to repair 4-by-4 memory core

Assume that C is the total number of columns or number of cells in a row and R is the total number of rows or number of cells in a column. For Figure 2.12 $C=4$ and $R=4$. The algorithm starts at cell (1, 1) and gradually checks all the cells in the core from top to bottom.

In case if there are no any faulty elements in the core it returns noreconfiguration signal and the cells are connected the way they are connected in Figure 2.11.

If there is one faulty element in a row, the algorithm replaces it with the cell that is located to the right from the damaged element. Otherwise saying, cell *(m, n)* will be replaced with the cell *(r, c+1)*, cell *(r, c+1)* will be replaced with the cell *(r, c+2)* and so on until the last *(r, R+1)* spare element. Consider Figure 2.12, the element (1, 1) is damaged and must be replaced with cell (1, 2). The cell (1, 2), in turn, must be replaced with the cell (1, 3). And so on until the last cell (1, 4) is replaced with the element (1, 5).

Consider the case when there are two damaged cells in the same row *(r, c) and (r, c+1)*. The rightmost cell *(r, c+1)* will be replaced as was explained in the previous case, with the element which is located to the right from the damaged cell *(r, c+2)*. And the leftmost element *(r, c)* will be replaced with the cell "stolen" from the row which is below the damaged one *(r+1, c)*. The cell (3, 3) in Figure 2.12 is replaced with the cell (3, 4) and the cell (3, 2) is replaced with the cell (4, 2).

Unfortunately, this algorithm fails to work if the element that has to be "stolen" is faulty itself. For example, one of the fail-to-work scenarios is when row r contains faulty elements in columns c and $c+1$. And row $r+1$ contains faulty element in column c. For this case the cell *(r, c+1)* must be replaced with the cell *(r, c+2)* and the cell *(r, c)* must be replaced with the cell *(r+1, c)*. But the cell *(r+1, c)* is faulty itself and the replacement strategy does not work already.

Another algorithm which is very popular and widely applied in industry is *Repair-Most Replacement Strategy*. The main idea behind this algorithm is that it looks for the most damaged rows or columns and replaces them in a very efficient way. The difference between *Repair-Most Replacement Strategy* and *Fault-Stealing Replacement algorithm* is

that repair-most strategy operates and replaces whole row (column) and fault-stealing algorithm operates and replaces individual cells. The repair-most algorithm uses the minimum number of redundant rows and columns.

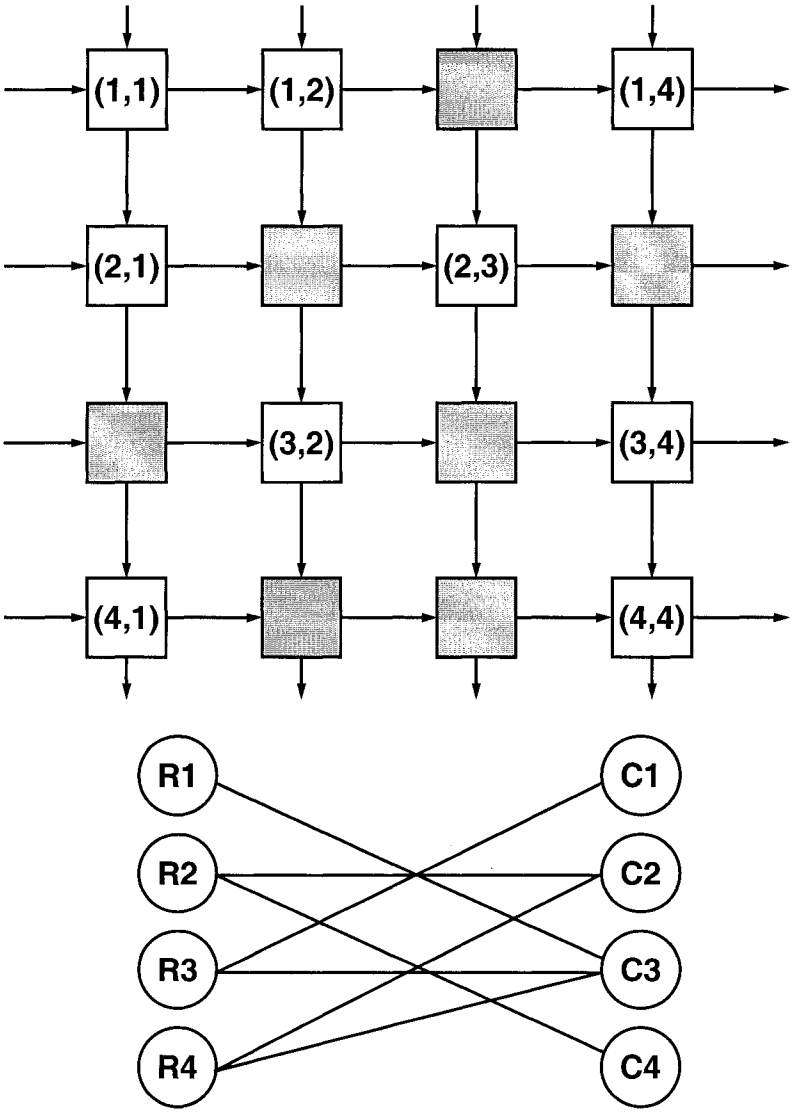


Figure 2.13: Memory core 4-by-4 with damaged cells and its bipartite graph

The principle of repair-most strategy is explained in Figure 2.13. Before replacing faulty cells, algorithm creates a bipartite graph representing the incidence of faulty cells. The number of rows is placed on the left part of the graph and the number of columns is placed

on the right part of the graph. The algorithm connects all the faulty cell row locations with the faulty cell column locations.

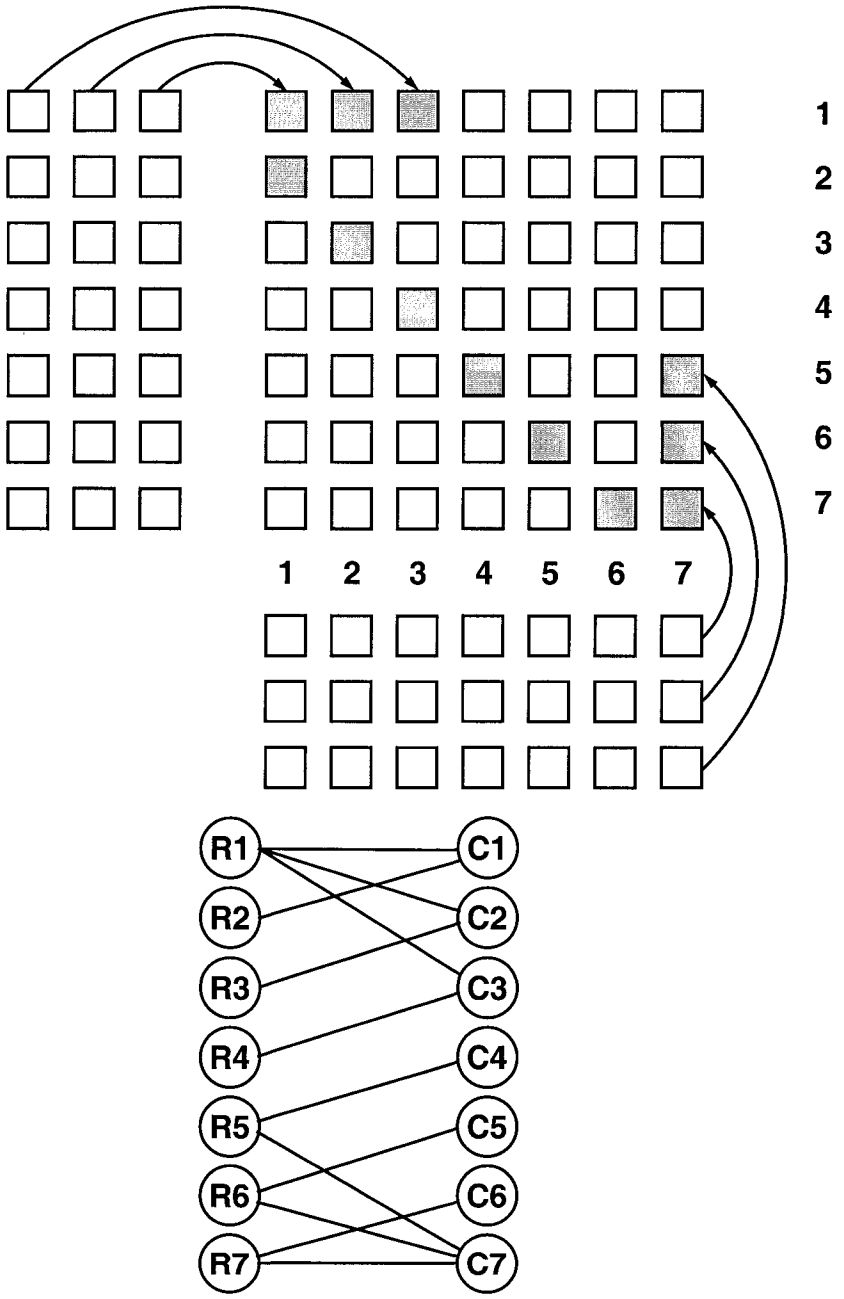


Figure 2.14: Real solution and faulty bipartite graph for damaged memory core 7-by-7

After fully completing the graph, it looks for the node with the biggest number of connections to it. For example in Figure 2.13 node C3 has the biggest number of connections and therefore column 3 contains the biggest number of faulty elements and has to be replaced at first place. When the column 3 is eliminated, the algorithm searches for next node with the biggest number of connections and so on, until all faulty nodes are eliminated completely.

Although the repair-most strategy is very straightforward, it also has some disadvantages. The simple scenario when the repair-most algorithm does not work is shown in Figure 2.14. There are only 3 redundant rows and 3 redundant columns available to protect memory. According to created bipartite graph, row 1 and column 7 have the largest number of faulty elements, and they have to be replaced first. The problem is that it is impossible to cover the rest of the damaged core with the remaining redundancy, and at least two cells will remain uncovered. Meanwhile, the solution exists and is shown graphically in Figure 2.14. Replacing rows 5, 6, 7 and columns 1, 2, 3 it is possible to cover all damaged cells and obtain fully repaired memory core.

Unfortunately, there is no perfect strategy or algorithm to repair all situations that may occur in an embedded memory core. Any algorithm has its weak points where it fails to repair the damage efficiently. The problem of allocating the redundancy for efficient repair of the embedded memory core is known to be *NP*-complete and remains open to further investigations of embedded memory designers.

After performing all necessary calculations, the actual reparation is done by disconnecting bad links using fuses. Normally, fuses are reconfigured with laser techniques.

2.6.2 - Error Correcting Codes Application for Embedded Memories

Originally, the principles of coding theory were discovered by communication theorists. However, these principles are so fundamental that they can be applied to many other domains that need reliable data retrieving and receiving. The embedded memories are also considered to be the systems for receiving, storing and retrieving data. There are many

types of error-correcting codes known. The reason for using error-correcting codes is to achieve good reliability of the protected system. The embedded memory reliability problem differs a lot from the reliable communication problem. Because of this, the embedded memory designer operates only with restricted number of error-correcting codes that are more or less suitable for memory protection.

Why are the error-correcting codes so important for embedded memories? According to statistical data, the majority of errors that happens in embedded memories are single bit errors (Figure 2.15). For example, according to diagram in Figure 2.15, single bit errors occur three times more often than column failure and four times more often than row failure. Instead of wasting the whole redundant row or column, to correct one bit, it is possible to use error-correcting codes.

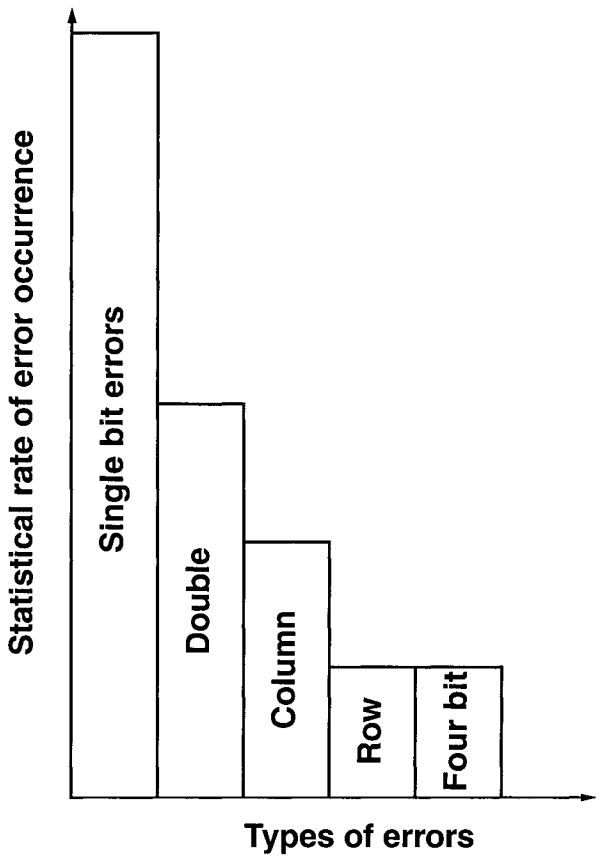


Figure 2.15: Statistical error rate vs. types of errors for embedded memories

Another advantage of using error-correcting code is that the correction of the error can be performed "on the fly", or during actual work, while introducing insignificant delay into the performed operation time. Meanwhile, the redundancy reconfiguration can not be performed "on the fly" and paralyzes the embedded memory for much more time to burn the fuses, in order to apply the redundancy.

Generally, the error-correcting *code* is a method that is able to detect, locate and correct an error inside of the information data word. The information carried by a code is stored in special *code words*. The error-correcting process consists of several steps. The first step is encoding of the information being protected into coding words. The second step storing of the code word into assigned memory location. The third step is decoding information. And the final step is taking a decision according to decoded information. Depending on the code performance, some error-correcting codes are able to correct the affected data and some are just able to detect the presence of the error. The basic term that all error-correcting codes operate with is the *Hamming distance*. Hamming distance is the number of bits that are different between correct data word and affected data word. For example, the given data word is 01011 and the erroneous word is 01110. Comparing these two words one can find out that the difference between them is two bits. Since the difference is two bits, the Hamming distance is equal to two. The minimum Hamming distance is called the *code distance*. The code distance represents the major factor that describes its ability to correct or detect the affected bits. There is a general equation to calculate the number of corrected and detected bits that the error-correcting code can perform. Let c be the number of bits that ECC is able to correct and d be the number of bits that ECC is able to detect. If the Hamming distance H_d is known the relationship between c, d, and H_d will be given by Equation (2.14)

$$2c + d + 1 \leq H_d \tag{2.14}$$

There are plenty of error-correcting codes in theory, but only few of them are used in the embedded memory application. The main codes that are used for embedded memory reliability and yield improvement are described below.

2.6.2.1 Parity codes

Parity code appears to be one of the simplest error-correcting codes and is a basis for deriving many other codes. The basic principle of this type of the code is shown in Figure 2.16. Initially the data word is passed through the *Parity Generator* circuitry and the generated *Check-Bit-To-Store* is stored with the information bits. After storing the information for some time, the data is passed through the *Parity Generator* again and the *New-Check-Bit* is compared with *Check-Bit-To-Store* in *Parity Checker.* Depending on the outcome, the *Parity Checker* either gives *Error Alarm* signal or *No error* signal.

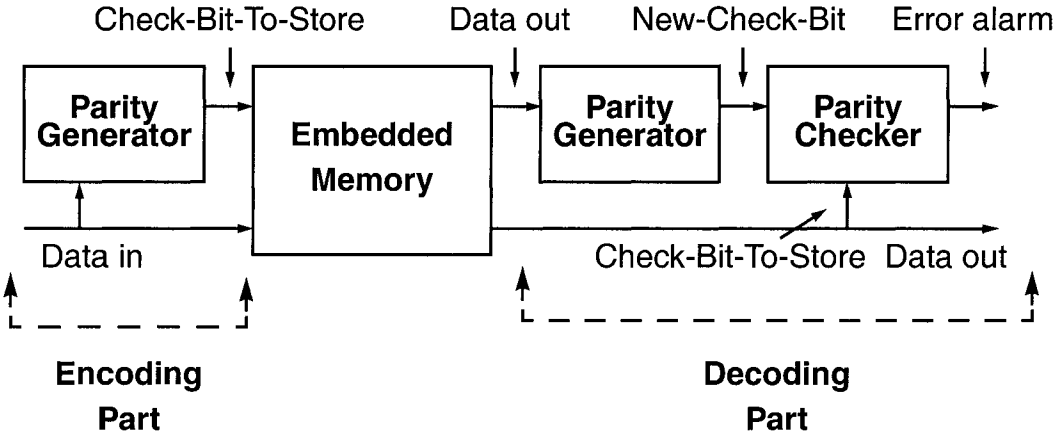


Figure 2.16: Main principle of parity code

Parity generator and Parity checker are usually implemented in the same block, as shown in Figure 2.17. The parity generator is based on XOR-gates.

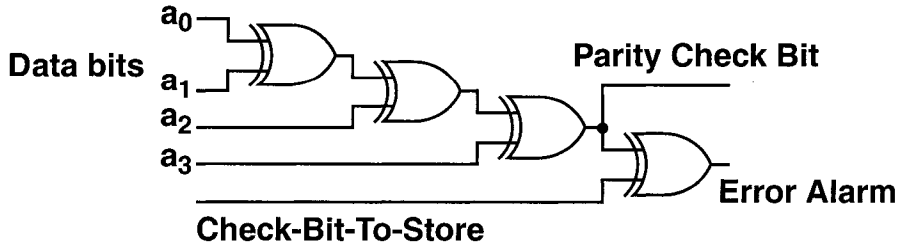


Figure 2.17: Parity generator and Parity Checker

The data passes through several XOR-gates, and at the end, the generated parity check bit is compared with previously stored check bit. The signal named *Error Alarm* shows the occurrence of error. Depending on the System-On-Chip architecture, the parity codes can be applied in different ways.

The first method to apply parity code is *bit-per-word* parity code. As shown in Figure 2.18(a), the memory data bus is divided into the words and the check bit is applied to every word. The main disadvantage of this method is that it can not detect the errors on adjacent lines.

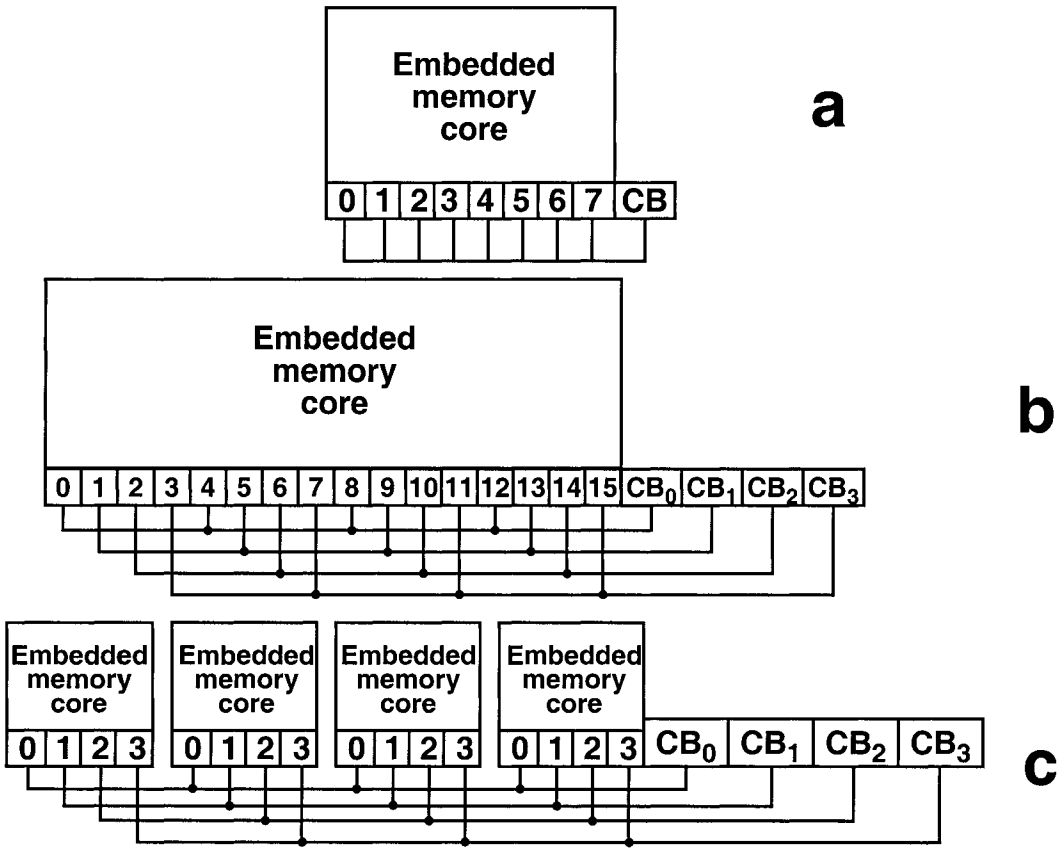


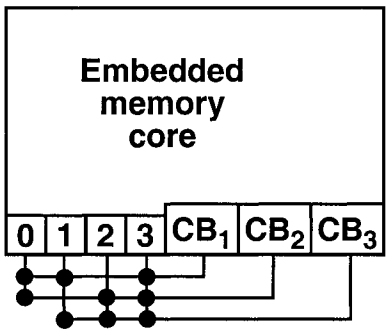
Figure 2.18: Types of parity codes: (a) bit-per-word, (b) interlaced parity, (c) bit-per-multiplememories.

The code architecture that overcomes this problem is the *interlaced parity scheme* shown in Figure 2.18(b). The main memory bus is divided into several groups and one

check bit is attached to each group. In this case, if error happens between two adjacent bit lines, both attached bits will show alarm thus pointing at the location of the error. If System-On-Chip contains several embedded memories the *bit-per-multiple-memories* method will be very helpful. As shown in Figure 2.18(c), one bit from each memory block is passed through parity generator. This method gives very high possibility to detect the error. If an error occurs in one memory block, it will be automatically manifested on the check bit that is responsible for particular memory block.

The most powerful method of all previously described methods is the overlapping parity method. The overlapping parity code is able not only to detect, but also to locate the error in the same codeword. The main principle of this method is that each erroneous bit generates the unique combination of check bits, as shown in Figure 2.19.

For example, if information bit 2 is affected the check bits create unique combination CB_2, CB_3. If information bit 1 is affected the check bits create combination CB_1, CB_3 and so on. The presented scheme can be easily extended to bigger number of information bits.



Erroneous Bit	Unique parity set		
0	CB_1	CB_2	
1	CB_1		CB_3
2		CB_2	CB_3
3	CB_1	CB_2	CB_3
CB_1	CB_1		
CB_2		CB_2	
CB_3			CB_3

Figure 2.19: Principle of overlapping parity method

As usual, there is a trade-off. The overlapping parity method is very effective for error detection and location, but at the same time it is area-hungry.

2.6.2.2 Hamming error-correcting code

Hamming ECC is probably the most popular code in the memory industry. It is very efficient, as it injects small time delays into circuitry, and requires small amount of redundancy area (10%-40%) [9]. Since the Hamming ECC was derived on the basis of the overlapping parity method, the basic structure of Hamming ECC is similar to this method. Assume that there are n information bits in the code word and we need cb check bits or redundant bits in order to detect and correct the error. The amount of redundancy needed for Hamming ECC can be found by

$$2^n \geq n + cb + 1 \tag{2.15}$$

Hamming correction code is described more precisely in Chapter 4 of this thesis.

There are also other error-correcting codes that can be used for embedded memory yield and reliability improvement. Berger, Reed-Solomon and bidirectional codes are considered to be very effective for specific needs.

2.7 - Conclusion

Choosing the appropriate protection method is an important issue, which affects not only yield and reliability of the system, but also its speed, power consumption and the area of the whole System-on-Chip. A designer can protect his embedded memory with either redundancy, ECC or synergistically, using both redundancy and ECC for the same embedded memory. It is proven [10] that using synergistic approach works much better than just using redundancy or ECC only. The embedded memory designer has to take in account a lot of factors that may affect the protection circuitry for embedded memory. Such factors are memory size, occupied area, proximity to noise sources, number of sensitive elements, maturity of fabrication process, power consumption, available space and so on. The number of all the factors are very large to present enough research material for another thesis.

Chapter 3 - MCSoC implementation

This chapter describes the design and implementation of the Managed Clock System-on-a-Chip (MCSoC) project. The primary objective of my part of the project is to design efficient embedded memory, and practically explore yield improvement and fault tolerance of the embedded memories. The second objective is to test novel software programmable clock management circuits, and substrate coupling exploration. As a part of the project, three ICs and one printed circuit board were designed towards testing and debugging of the MCSoC.

3.1 - Introduction

The architecture of the Managed Clock System-on-Chip (MCSoC) project is shown in Figure 3.1. The primary object of the MCSoC is a practical learning of embedded memories function, clock management schemes, processors, substrate noise modeling.

From a research standpoint, the practical design of the embedded memory will enable students to further investigations in this field. During the design, new practical knowledge was obtained. All data that was discovered after design implementation was successfully used in research outlined in Chapter 4 and Chapter 5. The collaboration of a team of people served also for better understanding of planning and managing the practical implementation, and gave students the possibility to experience the real implementation of their ideas.

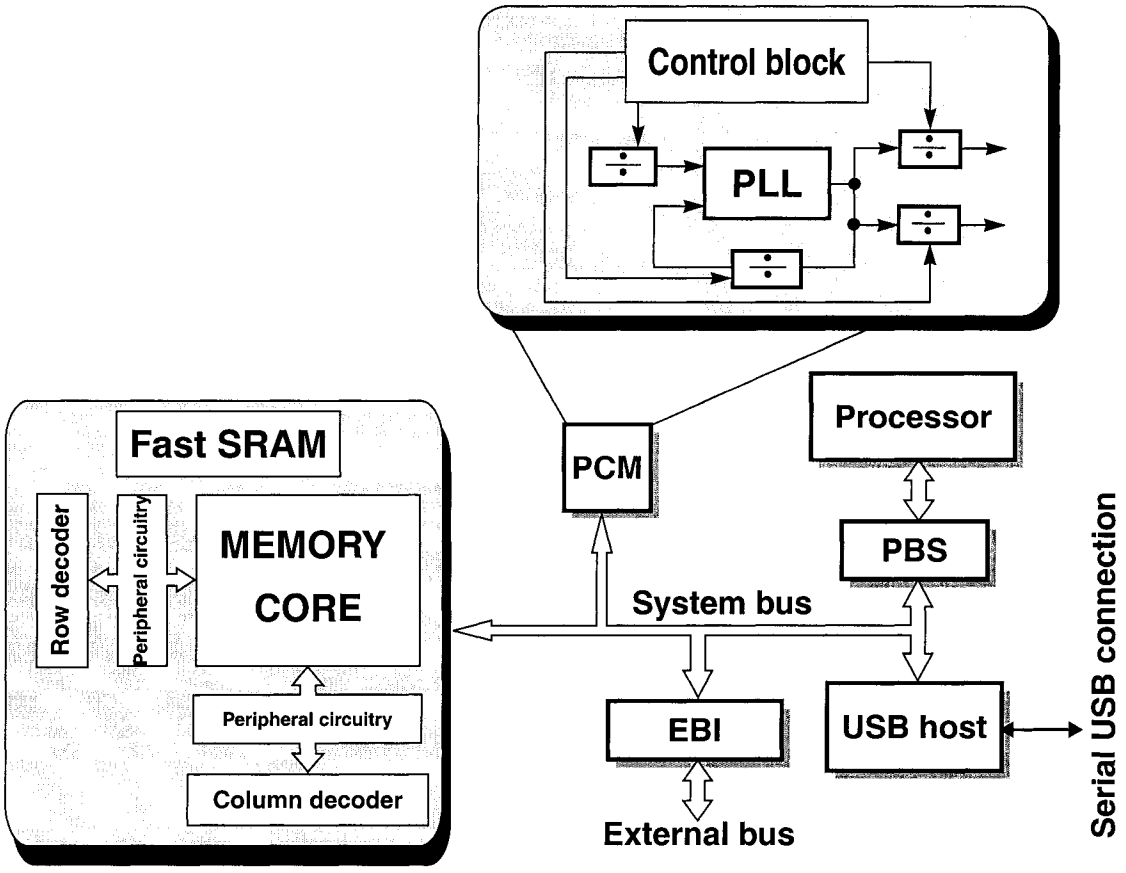


Figure 3.1: System Level Overview of MCSoC

The final MCSoC chip was fabricated in TSMC's 3.3V 0.35mm CMOS process. The design occupies 4mm by 4mm, including the I/O pads and buffers. The layout organization is shown in the die photo, Figure 3.10.

3.2 - MCSoC Organization

MCSoC is comprised of five main subsystems, interconnected by an asynchronous bus, as in Figure 3.1. The design occupies 4mm by 4mm, including the I/O pads and buffers. Custom fast embedded SRAM is implemented in order to understand memory operation and explore yield, reliability and fault-tolerance. Also included is an integrated processor

that interfaces with the bus through the Processor Bus Controller (PBC), a Universal Serial Bus (USB) host controller and a Programmable Clock Manager (PCM).

The processor is based on the Programmable Integrated Controller (PIC) from Microchip Technology Inc. [11]. The PIC is a generic processor with some custom adaptations. The design is compact due to the limited area of allotted silicon. The PBC was designed as a separate component from the processor to enable independent control, thus increasing its versatility.

For the purposes of external communication, a USB host controller has been incorporated into the SoC. The USB protocol was chosen due to its multiple operating speeds. Our intention is to allow MCSoC to act as a low power device, possibly attached to the USB bus.

The design and implementation of the MCSoC were a team effort of five people. The people who participated in MCSoC design are named in Table 3.

Task	Participant/Designer
Supervisor	Prof. Zeljko Zilic
Adaptation and High Level Architectural Design of MCSoC Processor, MCSoC Testing	Ian Brynjolfson and Yanai Danan
MCSoC Processor Implementation, VXI Code Implementation	Yanai Danan
PCM Design, USB Host Design, Test Plan, PCB Design	Ian Brynjolfson
Embedded SRAM Design, Control Block for PCM Design	Boris Polianskikh
Noise Modeling and Analysis	Henry H. Y. Chan

Table 3: Contributions to MCSoC

3.3 - Memory Control Block for Programmable Clock Manager

Although the control block for PCM was implemented in all three pilot chips, in this chapter we explain only the most improved implementation. It should be noticed that even after each implementation the core and reset circuitry were slightly changed, in order to

improve overall performance, the main idea and basic operation always remained the same.

The design explained in this chapter is a result of the experience accumulated during two previous design implementations. The first two pilot chips were designed in 0.35 μm technology and the last one in 0.18 μm technology. The control block is implemented as a matrix of size 4 by 12. As seen from Figure 3.2 each cell of the register was designed on the basis of 6 transistor SRAM cell.

The cell was modified according to the custom needs of the chip. The output transistor of the cell was removed and additional reset transistor was introduced. Each cell can be reset to "0" value by placing "1" on the reset transistor. There is an inverter at the output of the cell, which is placed for two reasons. One is to get non-inverted value at the output, since using only five transistors cell would always give inverted value. The second reason is to amplify the signal, in order to charge the load and get a stable signal at the output of the cell.

3.3.1 - Custom Reset for Control Block

For the proper operation of the whole Programmable Clock Manager, all cells in the control block have to be reset to "0" value upon power-up of the whole circuitry. Then, the reset signal has to be turned-off, and all cells have to be ready for normal operation again. Several designs from IBM patent library were considered for implementing the reset circuitry [12], [13], [14]. The main idea behind all those designs is that, upon powering-up the circuitry, the capacitor starts charging, thus giving time to reset all the cells in the control block. As soon as capacitor gets charged, the reset signal becomes "low", thus letting core the cells operate normally.

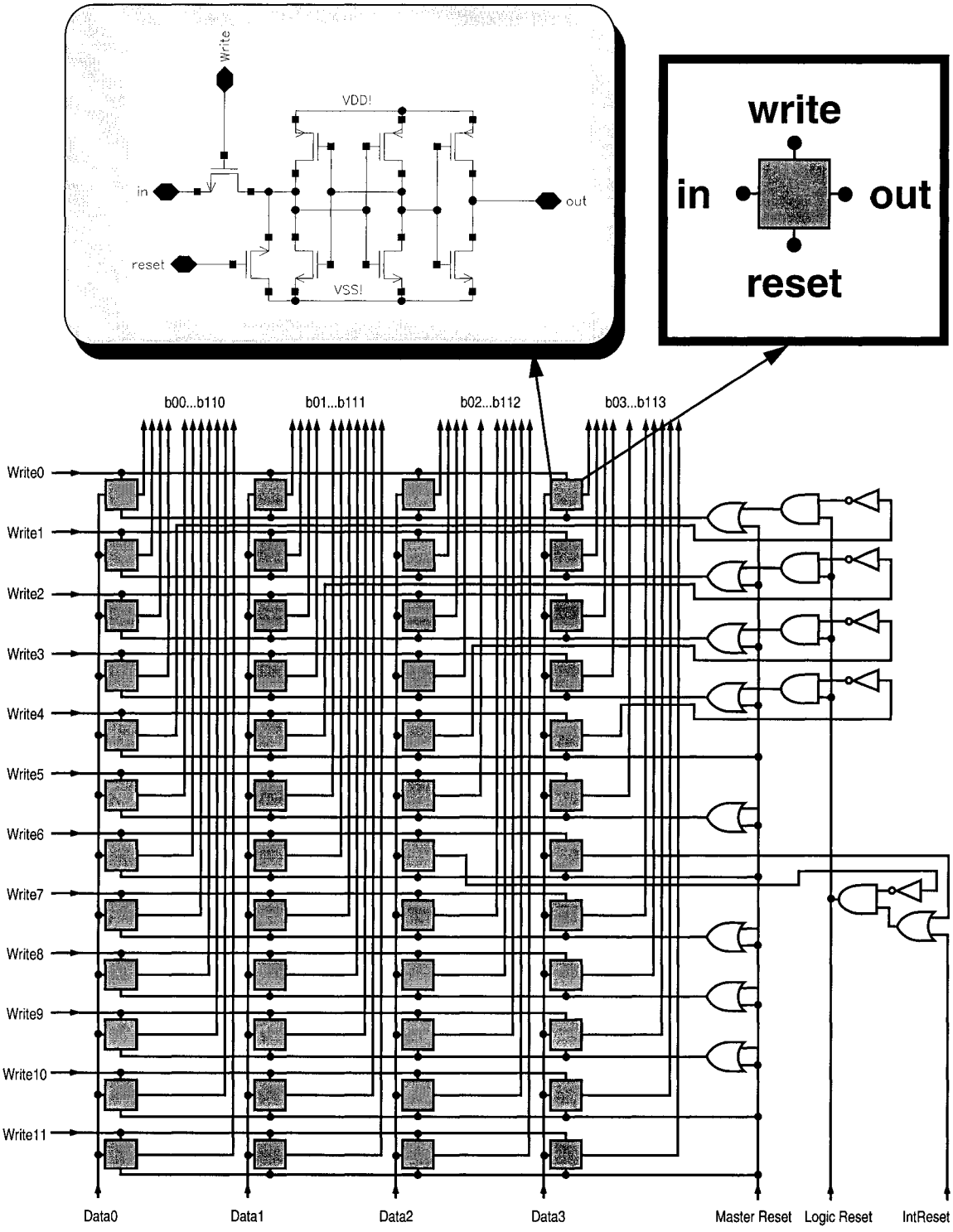


Figure 3.2: Architecture of the control block registers

After analyzing the available space on a die, the IBM designs were rejected, since all of these designs demand a lot of space, mostly occupied by a huge charging capacitor. Another factor which affected the decision of rejecting these designs, is that by introducing them in the circuitry, power consumption increases significantly.

Instead of using start-up circuitry with a capacitor, it was decided to use specially designed reset circuit, which is shown in Figure 3.2. The global reset is implemented with *Master Reset* signal, applied externally from dip switches. The dip switches are located on a *Printed Circuit Board* (PCB), which was designed for control and testing of the MCSoC chip.

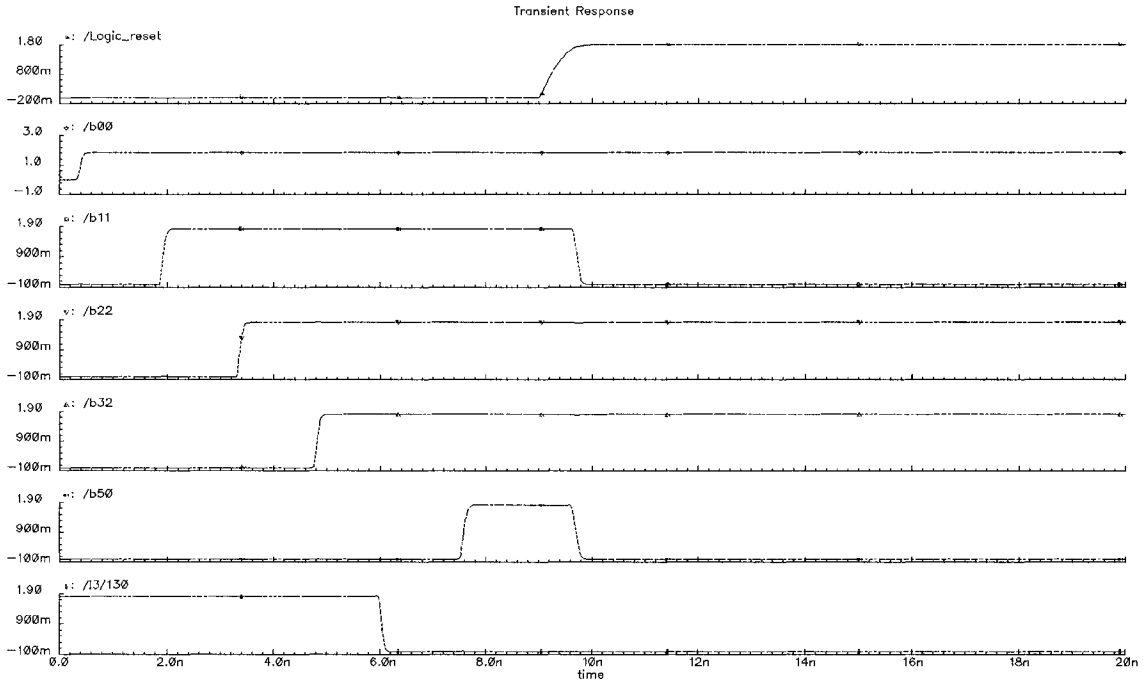


Figure 3.3: Example of the waveform for testing of the reset circuitry

Figure 3.3 shows an example of the testing of the reset circuitry. Comparing Figure 3.2 and Figure 3.3 helps to better understand the given logic reset operation. "High" values were gradually written to cells b00, b11, b22, b32 and b50. As soon as the signal "reset" went up, cell 50 was reset to "low" and since the value "0" was written to cell 41,which is responsible for row 1 reset, the cell b11 was also reset at time 9 ns.

Now control block can be reset in several ways.

• *Reset A*: all 48 cells can be reset by Master reset;

• *Reset B*: internal reset signal and "0" value in reserved cell 62 will enable logic reset signal;

• *Reset C*: cell 62 value is "0" and cell 63 value is "1" will also enable logic reset signal;

• *Reset D*: logic reset signal "1" will reset rows 5, 7, 8, 9;

• *Reset E*: logic reset signal "1" and cells 40, 41, 42, 43 "0" will reset divider control registers rows 0, 1, 2, 3 correspondingly.

Basic operation of the cell is shown in Figure 3.4. All signals of the waveform correspond to signals from Figure 3.2.

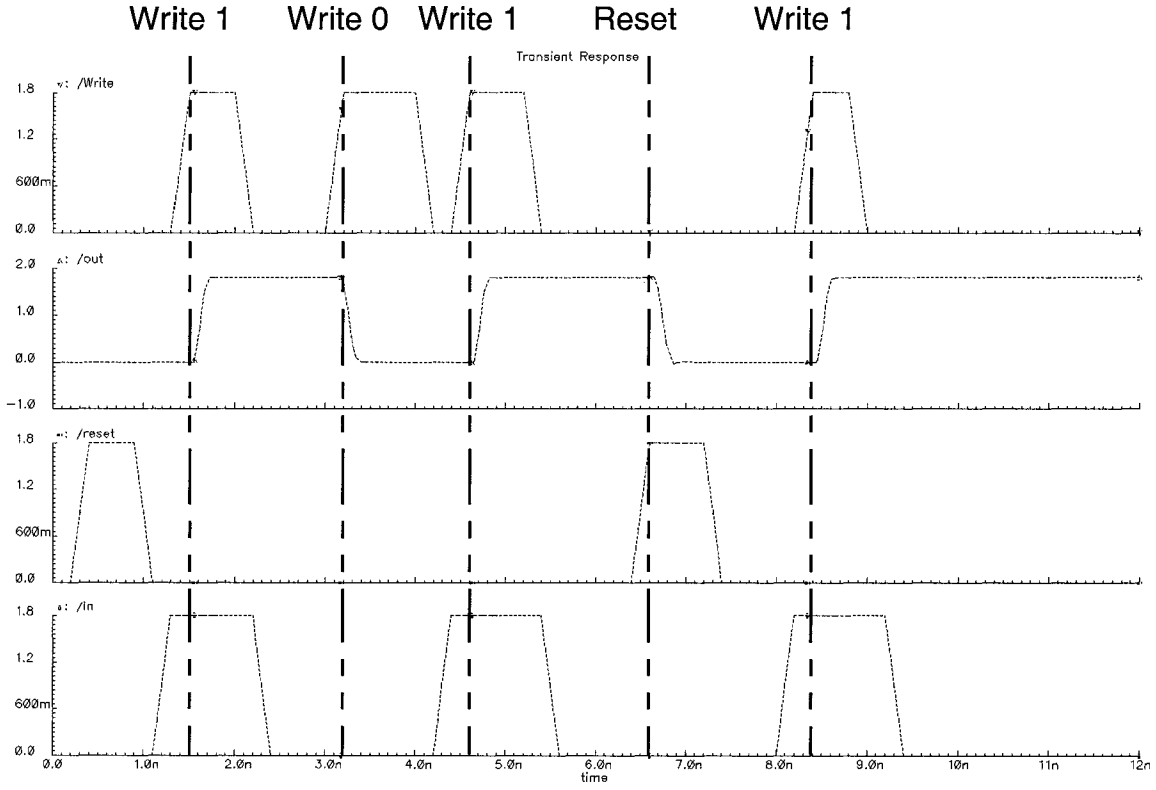


Figure 3.4: Control block cell basic operation

Initially, at time 200 ps, the cell is reset to "0". Next operation is write to "1". Input data has to stable shortly before "write" signal becomes "high". As seen from the waveform, the propagation delay is approximately 422 ps for 30 fF load. Write "0", and "1" operations follow. After this, reset function is checked. For reset signal the propagation delay is 246 ps.

The overall control block design is shown in Figure 3.5. The control block consists of the row decoder and the cells core with reset circuitry. Row address and data inputs are located at the bottom of the picture. Outputs are at the top of the design. The core of registers is directly connected to a decoder.

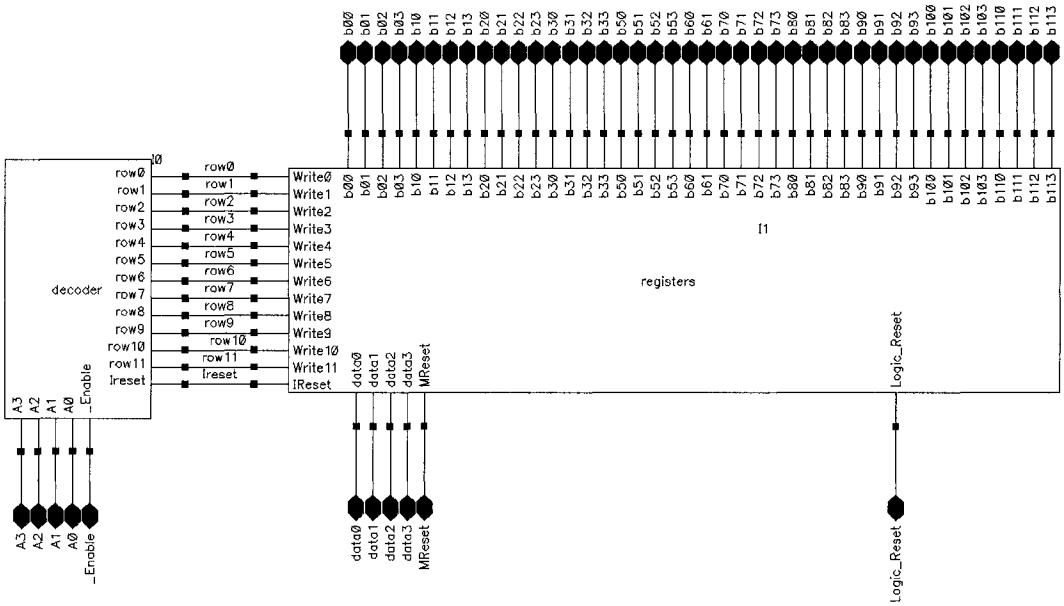


Figure 3.5: Control block for PCM

3.3.2 - Row Decoder

The decoder is shown in Figure 3.6. The row-address decoder is required to select one of the 2^N word lines in response to an N-bit address input [15]. There are 13 rows to decode in our case; consequently, N=4.

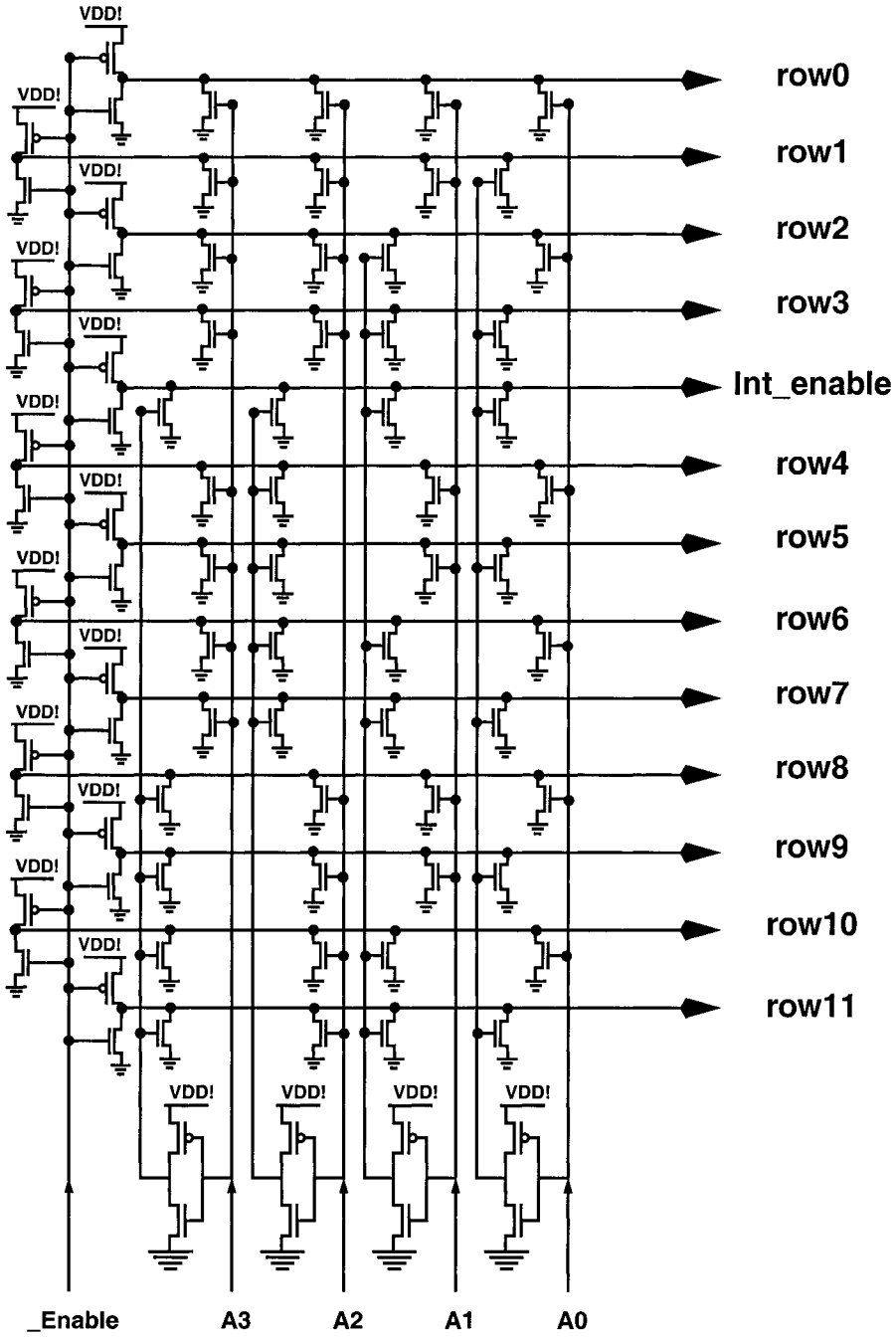


Figure 3.6: Decoder for control block

Address of the accessible row is chosen with address bits A_0 through A_3, where A_0 is least significant bit and A_3 is most significant bit. As shown in Figure 3.7, row 5 will be high,

when A_3=0, A_2=1, A_1=0, A_0=1. In this case all NMOS transistors in row 5 will be switched off, thus, preventing row 5 from discharging, and at least one transistor in all other rows will be "on", thus enabling discharging of these rows. Thus, it is possible to express row-line 5 as a Boolean function of A_3, A_2, A_1, and A_0,

$$Row5 = \bar{A}_3 A_2 \bar{A}_1 A_0 = \overline{A_3 + \bar{A}_2 + A_1 + \bar{A}_0} \quad (3.1)$$

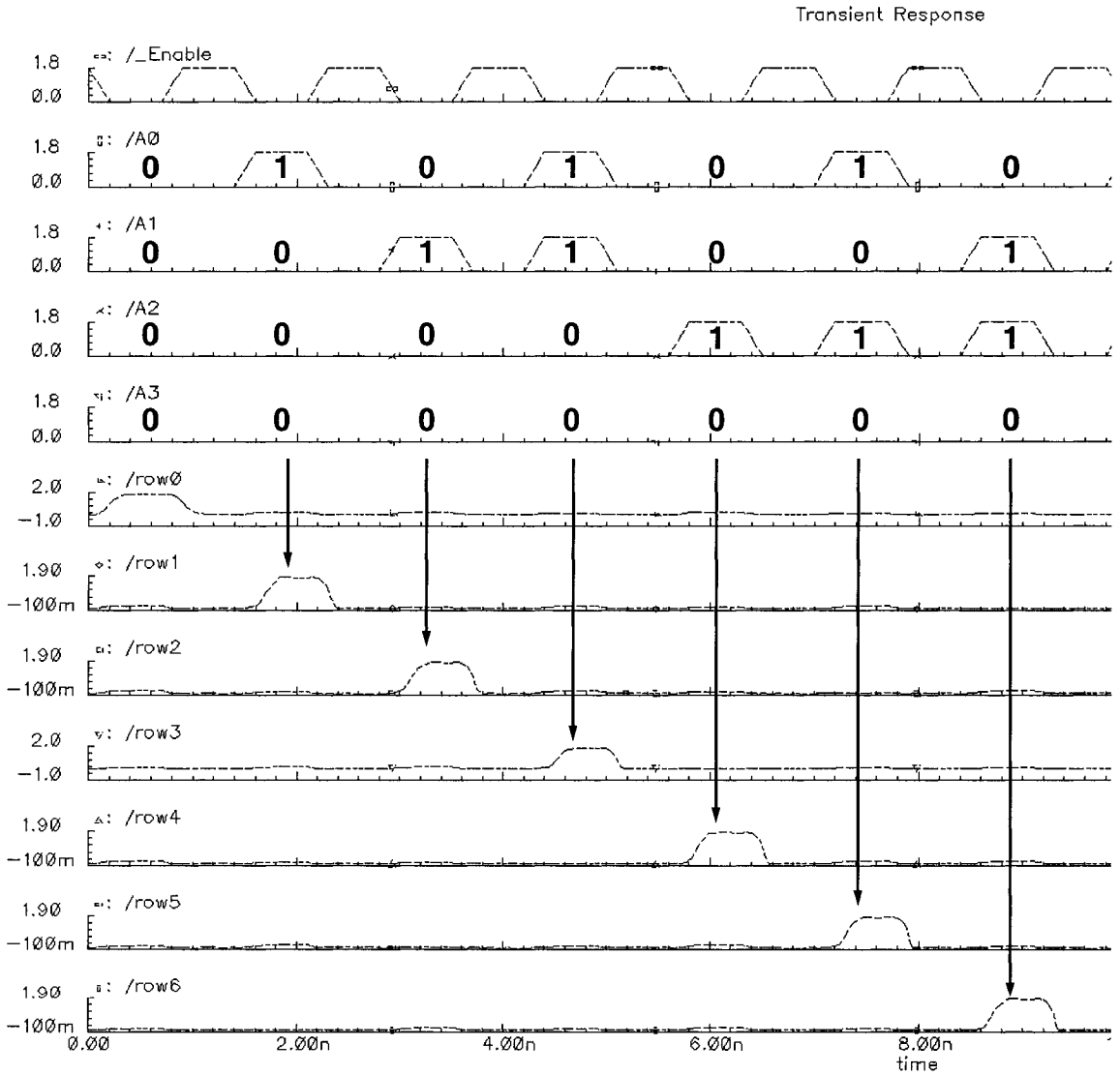


Figure 3.7: Word-line decoder operation

Next, applying Enable_not signal to PMOS transistors in the left part of the decoder (Figure 3.6) will pull row line 5 up. Internal_reset_enable signal, which is in fifth position from the top of the decoder, is also activated by a row line and has its own reserved address 15 or binary A_3=1, A_2=1, A_1=1, A_0=1. Since decoder has to activate only four cells in a row, it is not necessary to use amplifying buffers between the decoder and the registers. The 4-bit data and the address of the row have to be placed on the inputs data [0...3] and A[0...3] (Figure 3.5) at the same time, but prior to activating the decoder Enable_not signal. Meanwhile, these signals are getting in the stable condition, and the Enable_not signal has to be kept "high", in order to prevent accidental faulty writing into the cells. As shown in Figure 3.7, each time Enable_not signal becomes "high" it resets previous value of the row to "0". Whenever data and address signals are stable (approximately 200 ps after putting data on the lines), the decoder Enable_not signal becomes "low" and activates the decoder. The chosen row line is pulled up, and the cells, connected to that line, are written into. Approximately after 600 ps, written value propagates to the output and activates the load (approximately 30 fF). As soon as the data propagates to the output of the cell, the decoder Enable_not signal becomes "high" and resets all row lines to "zero". Thus, control data is trapped into the cells. The control operation starts.

The layout of the control block is shown in Figure 3.8. All inputs and outputs are located at the top of the design, which allows it to be easily accessible from controlled blocks. Changing technology for the same design means not only scaling the design according to new technology, but also new effects to be encountered for. For example, designing control block in 0.35 μm technology we did not have to take care of antenna effect (excessive charge accumulated through area occupied by a wire). For 0.18 μm technology it is already necessary to protect transistor gates from destroying. In order to avoid this effect we used n^+ diffusion connected to the wire near the gate of the transistor. All separate blocks of the control block are shown with lines on the layout picture.

All previously shown diagrams were simulated in Cadence environment after implementing the layout and extracting it with parasitic capacitances.

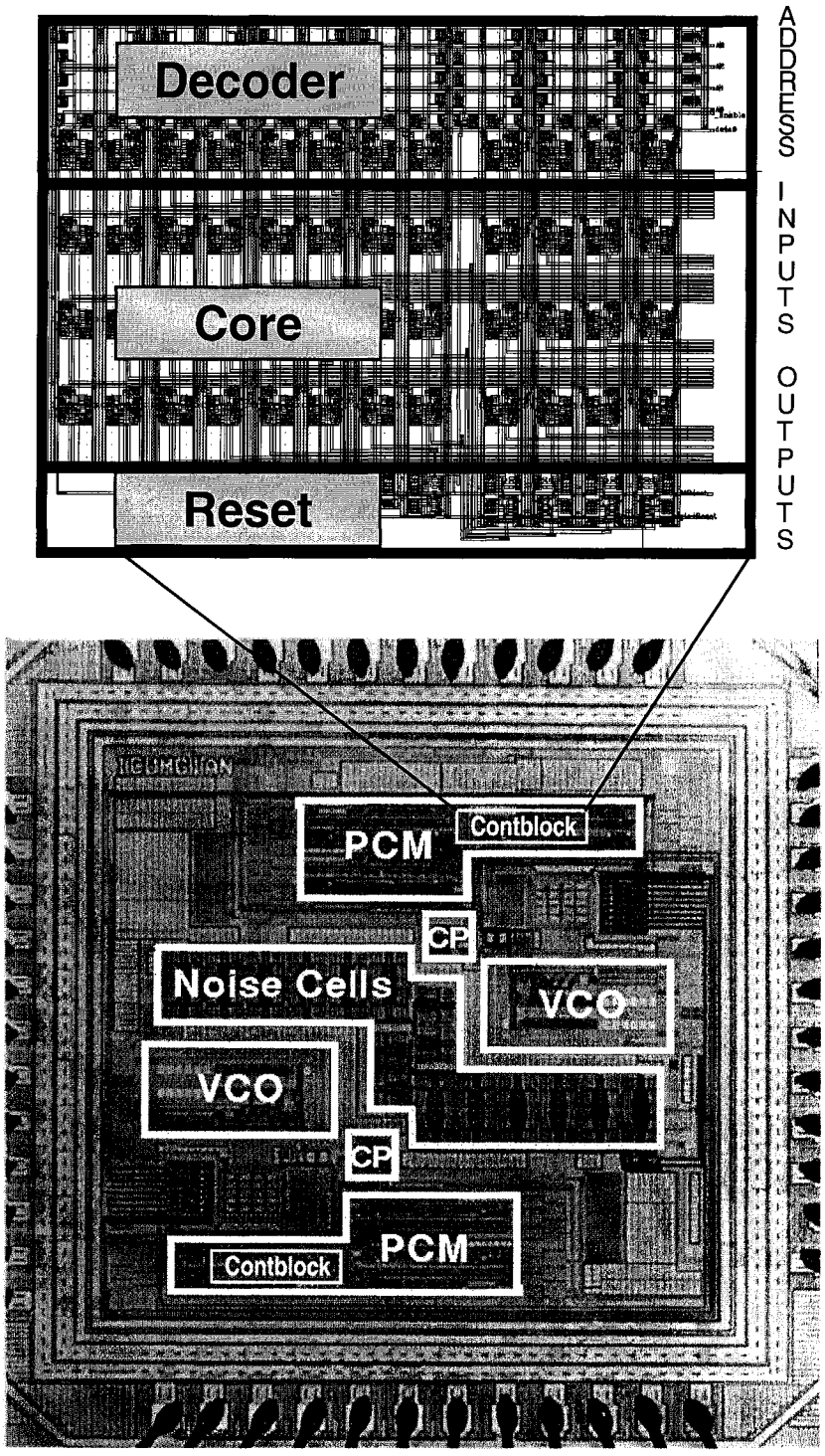


Figure 3.8: First pilot chip with the layout of the control block

3.4 - Designing Static Random Access Memory for MCSoC chip

Initially, it was planified to design 8kBits Multi-Level DRAM for the second pilot chip. Due to the fact that CMC (Canadian Microelectronics Corporation) did not support technology for MLDRAMs at that time (complicated trench capacitor) and due to area restrictions, the initial plans were changed towards simpler design of the fast SRAM.

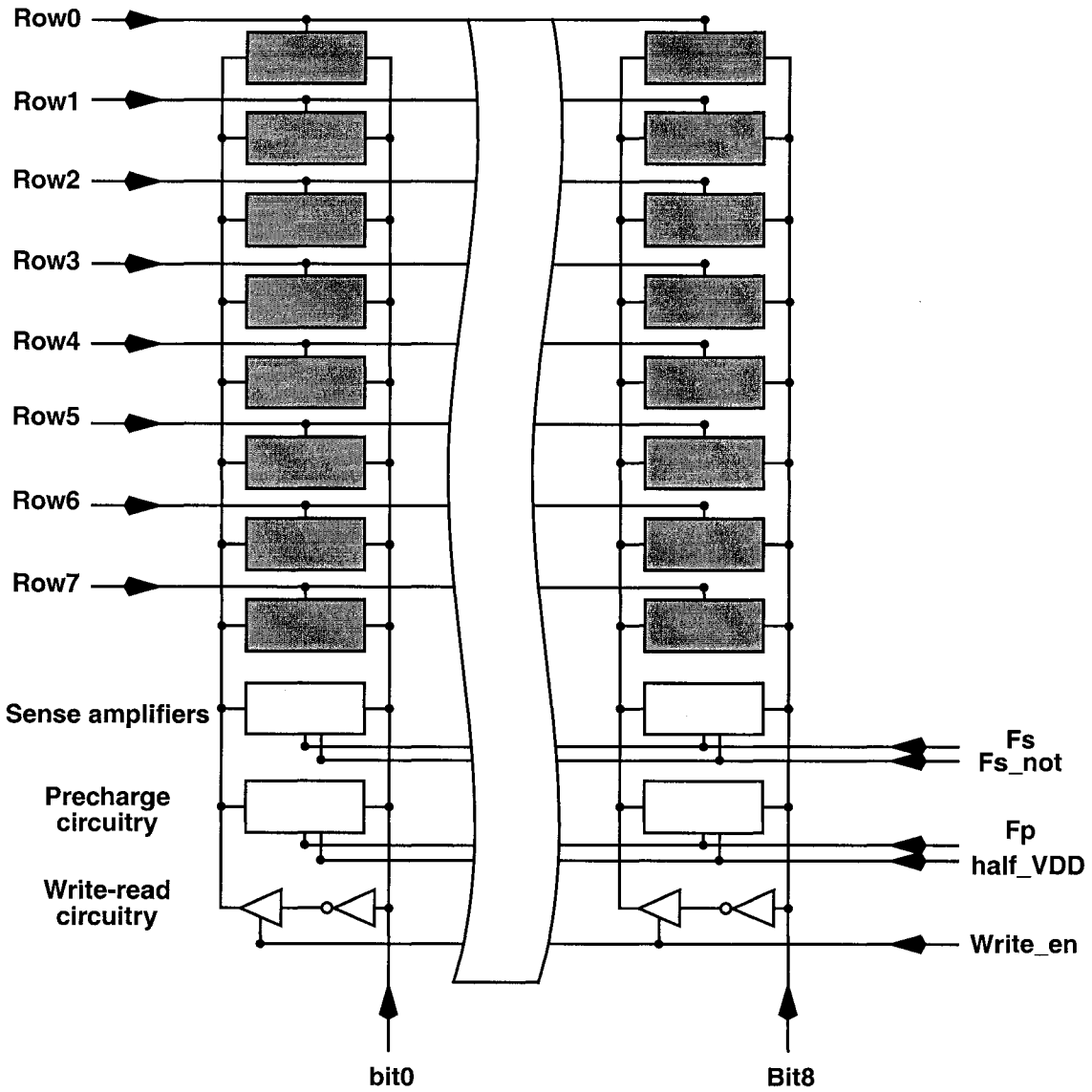


Figure 3.9: Memory sub-block schematic

Fast operation of the SRAM was obtained by adding sense amplifiers and precharge circuitry to the memory design. As shown in Figure 3.9, each block of the memory consists of 8 columns and 8 rows. Each column in turn consists of memory cells, sense amplifiers, precharge curcuitry and write-read circuitry. Write-read circuitry is used to separate write and read operations. For example, if we need to write some value to the cell, we have to put the information bit on the bit-line and the inverted value of the information bit on the bit-not-line. If the ***Write_en*** signal is activated, the tri-state buffer is turned on, and inverted information bit can freely propagate to the bit-not-line. The situation differs for the read operation. When the ***Word*** signal is applied during read operation, the data from the cell has to flow onto the bit-lines. This means that both bit-lines must be in a floating state. Let's assume that there is no tri-state buffer at the bottom of each column. In this case, even if the bit-line is in floating state, the bit-not-line will be either high or low (undefined), depending on the inverter behavior. This will affect the sense amplifier operation, since it has to detect voltage difference between bit and bit-not lines. Eventually, the sense amplifier will give erroneous value if one of the bit lines is in undefined state.

The layout architecture of the memory is shown in Figure 3.10. SRAM is located in the left-bottom corner of the die, and occupies approximately 20% of the whole area of the chip. Fast SRAM memory consists of two parallel blocks. Each block is in turn constructed of two sub-blocks. Each sub-block includes 512 cells (8rows*8 columns*8bits per each column). Each of the four sub-blocks has individual row and column decoders. The decoder for fast SRAM was implemented in the same way as the control block registers for PCM, see Figure 3.6. The only difference is that the number of rows were reduced down to 8 and, consequently, the address for the row became three bits wide.

The column decoder is based on a pass-transistor decoder with NOR pre-decoder and is shown in Figure 3.11. The address is selected with address bits ***B0, B1, B2***, where ***B2*** is the most significant bit and ***B0*** is the least significant bit. As soon as one of the lines of the pre-decoder is activated, the pass transistor is turned on, and data can propagate from or to the memory core. Since it takes too much space to draw 64 transistors in the same picture

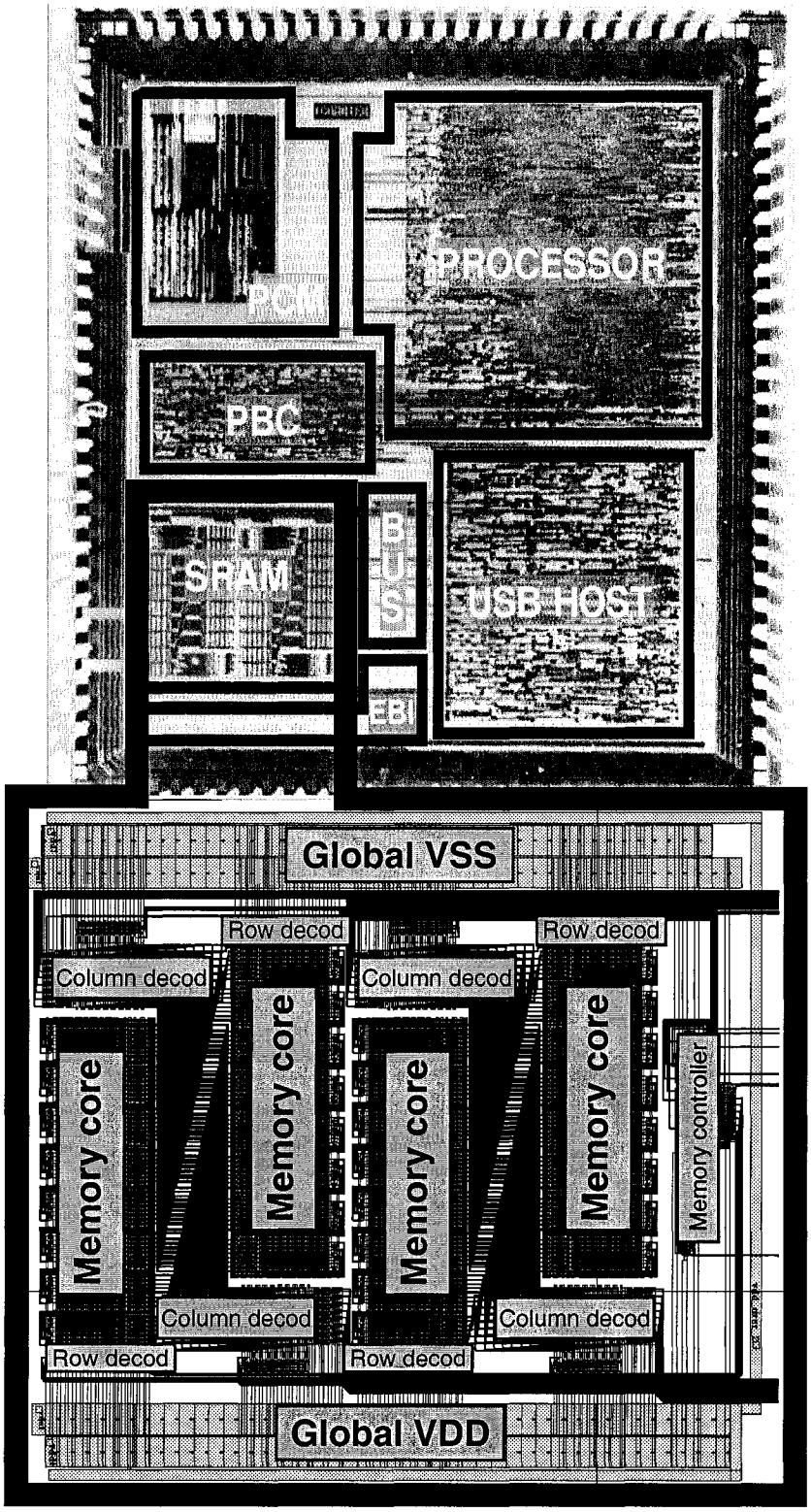


Figure 3.10: The second pilot chip and fast SRAM layout

Figure 3.11 shows column decoder for 1-bit bus only. In the case of 8-bit bus, each line of the pre-decoder is connected to the gates of 8 transistors in parallel. The architecture shown in Figure 3.11 is comparatively fast, although it uses more transistors than other similar architectures. It should be noticed that each time the signal passes through pass transistor, it has to be restored to the original value due to the body effect voltage loss.

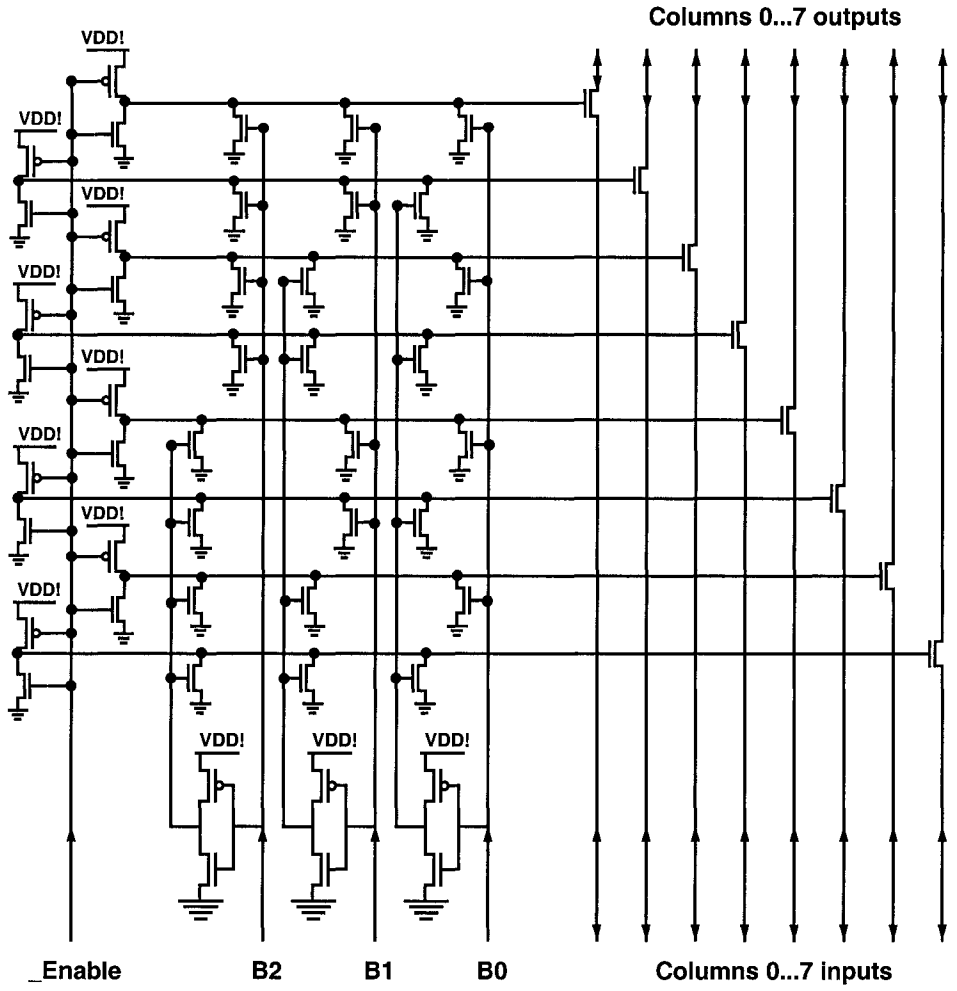


Figure 3.11: Column decoder

The operation of the memory is explained hierarchically in the following subsections. First, the operation of the single cell is explained, followed with the sense amplifier and the precharge circuitry operations.

3.4.1 - Operation of the cell

MCSoC chip memory is designed on the basis of the standard six transistor cell. Figure 3.12 shows typical architecture and operational waveform of the six transistor cell designed in CMOS technology. The cell is comprised of two cross-coupled inverters (transistors ***M0, M2*** and ***M1, M3***) and two access transistors ***M6, M7***. Access transistors connect the flip-flop to the column (***bit-line*** and ***bit-not-line***) lines. In order to connect the flip-flop with the bit-lines, ***Word*** line has to be pulled-up to ***VDD*** and transistors ***M6, M7*** to be turned on. Below we explain how to perform write and read operations on the actual waveform obtained during cell testing in Cadence environment. For better understanding, different operational states of the cell are numbered at the top and bottom parts of the waveform. All the signals are named in the right part of the waveform.

3.4.1.1 Write operation

State 1. ***Enable*** signal (waveform ***C***) becomes high and lets ***Data*** (waveform ***D***) propagate to bit-lines. Since the data bit is "1", ***bit-line*** becomes "high" and ***bit-not-line*** becomes "low" (waveforms ***F*** and ***I***). Now the memory cell is ready for the next operation.

State 2. ***Word*** signal (waveform ***E***) becomes "high", turns on transistors ***M6, M7*** and lets data propagate to the cell. Then ***Word*** signal changes to "low" and captures information bit in the cell. Form state 2 to state 3 the cell is in the storage mode.

3.4.1.2 Read operation

State 3. Read operation starts from precharging both bit-lines to mid-voltage value, for 0.35 μm technology it is 1.65 V. As soon as ***Fp*** signal (waveform ***A***) becomes "high" ***bit-line*** and ***bit-not-line*** are precharged to 1.65 V (waveforms ***F*** and ***I***).

State 4. As soon as bit-lines are fully precharged ***Fp*** signal becomes "low" and at the same time signals ***Fs*** (waveform ***B***) and ***Word*** (waveform ***E***) becomes high. When the word line is selected the current flows from ***VDD*** through ***M1*** and ***M7*** onto ***bit-line***, charging the capacitance of the ***bit-line***. On the other side of the flip-flop the current flows from ***bit-not-line*** to ground through transistors ***M6, M2*** thus discharging ***bit-not-line***.

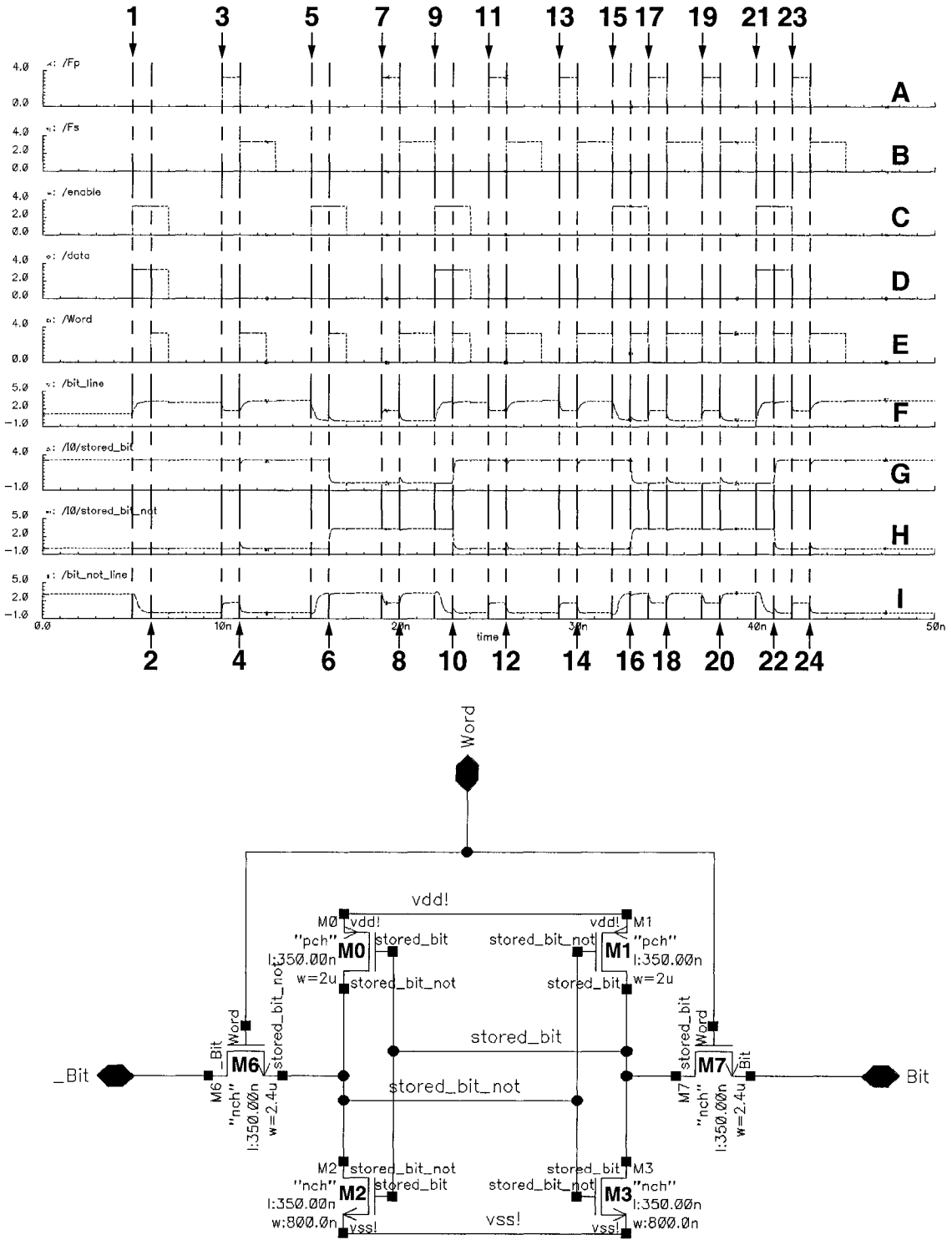


Figure 3.12: Single-cell basic operations waveform and circuit diagram

As soon as sense amplifier is able to see the voltage difference, the bit-lines are precharged or discharge to full voltage swing.

States 5, 6 show write "0" operation. As seen from waveforms ***F*** and ***I*** at state 5, ***bit-line*** gets discharged and ***bit-not-line*** gets charged. At state 6, when ***Word*** line is selected the information propagates from bit-lines to the cell. As seen from the waveforms ***G, H*** the value written to the cell changes to "0".

States 7,8 show read "0" operation. The operation follows similar to read "1". The only difference is that now ***bit-line*** is getting discharged and ***bit-not-line*** is getting charged. When the word line is selected the current flows from ***VDD*** through ***M0*** and ***M6*** onto ***bit-not-line***, charging the capacitance of the ***bit-not-line***. On the other side of the flip-flop, the current flows from ***bit-line*** to ground through transistors ***M7, M3*** thus discharging ***bit-line***.

In order to test maximum performance of the cell, all operations were accelerated. ***States 9,10, 11, 12*** show write 1, read 1 operations. ***States 13,14*** perform read 1 operation again, in order to check non-destructiveness of the read operation. ***States 15*** through ***24*** repeat write 0, read 0, read 0, write 1, read 1 operations with maximum possible speed.

3.4.2 - Sense amplifier operation

Sense amplifier is designed on the basis of a full-complementary positive-feedback voltage sense amplifier and is shown in Figure 3.13. The amplifier consists of two cross-coupled latches and two switch transistors. One latch consists of transistors ***M1, M2*** and another consists of transistors ***M3, M4***. The main purpose of the sense amplifier is to detect small voltage difference between bit-lines ***_B*** and ***B*** and to amplify detected difference to full voltage swing, as shown in Figure 3.13. Two switch transistors serve to connect cross-coupled latches to VDD and ground. The operational waveform of the sense amplifier is also shown in Figure 3.13. Initially **V*(bit-line)*** is precharged to VDD/2. As soon as ***Word*** signal is activated the small difference in a voltage appears on a bit-line. Then sense amplifier is activated with the signal ***Fs.*** Sense amplifier detects the difference **ΔV** and pulls the detected value to a full voltage swing. All these operations can be

compared with Figure 3.12, for example, sense operation can be observed at states 4, 8, 12 and so on.

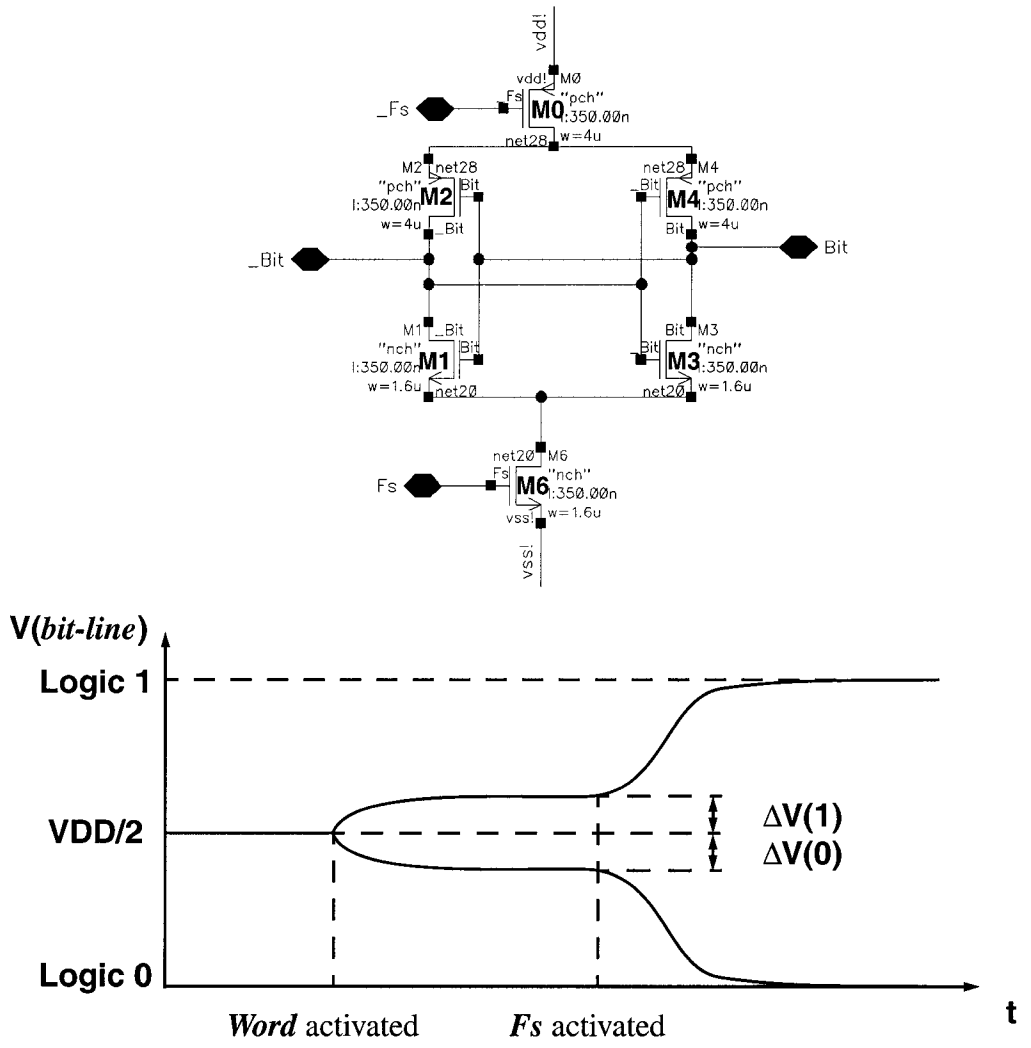


Figure 3.13: Sense amplifier schematic and operational waveform.

3.4.3 - Precharge circuitry operation

Precharge circuitry is responsible for precharging and equalizing bit-lines to mid-voltage. As shown in Figure 3.14, precharge circuit consists of three transistors powerful enough to charge and equalize one column to VDD/2. The operation of the precharge circuit is quiet simple. All three transistors conduct when signal ***Fp*** becomes high. While

transistors ***M1*** and ***M2*** precharge bit-lines to mid voltage, transistor ***M0*** equalizes both bit-lines and at the same time speeds-up precharging operation. All three transistors have to be carefully matched.

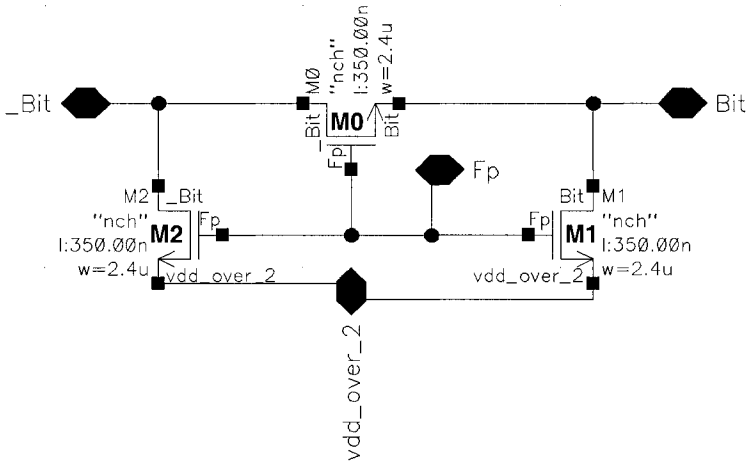


Figure 3.14: Precharge circuitry schematic

If precharge circuit is not matched properly, it will introduce erroneous voltage onto the bit-lines prior to sense operation and erroneous value will affect the sense amplifier function and consecutively the whole memory performance.

3.5 - Testing of the pilot chips

Both pilot chips were tested and verified after actual fabrication. For the purposes of efficiency, a test plan was used to organize efforts, share access to the device and test equipment, and to ensure incremental debugging. The debugging process began with the system bus and PCM, followed by the SRAM, then the processor and PBC. The USB is tested last due to its complex serial communication protocol, and because the EBI can be used for external communication.

The *Printed Circuit Board (PCB)* is shown in Figure 3.15 was created to preserve the integrity of the power, ground, clock and digital signals, and to provide a platform for interconnecting the multiple buffers to the MCSoC IC and test equipment connectors.

Initial tests, using a generic test head and HP16500B logic analyzers, were conducted to ensure functionality before proceeding with the design and construction of the PCM. Many tri-state buffers are needed on the PCB due to the bidirectional connections of the EBI and VXI bus. Buffers are also needed to step down the 5V signal of the VXI bus to the 3.3V operating voltage of the MCSoC IC.

Although it was impossible to test the Control Block Registers and the fast SRAM directly, due to restricted number of pins on the chip and lack of direct access the Control Block Registers and the fast SRAM were tested through observation of the connected to them components. For example, since the Control Block Registers are connected to the clock dividers of the Programmable Clock Manager the only way to test them is to observe the changing of outputs of the PCM with the changing of the inputs of the Control Block Registers. The test preparations are explained in following section.

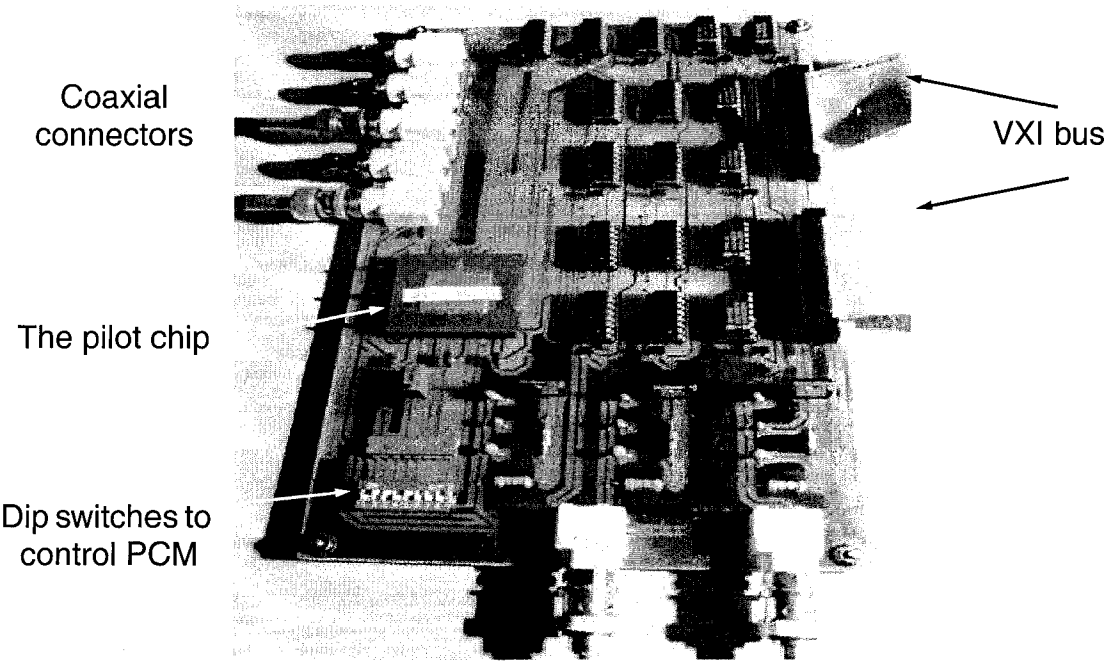


Figure 3.15: PCB Test Board for testing of the pilot chips

3.5.1 - The IEEE 488 and VXI Bus Measurement Setup

The test plan included the development of C code used for controlling the HP1401 VXI bus mainframe, the primary test device for communicating with the MCSoC system bus through the EBI. The VXI bus is capable of generating and reading several digital and analog signals. For the purposes of MCSoC, the VXI bus used 32 bi-directional digital pins and seven digital control pins. Although the VXI bus generators and analyzer modules are limited to an operational speed of 20MHz, testing was possible due to the full handshake protocol of the system bus. Additional analog test equipment is connected through an IEEE 488 bus, and in early stages, the 50 MHz HP16500B logic analyzer was used together with its signal generator modules.

The test process was established in advance, such that all desired options could be built into the C code during its development. This resulted in an efficient hierarchical code structure that is easily maintained and understood by successive designers. The test code was written so that the specific functions for controlling the test equipment were transparent to designers when setting up programs to test their components. Designers need to concern themselves only with high level functional calls for bus operations such as master write, slave read and observation. With the hardware functionality of MCSoC confirmed, a C program is used to automatically load the assembly level code from a text file into the processor by merely running the executable and specifying the file on the command line.

3.6 - Summary

The System-on-chip (SoC) implementation of embedded memory is implemented in the MCSoC project. This platform eases research in the areas of fault-tolerance of embedded memories, yield improvement and reliability of the embedded memories and the confirmation of ideas of organizing systems-on-chip. With embedded SRAM, a simple generic processor, communication systems, and the PCM, MCSoC is has been optimized for functionality from a research standpoint.

Hardware research will continue into successive embedded memory designs methods, yield and reliability improvement. These future developments will be carried out by students furthering the work presented in this thesis.

Memory area 1569.65 μm x 1762.35 μm. Read cycle (RC) fastest time 33 ns, write cycle fastest time (WR) 16 ns. Data has to be stable 1 ns before activating write-enable. Clock speed 500 *MHz*. Dynamic power consumption for max clock speed is 97.62 mW, 1/2 of max clock speed 77.9 mW and for 1/4 of max clock speed is 68.91 mW.

Chapter 4 - Induced Error-Correcting Code for 2-bit-per-cell Multi-Level DRAM

As microelectronics technology moves towards Systems-On-Chip (SOCs), it is often profitable to create a few hardware components, and implement many features in software. In order to facilitate a large software component, dense memories will be needed. Multi-Level DRAMs (MLDRAM) can provide high capacity memories: in 1999, NEC broached the 4-Gbit density level with a file-memory DRAM that uses multiple voltage levels to store 2 bits in each cell [16]. Also, MOSAID technologies implemented several sensing techniques of different voltage levels for MLDRAMs.

High susceptibility to errors due to reduced noise margins prevents MLDRAMs from entering the commercial market. MLDRAMs must be able to tolerate process variations, as well as common memory errors. Error Correcting Code (ECC) is one of the methods to protect MLDRAM. Due to the fact that MLDRAM stores 2 bits/cell, conventional ECC is not able to provide sufficient protection. In this chapter, we propose the induced ECC, that protects MLDRAM in a better way.

As induced ECC implementation is tightly coupled in MLDRAM sensing circuitry, its operation is described first. We then describe the types of faults which are usual for MLDRAMs, and outline details of implementation and trade-offs of designing the induced

ECC.

4.1 - MLDRAM Basic Operations

MLDRAM operation is more complicated than that of the conventional DRAM. The state flow diagram of the cell is shown in Figure 4.1 Values show stored binary value and storage voltages in $1.8V$ technology. Following presented diagram, the read operation first compares stored voltage with reference voltage of *0.9V*. Then, depending on obtained result from sense amplifier, the Most Significant Bit (MSB)is assigned to be one (high) or zero (low), where **X** in the circles stands for unknown value of the Least Significant Bit (LSB). The next operation consists of comparing the stored voltage value with the LSB voltage reference. At the far right side of the diagram, the circuitry reads out the value of the cell.

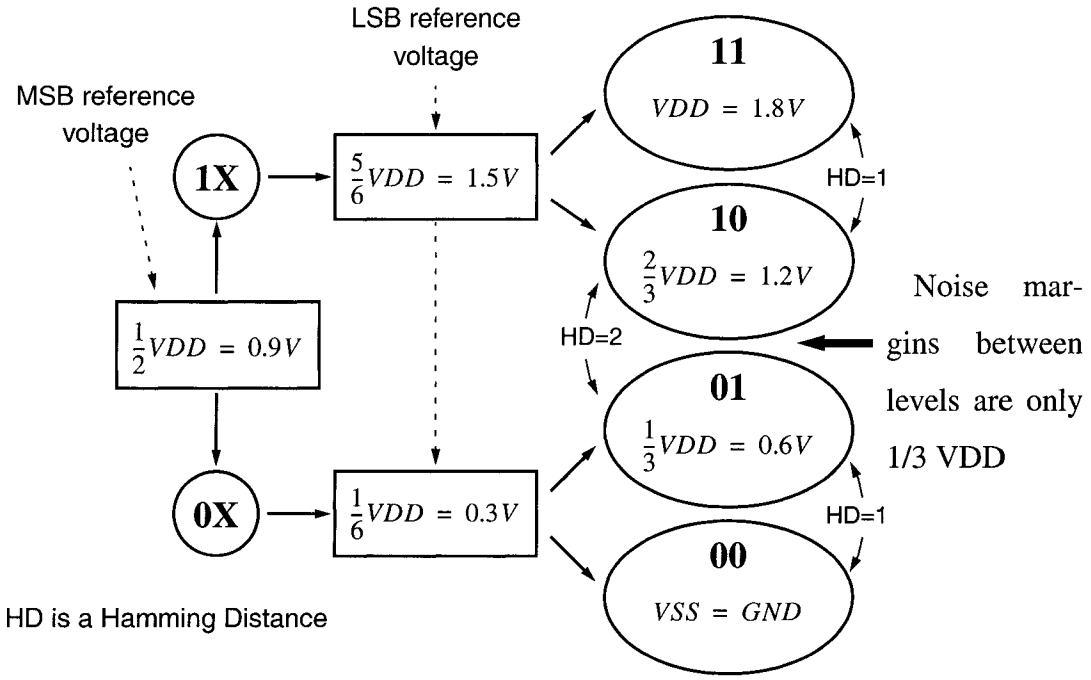


Figure 4.1: MLDRAM basic operation

Figure 4.2 shows the circuit schematic, on the basis of which all basic operations are explained. MSB is always accessed from the bottom side of the circuit and LSB from the top side. There is an interconnect matrix in the middle of the design, which serves for sharing signals between right and left, top and bottom parts of the circuit. The matrix is controlled by signals *X*, *C*, *_X* and *_C*. All operations are shown on a timing diagram in

Figure 4.3. Each point represents the state of the memory at a given time. Operation is performed on a cell *WLi*, which is situated on the left complementary bit line.

The main difference between operations of the MLDRAM and conventional DRAM is that, for read operation, MLDRAM requires different voltage reference generations, depending on the previously read value. All operations are explained next.

4.1.1 - Write operation

Figure 4.3 shows waveform diagram for 0.18 μm technology. During this cycle, a two-bit value is written into the cell. The first operation starts from state *1*.

State 1. Control signals *YSR* and *YSL* become "high" and allow data propagate to bit lines.

State 2. Control signal *IR* become "low" and disconnects right bit line from sense amplifier, leaving LSB captured on complementary right bit line *_BR*. Right after this *_X* becomes "high" and connects *_BL* and *BR*, thus allowing both bit lines to carry MSB. Left sense amplifier is disconnected and signal ER pulled up, thus allowing bit lines *_BR, _BL, BR* to carry total charge.

4.1.2 - Isolate and store operation

During this cycle, the data is isolated from sense amplifiers and put in storage state.

State 3. Control circuit causes word signal *WLi* to go "high" and dummy word signal to go "low". This action captures stored value into the cell.

State 4. All bit lines are precharged to VDD/2, and sense amplifiers are turned off. After this, the memory is in the storing state.

4.1.3 - Read operation

During this operation, memory reads out stored value. Since read operation is self-destructive (like in conventional DRAM), data needs to be restored.

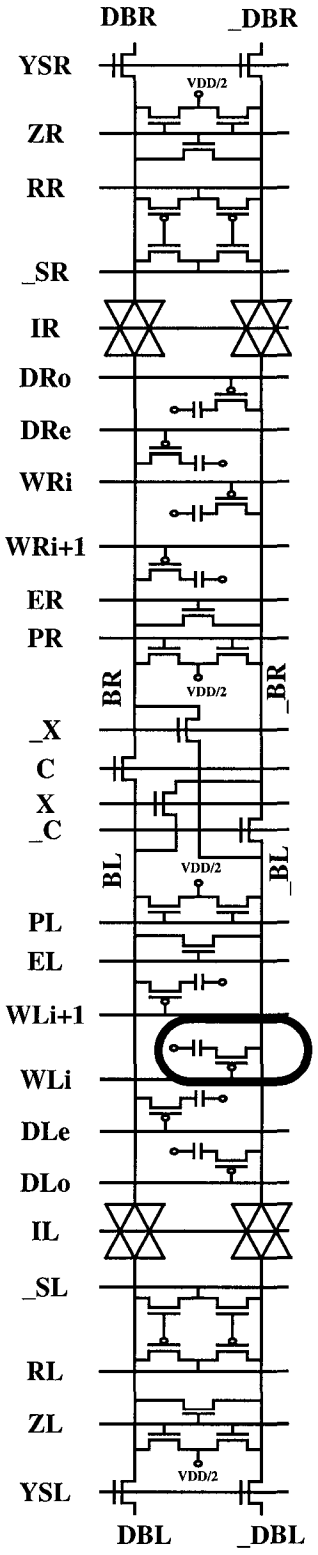


Figure 4.2: MLDRAM access circuit schematic.

At the end, during state 9, the restore operation occurs.

State 5. At first *WLi* is enabled, then *_C* is also enabled. Signal stored in a cell is divided between two bit lines *_BL* and *_BR*.

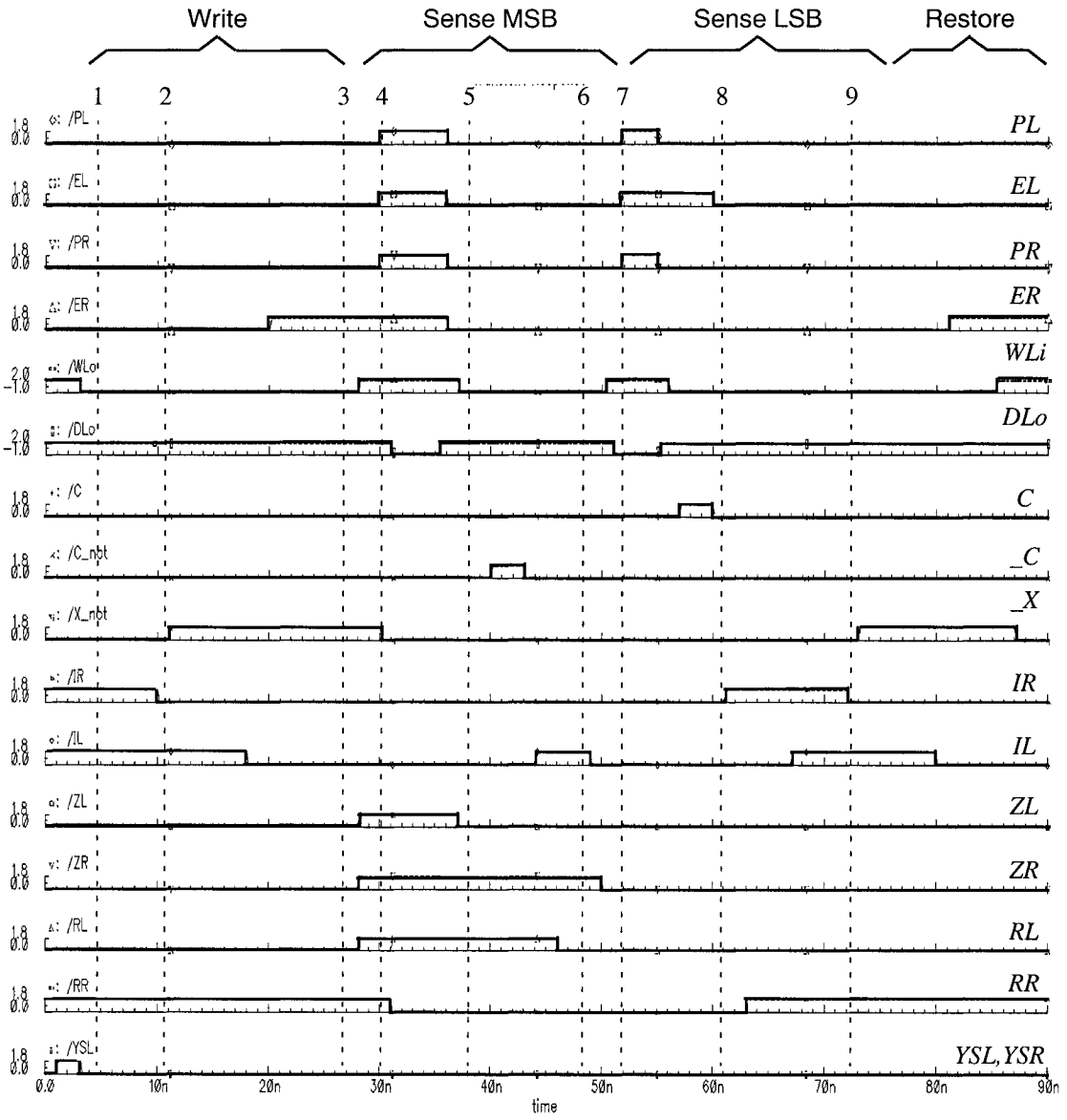


Figure 4.3: MLDRAM basic operations

State 6. Left sense amplifier compares stored value on *_BL* to *BL* precharged to *VDD/2*. Left sense amplifier applies full swing voltage to *BL* and *_BL*. Signal *IL* becomes "low" and latches MSB value on the left sense amplifier.

State 7. Word line *WLi* becomes "high" and captures full MSB value (VSS or VDD) in the cell, thus creating reference voltage for LSB. Then, the left bit lines are precharged to mid-voltage and equalized. The MSB cell signal is then distributed onto bit lines *_BL, BL, BR* by asserting *EL* and *C*. The resulting reference signal on *BR* is then isolated.

State 8. The signal IR is asserted, and right sense amplifier compares LSB stored value on *_BR* with reference voltage on *BR*. Then, obtained value is amplified up to a full swing voltage and data is ready to be propagated to the data bus.

State 9. Restore operation occurs. Its operation is similar to write operation, states 1-3.

4.2 - Common MLDRAM faults

There are two general types of faults affecting MLDRAM. They are known as *hard faults* and *soft faults*. The protection scheme has to be applied to both types.

Short between word line and cell capacitor
Short between sub-bit line and cell capacitor
Short between two cell capacitors
No connection between sub-bit line and cell
Cell access transistor stuck open
Cell access transistor stuck on
Excessive cell leakage current
Interrupted word line (WL_i, WR_i)
Interrupted sub-bit line (BL, _BL, BR, _BR)
Short between adjacent word lines
Short between adjacent sub-bit lines
Short between word line and sub-bit line
Stuck word line (WL_i, WR_i)
Stuck dummy word line (DL_o, DL_e, DR_o, DR_e)
Stuck bit-line precharge control (PL, PR)
Stuck bit-line equalize control (EL, ER)
Stuck sense amplifier isolation control (IL, IR)
Stuck sense amplifier precharge control (ZL, ZR)
Stuck switch matrix signal (C, _C, X, _X)

Table 4: Hard faults in MLDRAM

Hard faults are permanent and manifested in at least 16 ways [17] as shown in Table 4. Normally, because of the large areas affected by hard faults, they are better repaired with adding spare memory elements. However, in cases such as cell access transistor stuck open or excessive cell leakage current, when fault affects a small number of cells, it is better to repair the damage with ECC. The second type of fault is a soft fault. Due to reduced noise margins (600 mV for 0.18 μm technology) MLDRAM becomes very susceptible to soft errors. Soft errors mostly occur because of the α-particles, that are He^{2+} nuclei (two protons, two neutrons) emitted from radioactive elements during decay. Traces of such elements are unavoidably present in the device packaging materials. With emitted energy of 8 to 9 MeV, α-particles can travel up to 10 μm deep into silicon. While doing so, they interact strongly with the crystalline structure, generating roughly 2 x 10^6 electron-hole pairs in the substrate. The soft error occurs when the trajectory of one of these particles strikes the storage node of a memory cell.

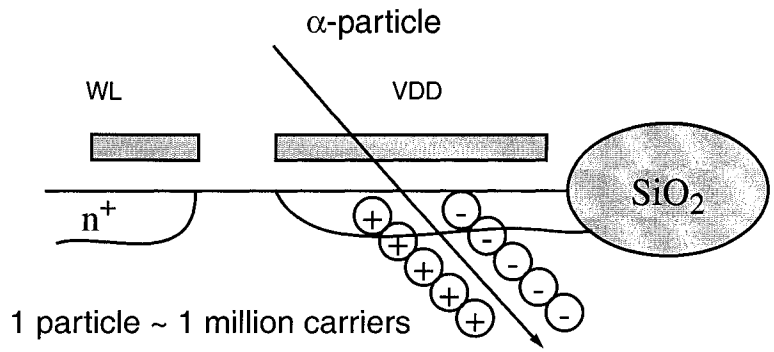


Figure 4.4: α-particle induces soft error

Consider the cell of Figure 4.4. Electrons and holes generated by a striking particle diffuse through the substrate. Electrons that reach the edge of the depletion region before recombining are swept into the storage node by the electrical field. If enough electrons are collected, the stored value can change [19].

4.3 - Induced ECC code for MLDRAM

Figure 4.5 shows conventional ECC for a 1bit/cell memory, that is also used for MLDRAM [18], [9]. During the write operation, k-bits wide input data propagates through

encoder (check bits generator) and is stored in additional memory location. During the read operation, data passes through check bit generator again and stored check bits are compared with new check bits. A block responsible for the compare operation is called a *syndrome*. A decoder is used to decode information returned by syndrome. Depending on the values returned by decoder, the correction circuitry either corrects a single bit error or generates the double bit error detection signal.

In order to protect k-bits of information we need c parity check bits

$$2^c \geq k + c + 1$$

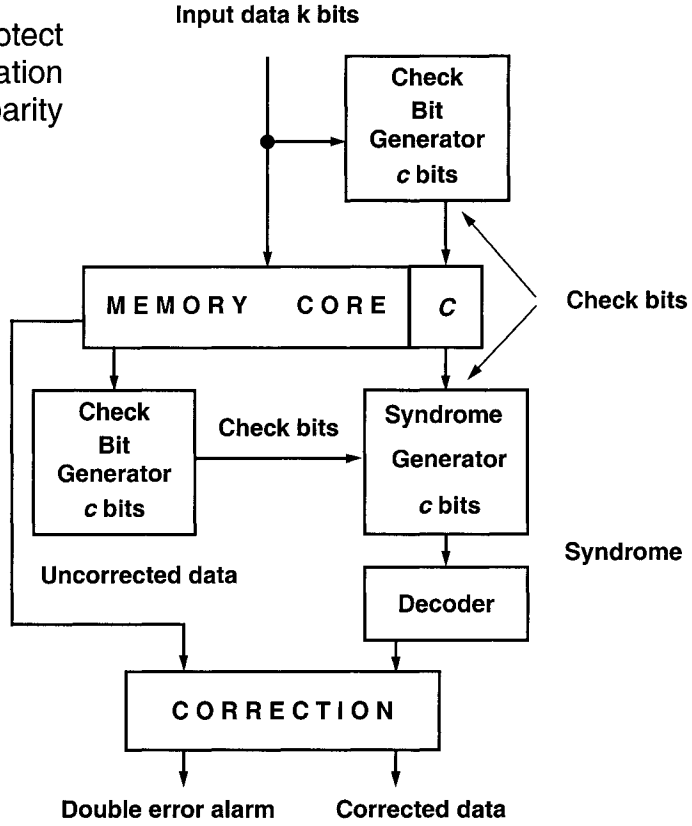


Figure 4.5: Conventional modified Hamming ECC for 1-bit/cell memory

Unfortunately, this architecture of the ECC is not efficient for the MLDRAM, for reasons apparent from Figure 4.1. The noise margin between levels of storage 01 and 10 is only 1/3 of *VDD*. This fact creates high probability of errors with Hamming distance 2 (difference between two binary words). Another drawback is that this ECC has to wait for both MSB and LSB to be ready for processing through ECC.

The proposed ECC, shown in Figure 4.6, is able to avoid and solve problems that occur using conventional ECC. Based on a fact that ECC is able to interact with the memory and

affect its operations, this ECC is called *induced*. Induced ECC exploits an idea that MSB and LSB are read out at different time and that LSB value strongly depends on previously read MSB value.

In order to protect k-bits of information we need 2z parity check bits

$$2^z \geq z + \frac{k}{2} + 1$$

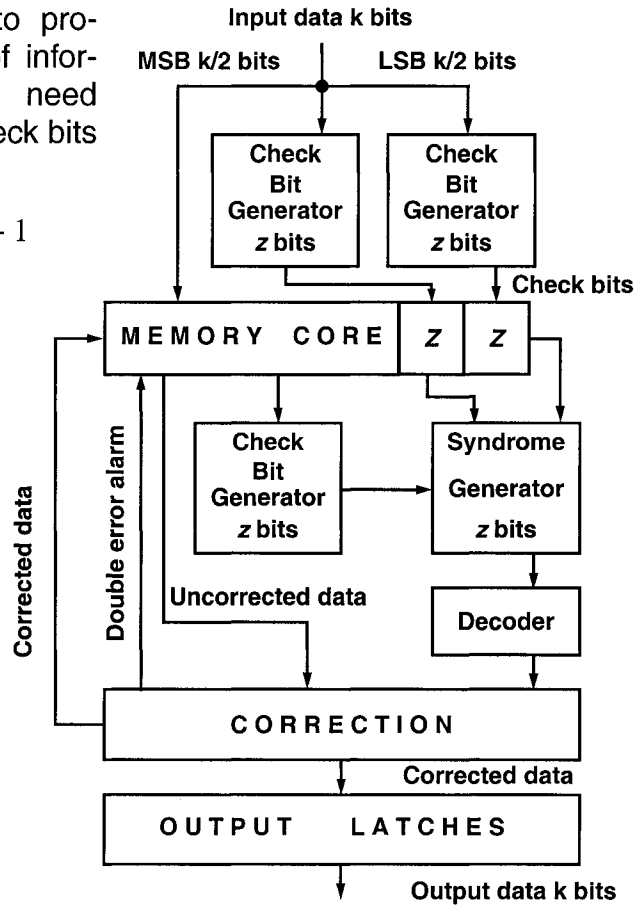


Figure 4.6: Induced ECC for 2-bit/cell memory

The operation of the induced ECC is quite simple. During the write operation, data is divided in two groups: MSB group and LSB group. Since MSB and LSB can be written at the same time, they are processed through different check bit generators. This also can be done by using only one check bit generator, but in this case, the time to generate check bits will double.

Check bits are generated according to Table 5. The table is constructed in such a way that the effect of each erroneous bit is unique; a unique combination of parity check bits is produced in each individual case, so the erroneous bit can be easily located. Also one total parity check bit has to be generated in order to detect a double bit error. The check bits

generator schematic is shown in Figure 4.8. Check bits are stored in two separate locations, z bits for MSB and z bits for LSB.

k	c1	c2	c3	c4	c5	
0				c4	c5	3
1			c3		c5	5
2			c3	c4		6
3			c3	c4	c5	7
4		c2			c5	9
5		c2		c4		10
6		c2		c4	c5	11
7		c2	c3			12
8		c2	c3		c5	13
9		c2	c3	c4		14
10		c2	c3	c4	c5	15
11	c1				c5	17
12	c1			c4		18
13	c1			c4	c5	19
14	c1		c3			20
15	c1		c3		c5	21
c1	c1					16
c2		c2				8
c3			c3			4
c4				c4		2
c5					c5	1
	2^4	2^3	2^2	2^1	2^0	

Table 5: Check bits generation for induced ECC (k=32).

The most left column of the table shows all 16 bits of the bus plus check bits c1 to c5. The right most column represents positions of the erroneous bit for row decoder. For example, if bit 2 is affected, the row number 6 will be excited since the unique combination of check bits c3 and c4 is responsible for bit 2. It should be noted that bit c5 represents less significant bit and c1 represents most significant bit as it may be seen at the bottom of Table 5. Comparing check bits c1 and c2 in Table 5, one can see that the check bit c2 can serve as a parity generator for bits 4 to 10, inclusively, and check bit c1 as a parity generator for bits 11 to 15. This fact was exploited for total parity generator bit c6.

4.3.1 - Efficient total parity check

Normally, in order to get total parity check bit, one should use 15 two-input XOR gates, as shown in Figure 4.7. Comparing Figure 4.7 and Figure 4.8 one can notice possible improvement for total parity check bit generation, which saves time, area and power consumption for given ECC. As seen from Table 5 and Figure 4.7 check bit c1 serves as a parity generator for bits 11, 12, 13, 14, 15 and check bit c2 serves as a parity generator for bits 4, 5, 6, 7, 8, 9, 10.

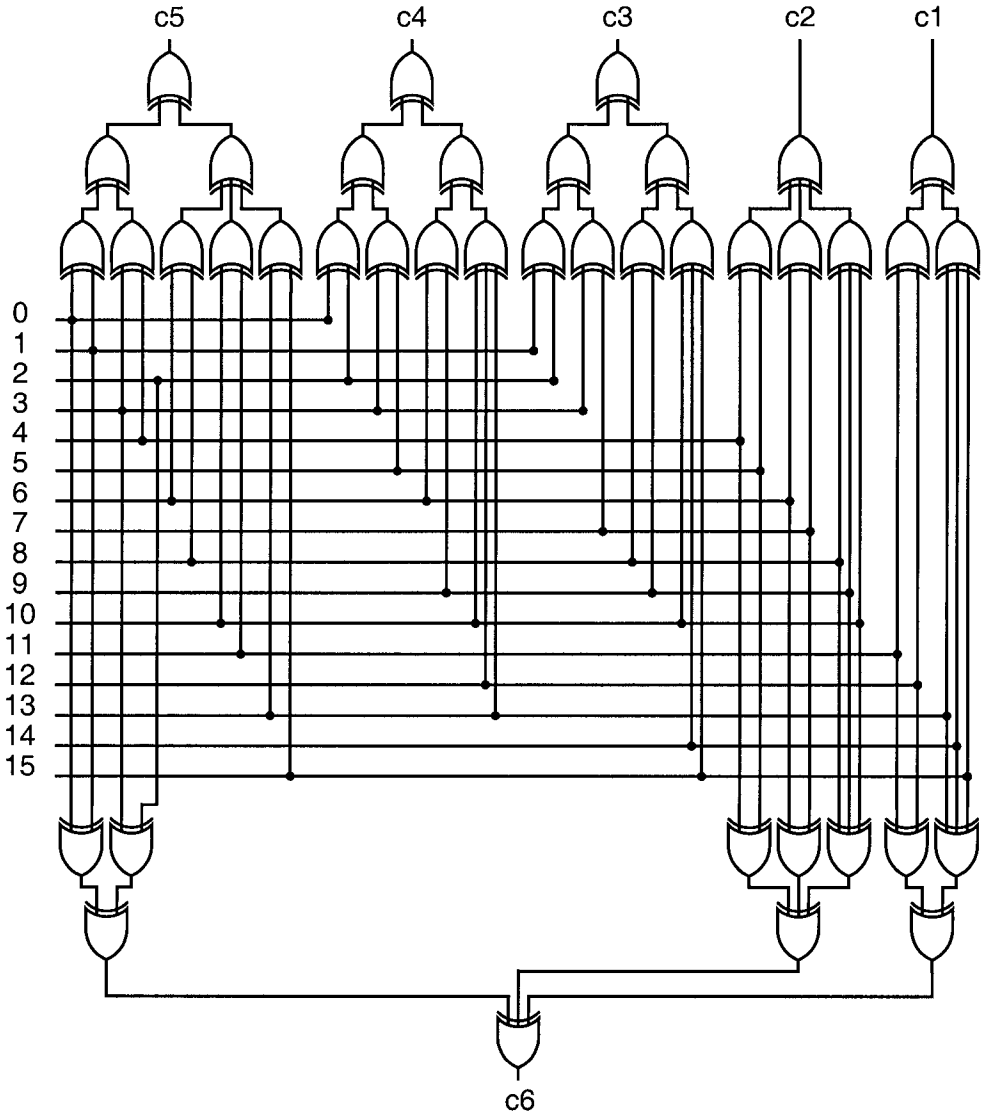


Figure 4.7: Check bit generator without improvement

Since check bits c1 and c2 are used as parity generators for completely different bits we understood that it was possible to reuse them for total parity generator bit c6. As shown in

Figure 4.8, total parity check bit was constructed using check bits c1, c2 and 5 additional XOR gates, instead of using additional 15 XOR gates for total parity generation, which significantly reduces area occupied by induced ECC.

Check bit generators take *k/2* bit on the input and encode *z* bits on the output, where *z* satisfies bounds in Figure 4.6.

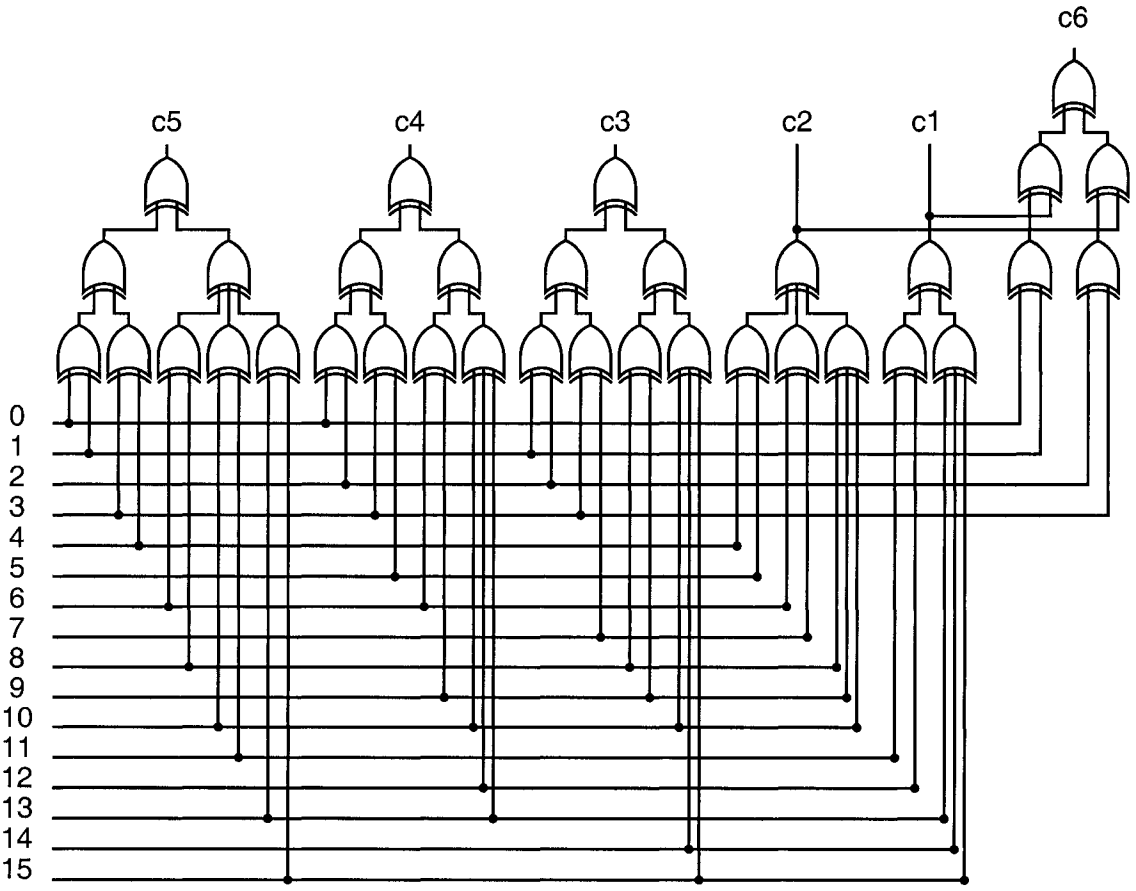


Figure 4.8: Improved check bit generator for induced ECC

During the read operation, the bits that were generated during the write operation (c1w, c2w, c3w, c4w, c5w) are compared with newly generated bits (c1r, c2r, c3r, c4r, c5r) through syndrome circuitry, as shown in Figure 4.9. There are three possible outcomes that can happen during the comparison.

A) the first case, there are neither single errors nor double errors. In this case, the binary syndrome S1 through S5 returns "zero" and the output of total parity bits c6w and c6r do not differ, which returns zero on the output of the XOR gate for c6 comparison. A "zero"

from total parity check bit c6 propagates through inverter, and there are "zero" and "one" at the inputs of AND gate, which gives a negative result for double bit error detection alarm.

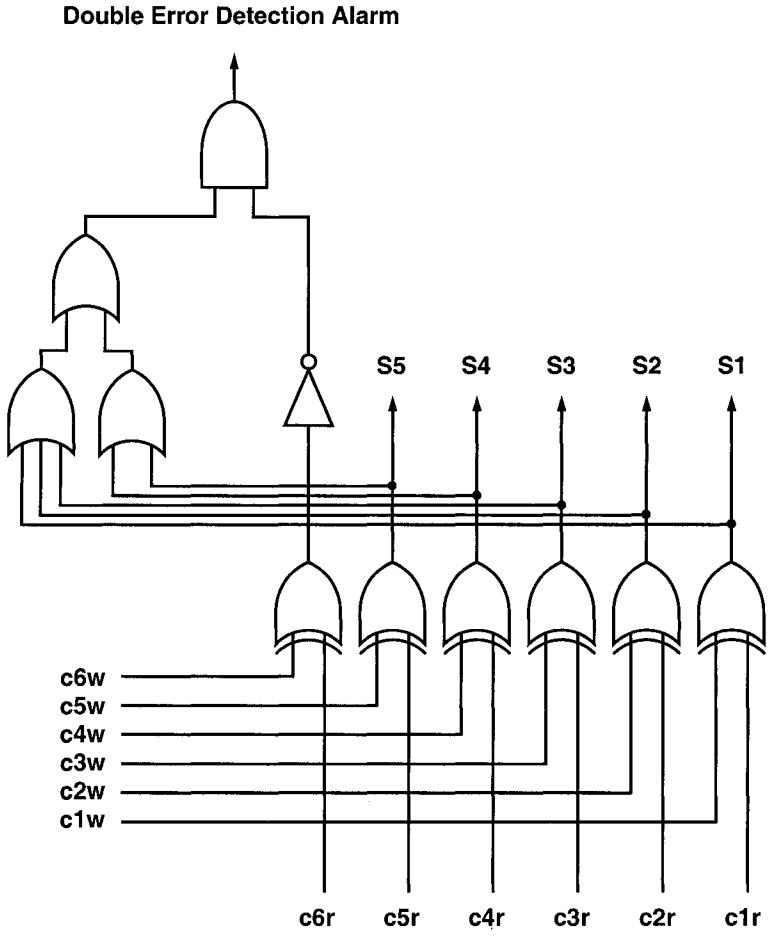


Figure 4.9: Syndrome generation and DED circuitry.

B) the second case, there is a single error and no double errors. In this case, the binary syndrome S1 through S5 gives some binary value other than "zero" showing the exact location of the error. The total parity bits c6w and c6r also differ, which returns "one" on the output of XOR gate for c6 comparison. "One" from total parity check bit c6 propagates through inverter and there are "zero" and "one" at the inputs of AND gate, which again gives a negative result for double bit error detection alarm.

C) the third case, there is a double bit error. In this case, the binary syndrome S1 through S5 gives some binary value other than "zero" showing the location of the error, but the

location is erroneous itself and should not be used for correction. At the same time, bits c6w and c6r do not differ, which returns "zero" on the output of XOR gate for c6 comparison. "Zero" from total parity check bit c6 propagates through an inverter and there are "one" and "one" at the inputs of AND gate, which now gives a positive result for double bit error detection alarm. This situation produces double bit error alarm. If there are more than two erroneous bits, this circuitry fails to work properly.

After the comparison, the data has to propagate to the decoder and correction circuitry. The decoder and correction circuitry were implemented as a single block, shown in Figure 4.10. The decoder is implemented on a basis of a standard NOR decoder with improved enable signal. All uncorrected data from a memory location propagates to the inputs of XOR gates and is compared to the same data that propagated through ECC circuitry. The operation of the decoder and correction circuitry is best demonstrated on an example. Assume that the bit 7 originally was "0", and after read operation it was read as "1". In this case, as it is shown in Table 5, only check bits c2 and c3 will differ after read operation. This combination corresponds to binary value 12 as it is seen from right part of Table 5.

Syndrome bits S1 through S5 will propagate to decoder. As soon as the decoder is enabled with negative signal, only the row that goes to the same XOR gate as bit 7 will be pulled up "high". Since corrupted value is "1" and exited row produces "1", the XORed output (bit7out) will give a corrected value "0".

The same is true for the opposite corruption polarity. Assume that the bit 7 originally was "1" and got corrupted to "0". The same row will be exited and corrupted "0" XORed with exited row will give "1".

An additional feature that improves the decoder function is the enable signal. NMOS and PMOS transistors in the left part of the Figure 4.10 serve for precharging and discharging the row lines. In order to precharge one of the decoder lines, enable has to become "low". In this case, all PMOS transistors in the left part will be turned on, but only one row line will be pulled up since the rest of the lines will be discharged by NMOS transistors in the right part of the circuitry.

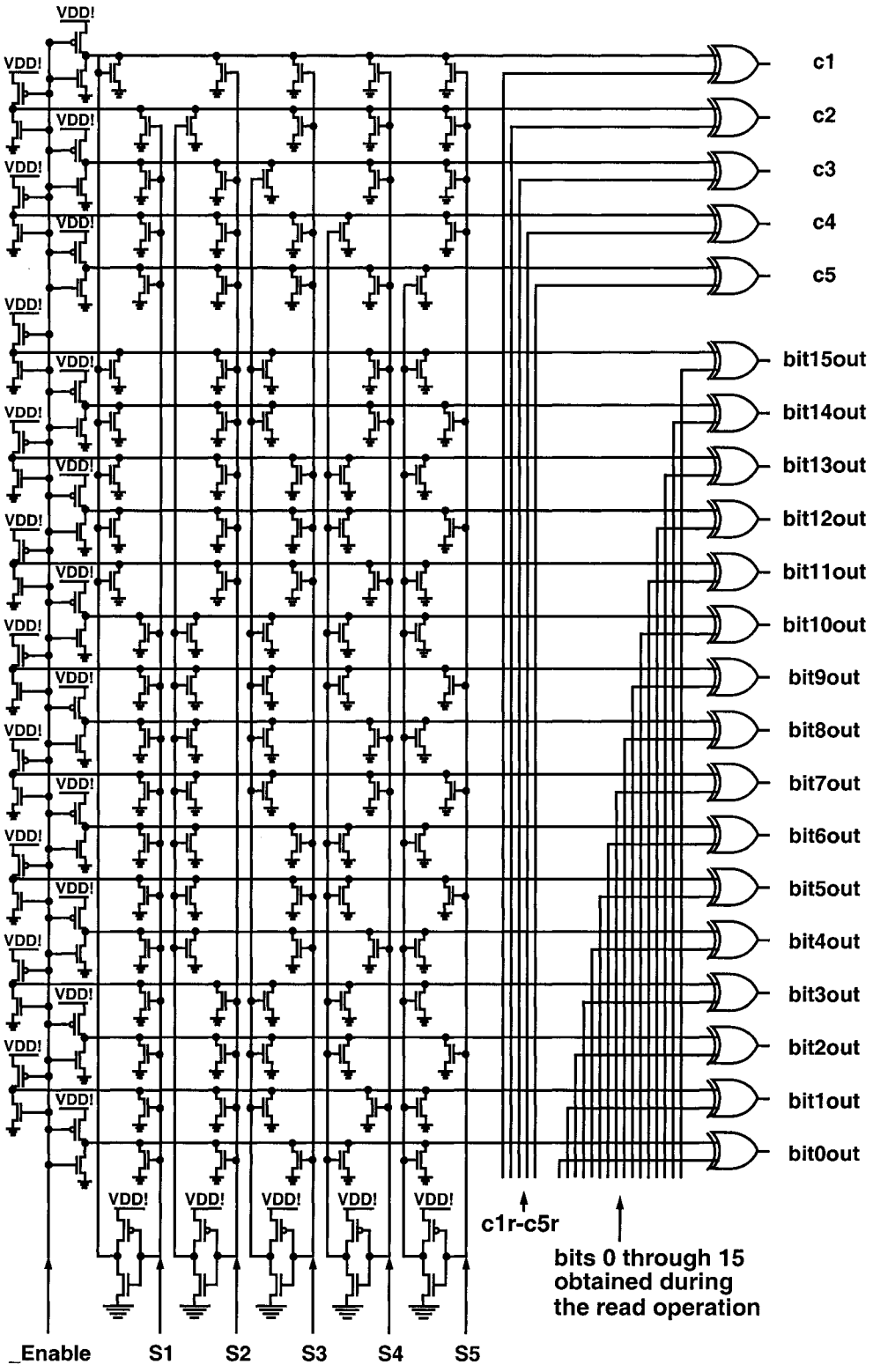


Figure 4.10: The decoder and the correction circuitry schematic

However, if at certain moment the decoder has to be disabled, the enable signal has to become "high". In this case, the enable signal is pulled up and NMOS transistors at the left part of the circuitry are turned on. As soon as NMOS transistors are turned on, all row lines become discharged, and the decoder does not make any changes to the correction circuitry. The addition of the extra NMOS transistors in the left part of the decoder prevents ECC from erroneous correction and saves power consumption, since we do not need to apply additional signals on S1through S5 in order to discharge all row lines.

As shown in Figure 4.3, an MSB is generated first, and is ready at state 7. At this point, all MSBs can be processed through check bit generator and compared with MSBs that are stored in the memory. After processing MSBs through the rest of the ECC, there are three possible outcomes available.

• *First outcome*: there is a single bit error amongst MSBs. ECC is able to correct it and returns correct values back to memory. By doing so, we avoid a double error, since the wrong MSB value will be automatically written back to the cell (state 7 of Figure 4.3). Also, the memory is protected against the error between levels 01 and 10 (Hamming distance two).

• *Second outcome*: there is a double bit error. ECC is not able to correct it, but it returns the double bit error detection alarm to the memory controller. Since the double bit error between MSB will provoke double error between LSB, it does not make any sense to continue the read operation and upon receiving MSB, a double error alarm signal memory controller stops the read operation at state 7. As it can be seen from Figure 4.3, by doing so we save time and power consumption. Simulations show that time saved is about 35 ns for 0.18 μm technology.

• *Third outcome*: there are no errors between MSBs. MSB values are latched to output latches and ECC is ready for LSB checking. Now, LSB check bits are propagated through the same blocks as MSB. Depending on the outcome of the correction circuitry, data is either latched to output latches and ready to be read or ECC sends double bit error signal to the processor and memory waits for further instructions.

4.4 - Performance of the induced ECC

The reliability model was derived and all modelling was performed using MATLAB. The model is *combinatorial*, which means that the set of the operational states of the system is categorized in such a way that the probabilities of each of the states can be determined by combinatorial means [9].

As shown in Figure 4.11, the MLDRAM can be described as a state flow diagram with all possible states it can tolerate. Curved lines show normal transitions and straight lines show faulty transitions that can happen due to several reasons. Figure 4.11 (a) shows that MLDRAM protected with conventional ECC can not tolerate faulty transitions with Hamming distance 2. Figure 4.11 (b) shows that MLDRAM protected with induced ECC can tolerate any faulty transition between the states.

Equation (4.1) describes reliability $R(\gamma)$ of the previously described system.

$$R(\gamma) = \sum_{i=1}^{f} \binom{F}{i} (1-\alpha)^i \alpha^{(F-i)} \tag{4.1}$$

where F is a total number of faulty transition that can happen in the system. From Figure 4.11 it follows that there are 6 faulty transitions which can happen in MLDRAM; f is the number of faulty states that system can tolerate. From Figure 4.11 follows that MLDRAM protected with conventional ECC can tolerate only up to 4 faulty transitions (f=4) and MLDRAM protected with induced ECC can tolerate up-to 6 faulty transitions (f=6). α is the probability of the possible faulty transition. α itself is a function of the cell area A_{cell} and of the, so called, γ factor and is expressed by Equation (4.2). Actually, for MLDRAM γ is the main factor that affects MLDRAM system reliability, which itself depends on many factors, such as noise margins of the system, defect density and so on.

$$\alpha = e^{-(A_{cell} \cdot \gamma)} \tag{4.2}$$

Equation (4.1) is graphically shown in Figure 4.12. The induced ECC code significantly improves reliability of the MLDRAM comparing to the conventional ECC code. It has to be mentioned that the plot for MLDRAM without ECC was obtained as a system that can not tolerate any faulty transitions.

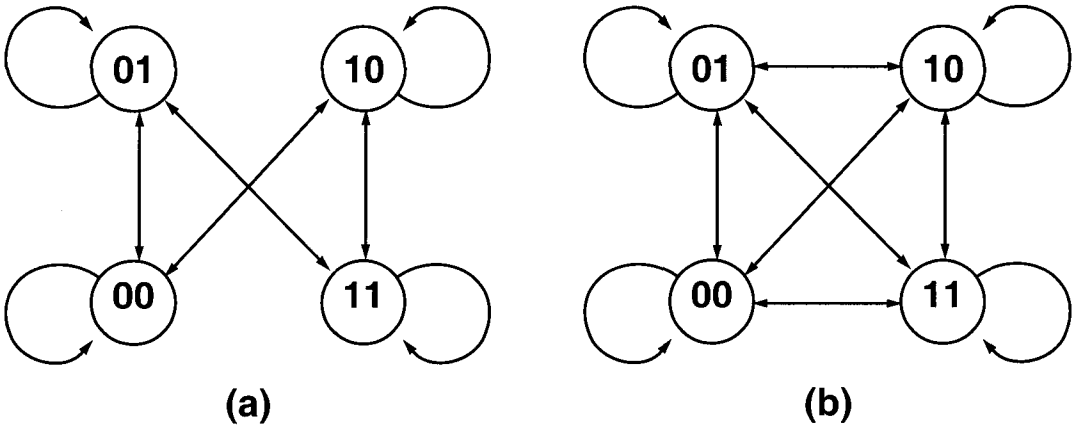


Figure 4.11: State flow diagram for MLDRAM **(a)** MLDRAM protected with conventional ECC **(b)** MLDRAM protected with induced ECC

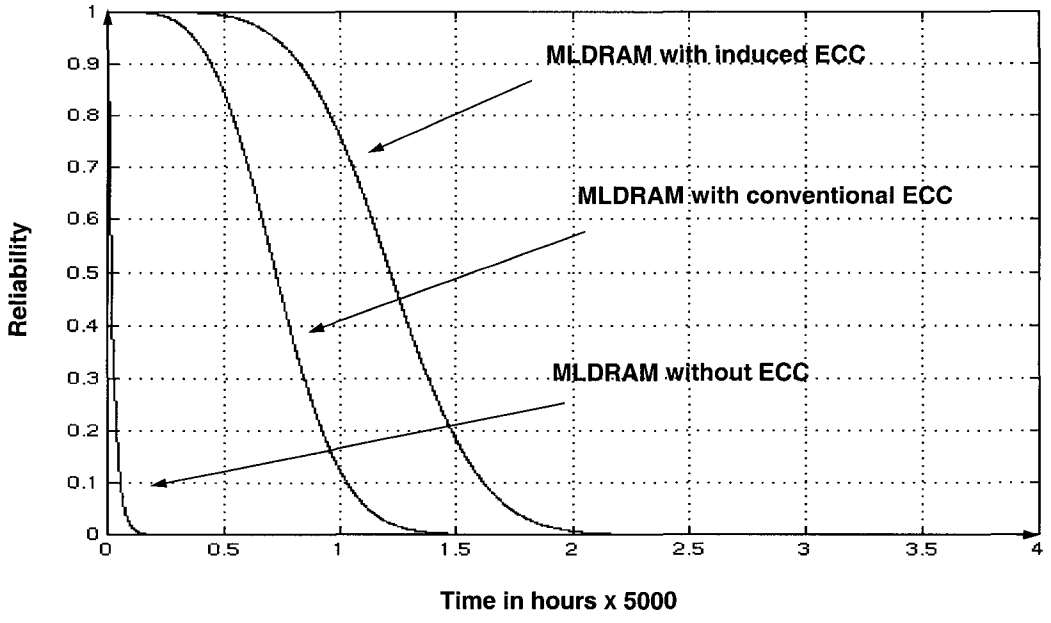


Figure 4.12: Reliability improvement with induced ECC

By designing induced ECC, one has to cope with area and delay propagation that eventually affects the overall memory performance. As follows from Figure 4.13 (a), bus width dramatically affects area occupied by check bits. Figure 4.13 (b) shows that it is more profitable to have wide buses than narrow ones. For example, for 2 bit wide bus we need 4 check bits, which makes it 200% of the area occupied by redundancy bits; for 16 bit wide

bus we need already 50% redundancy and so on. At the same time, one should deal with propagation delay introduced by ECC in the whole circuitry. As shown in Figure 4.14, the propagation delay time dramatically increases with increasing the number of information bits. For example, just increasing the bus from 4 bits to 100 bits introduces as much as almost 7 times more delay into the circuitry. Comparing Figure 4.13 and Figure 4.14, it is obvious that bus width introduces completely opposite effects on the redundancy percentage and propagation delay. For redundancy percentage it is better to have a larger bus. Meanwhile, for propagation delay it is better to have the bus width as small as possible.

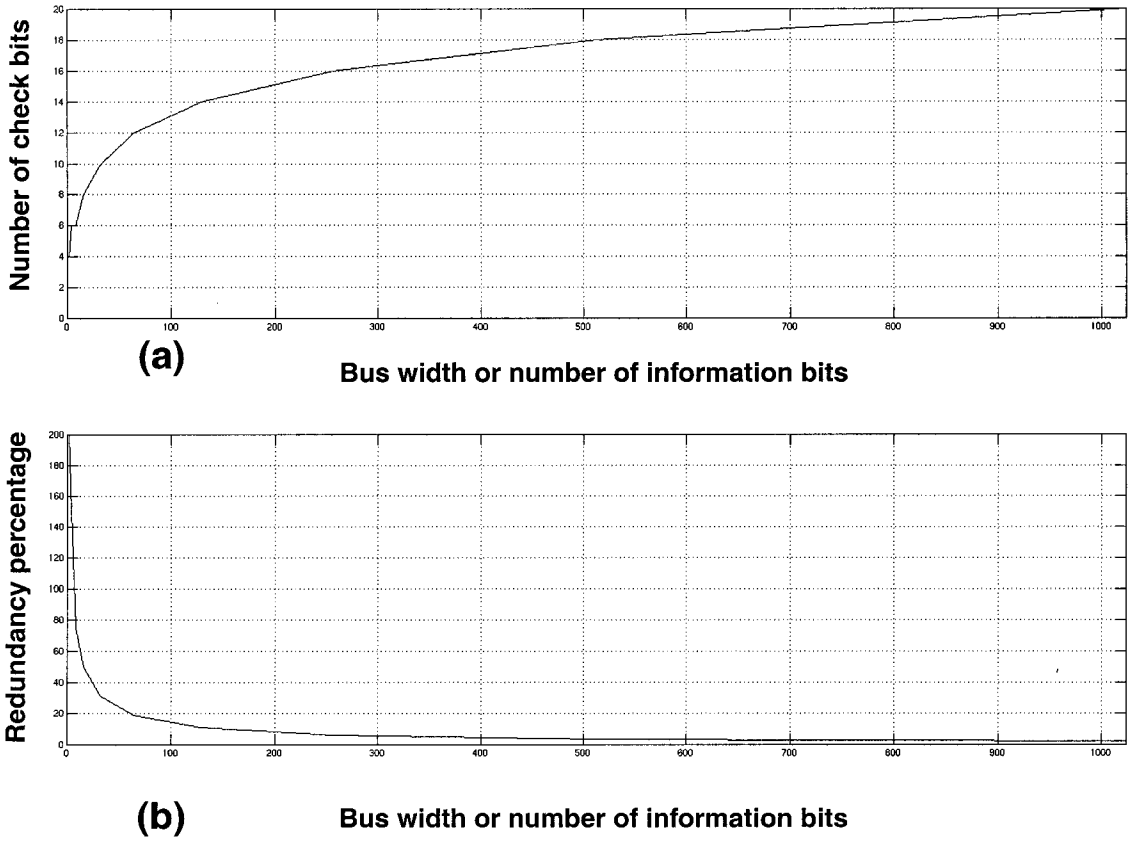


Figure 4.13: Area **(a)** and the redundancy percentage **(b)** of the induced ECC as a function of a number of information bits.

Eventually designer has to choose optimal trade-off that best suits to his needs, either it is a space-saving strategy or an increased speed of operation strategy.

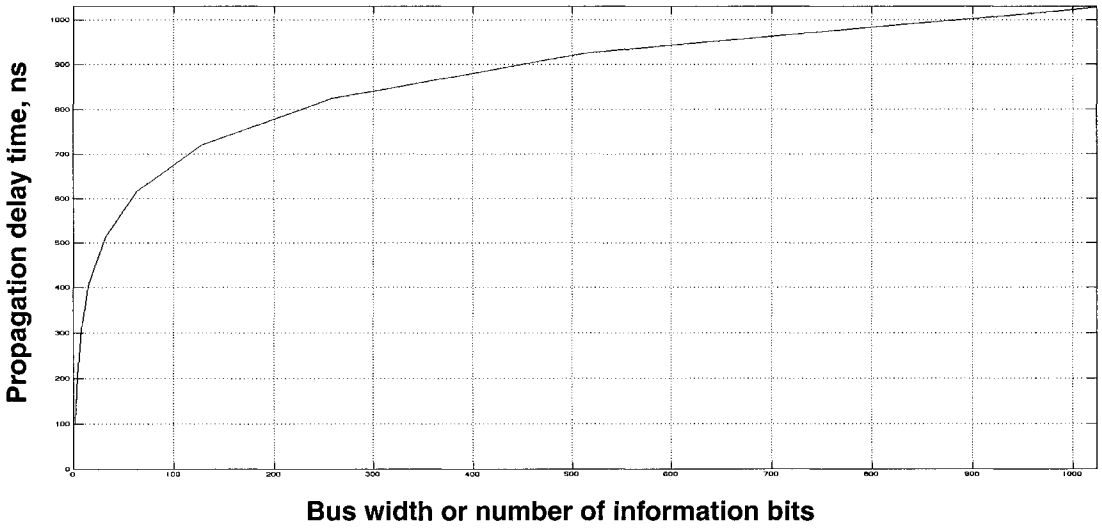


Figure 4.14: Propagation delay as a function of a bus width

4.5 - Conclusions

The conventional ECC can not cope with some of the most common errors in MLDRAM, as they appear to corrupt two bits. The induced ECC scheme improves correction of these errors, and hence increases the MLDRAM reliability. An implementation of induced ECC circuitry that is tightly coupled in MLDRAM sensing circuitry is presented and its performance is shown to surpass that of the conventional ECC.

Chapter 5 - New Embedded Memory Architecture for Enhanced Yield, Performance and Power Consumption

5.1 - Introduction

With advances in deep-submicron CMOS technology, it is practical to create Systems-On-Chip (SOCs), where designers can integrate many components on the same die [21]. SOCs provide a lot of flexibility, but engineers have to account for effects such as interference between digital and analog parts, yield and reliability. Proper function of the whole system depends on the function of each block. They all have to operate correctly.

Since software gives more flexibility, distinguishing features of SOCs are often implemented in software, rather than in hardware. It is becoming more profitable to create hardware as small as possible, and use instead a large software component. To accept these trends, engineers need more and more fast and reliable memory, which consumes less power. Embedded memories fit perfectly to these conditions. This chapter investigates the means to improve the yield of embedded memory, while retaining speed and power performance bounds, if redundant elements are to be employed.

5.2 - Cross-Shared Redundancy

In this chapter, we introduce a cross-shared redundancy (CSR) scheme. Architecture of the CSR model was designed in consideration of yield improvement and protection of the embedded memory against most important types of failures, such as single-cell, row, column and chip-kill failures. As shown in Figure 5.1, on example of 4 Mbit memory, the memory core is organized as a square array of M independent blocks (M=16 in Figure 5.1).

Each block consists of 512 rows and 512 columns and has its own column and row decoders. There are also redundant columns and rows for fault tolerance, and BIST (Built-In-Self-Test) circuitry and *main* MC (Memory Controller) in the middle of the core. In addition to the main controller, there are four redundant memory controllers to protect memory from chip-kill or fatal defects. As known from practice, defects tend to occur in clusters and do not spread evenly over the chip [20],[7],[8]. For this model, if a cluster error occurs and destroys the memory controller, the defected controller can be easily replaced with one of the four spare controllers. This feature significantly augments yield of the whole chip (probability of chip kill defect reduced by power of 4), since memory functioning is very important for a SOC.

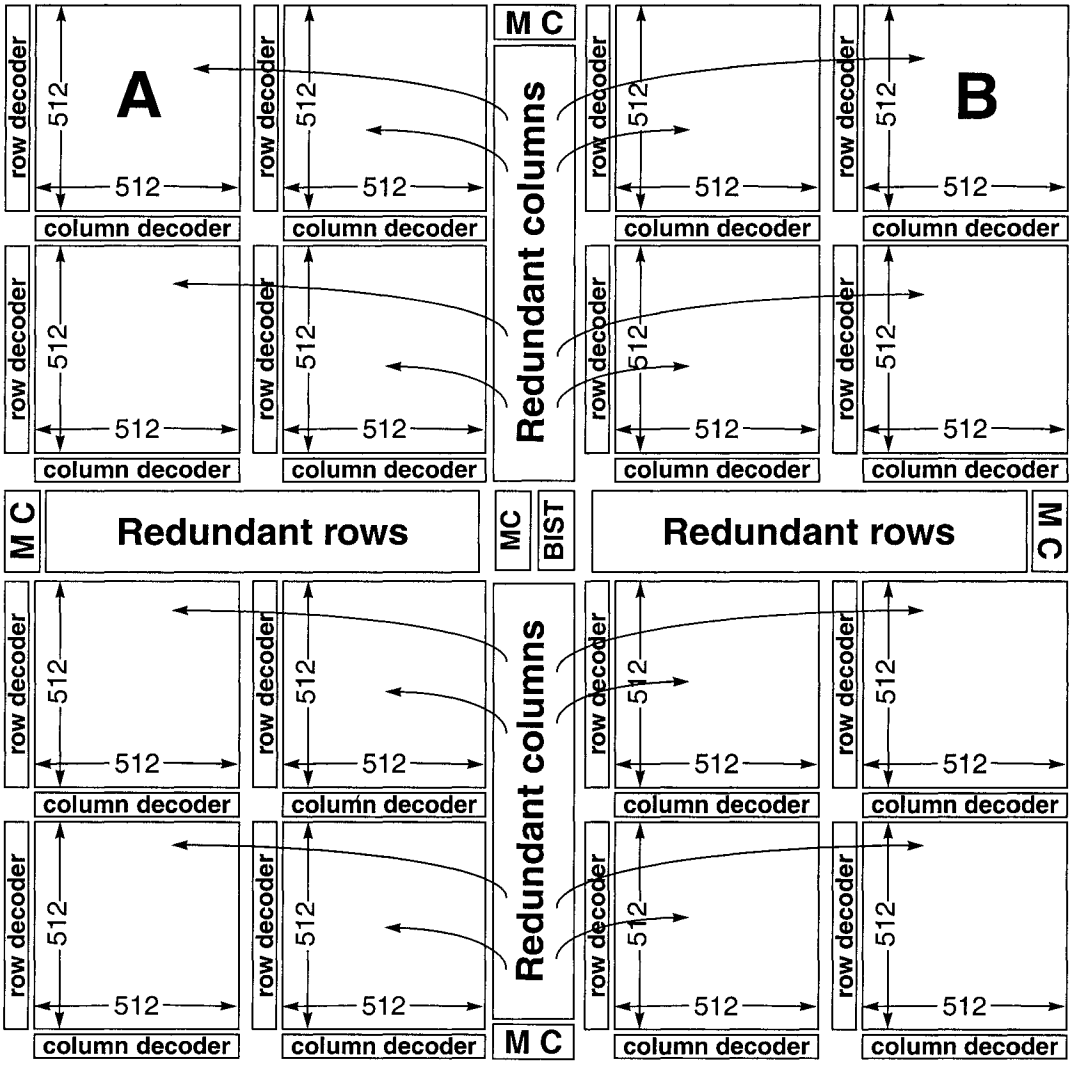


Figure 5.1: Cross-Shared Redundancy memory

As seen from Figure 5.1, redundant rows and columns are positioned in such a way, that redundant columns between blocks A and B may be used for both blocks depending on where the fault happens. The same is true for rows.

The same effect could be achieved just by placing the redundancy on the side and at the bottom of the core, but for speed and low-power consumption reasons, this model is more suitable. Power consumption is defined by equation $P_d = kV_{DD}^2 C_L f$, where C_L is load capacitance, k switching activity, V_{DD} is a power supply, f is a switching frequency. One way to reduce power consumption is to reduce load capacitance. For CSR model, by placing redundancy in the middle of the core, load capacitance of the access wire is reduced by

a factor of 2 in the worst case. For example, if there is a couple of destroyed columns in the left-top part of block **A** and there are redundant columns on the right side of the global core, each signal has to propagate through the whole memory and it needs to charge wire twice as much compared to our model, in order to activate necessary operation.

Meanwhile, with redundancy in the middle of the memory core, the time for the signal to propagate will be reduced to half, and the wire to charge will be twice shorter.

5.3 - Failure Types

As shown in [10], if all faults in memory were single-cell failures, the error-correcting code could bring yield up to 99.9%. However, most of these failures affect the chip support circuits, and the word and bit lines. For deep-submicron technology, this is even more true. Transistor sizes and, consequently, cell sizes become smaller. Chances that defect cluster affects only one cell, are very low. Normally, it is several cells, word and bit lines that are covered with a cluster failure. These types of defects can not be repaired by ECC (Error-Correcting Code) alone, due to 2-dimensional nature of cluster defects. It is essential to have sufficient row and column redundancy to repair these faults.

This model covers three major types of failures. Each type, in turn, consists of different subtypes of faults.

The first type is a *single cell failure*. This type of fault describes the situations when the defect is contained inside the cell. This can happen for several reasons.

1) Transistor damage happens more often for deep-submicron technologies [5].

2) Capacitor damage (very important for state-of-art DRAMs), happens due to several reasons, such as absence of metal [6] or a short circuit to another metal layer. As shown in Figure 5.2, cluster B damages only four cells and does not harm whole row or column. This type of fault is efficiently repaired with ECC, in order not to waste unnecessary cells of redundant rows or columns.

The second type is a *row failure*. If a cluster error breaks a word line close to the row decoder, the whole row does not function and can be replaced only with a redundant row. As shown in Figure 5.2, cluster *A* destroys rows 5 and 6 completely and it is necessary to

have two redundant rows *I* and *II* to repair such a damage. This type of faults happens for several reasons, such as bridging faults, cluster faults and many others.

The third type of faults is a *column failure*. It is known that columns are more susceptible to failure than rows [10]. There are several reasons why column failure happens. They can be classified in following way:

a) one or both bit lines are damaged;

b) precharge circuitry failure;

c) sense amplifier failure.

This type is especially important because the very sensitive and precisely tuned-up sense amplifiers have to sense shrinking voltage levels, as the technology shrinks.

For this type of failures, ECC is not a suitable solution, because it is impossible to repair sense amplifiers and the only acceptable solution is to replace the column, which contains a damaged sense amplifier, with the redundant one.

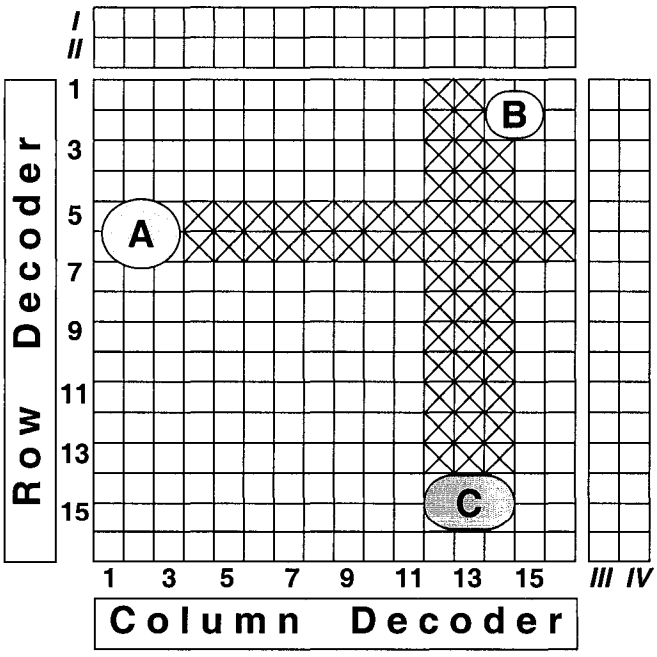


Figure 5.2: Embedded memories major types of failures

5.4 - Yield Modeling

Since redundancy allocation is a NP-complete problem, it is impossible to describe exact allocation of redundant columns and rows. The model presented below describes rather approximate outcome of spare allocation. Since all three types of failures are independent and can happen at the same time, yield is obtained if their probabilities are multiplied [24]. Our model uses combined Poisson and Binomial distributions for yield modelling [7]. The Poisson distribution is used to describe the yield of a single cell and the whole core, in the case when there is no redundancy. Binomial distribution describes the yields of individual columns and rows. The yield of one cell, i. e. probability that one cell functions correctly, is given by Equation (5.1)

$$P(cell) = \lambda_{cell} = e^{-(A_{cell} \cdot D_0)} \tag{5.1}$$

where A_{cell} is the area of one cell and D_0 is the defect density, which depends on process variations and process conditions. D_0 is defined on the basis of empirical data for a specific process. There are three possible cases to consider to obtain memory yield.

5.4.1 - Memory without redundancy

In this case, memory is not protected with redundancy and can not tolerate any damage. In this case, yield is expressed as probability of not having any faults in the memory core. Since there are N_{row} times N_{col} cells in memory block (where N_{row} is the number of rows and N_{col} is the number of columns), probability that a block functions without any faults will be expressed by Equation (5.2)

$$Y_1 = e^{-(A_{cell} \cdot D_0 \cdot N_{col} \cdot N_{row})} \tag{5.2}$$

There are M blocks in the core and there is equal possibility for any of them to be damaged, the yield of the whole core is going to be as in Equation (5.3)

$$Y_{wor} = (Y_1)^M \tag{5.3}$$

where Y_{wor} stands for *Y*ield *w*ith*o*ut *r*edundancy.

5.4.2 - Memory protected only with redundant rows or columns

In this case, memory can tolerate only one type of damage: either row or column, but not both. Normally, this is a column redundancy because columns are more susceptible to failure. In this case, yield is expressed as the probability of having a certain number of failed columns or rows. Yield is described with combined Poisson and Binomial distributions. Yield of one column is expressed with Poisson distribution and the event of damage happening in several columns out of total number of columns is described with Binomial distribution. Since the number of cells per column is N_{row}, probability that one column functions is defined by Equation (5.4)

$$\lambda_{col} = e^{-(A_{cell} \cdot D_0 \cdot N_{row})} \tag{5.4}$$

Probability of having c failed columns out of N_{col} is given by Binomial distribution, Equation (5.5)

$$P(c \ \ failed \ \ columns) = \binom{N_{col}}{c}(1-\lambda_{col})^c \lambda_{col}^{(N_{col}-c)} \tag{5.5}$$

Now consider assumption that there are c failed columns that may be replaced with redundant columns. As is shown in Figure 5.3, any cell in redundancy-protected columns c can be repaired for any damage, but if one of the rows of length N_{col}-c is damaged, the memory can not function properly already. Since no row redundancy is employed, memory can not tolerate any row damage and 1D-redundancy for one block is described by Equation (5.6)

$$Y_2 = \sum_{c=1}^{R_{col}} \binom{N_{col}}{c}(1-\lambda_{col})^c \lambda_{col}^{(N_{col}-c)} \lambda_{row}^{N_{row}} \tag{5.6}$$

where λ_{row}, the probability that one row of size N_{col}-c will function is given by Equation (5.7) and R_{col} is the number of redundant columns

$$\lambda_{row} = e^{-(A_{cell} \cdot D_0 \cdot (N_{col}-c))} \tag{5.7}$$

For function memory core only several outcomes are possible: either there are no faults

in the core and this event is described with Equation (5.2), or there are several faults but not more than number of redundant columns and these events are described with Equation (5.6). Since all these events are mutually exclusive, the yield of the whole chip protected with 1D-redundancy can be given as in Equation (5.8), where Y_{1D} stands for *yield with 1D-redundancy*.

$$Y_{1D} = (Y_1 + Y_2)^M \tag{5.8}$$

5.4.3 - Memory protected with redundant rows and columns

Now, memory is able to tolerate 2D-cluster damage. The yield of any of M blocks is

$$Yield = P(no\ \ faults) + P(1D) + P(2D)$$

$$Y = Y_1 + Y_2 + Y_3 \tag{5.9}$$

where Y_1 describes yield in case when memory core is free of damage, Y_2 describes yield in case when memory is damaged and it is possible to repair the damage only with redundant columns or rows, Y_3 is the part that describes failures which may be repaired only with redundant columns and rows together given by Equation (5.10)

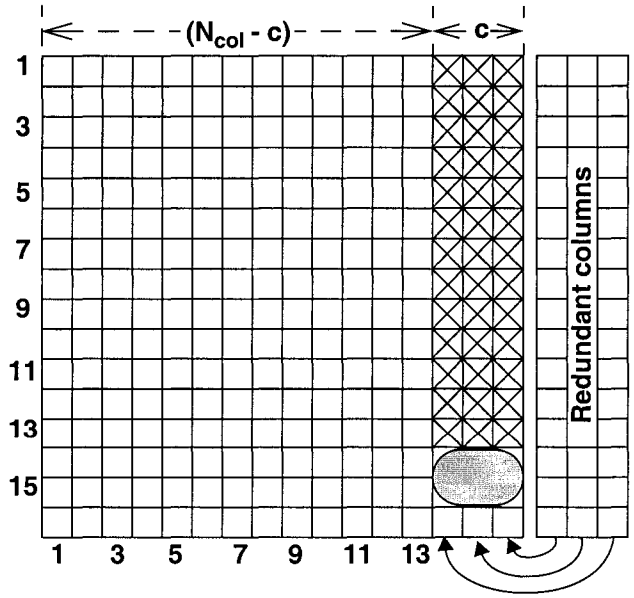


Figure 5.3: Effect of redundancy on yield

The double summation in Y_3 goes through all possible combinations, i. e. 1 column+1 row,

1 column+2 rows, 2 columns+1 row, 2 columns+2 rows and so on, until all redundant columns and rows are exhausted. For example, if there are three redundant columns and four rows (R_{col} =3, R_{col} =4), the model will sum all 12 possible situations which memory can tolerate, with P (c col, r row) denoting probability of having c damaged columns and r damaged rows in the memory core.

$$Y_3 = \sum_{c=1}^{R_{col}} \sum_{r=1}^{R_{row}} \binom{N_{col}}{c}\binom{N_{row}}{r}(1-\lambda_{col})^c \cdot \lambda_{col}^{(N_{col}-c)}(1-\lambda_{row})^r \lambda_{row}^{(N_{row}-r)} \tag{5.10}$$

Since the memory is divided in M blocks and yield of one block is given by Equation (5.9) the yield of the memory will be

$$Y_{2D} = (Y_1 + Y_2 + Y_3)^M$$

where 2D stands for two dimensional redundancy.

5.5 - Results

As is shown in Figure 5.4, the curve *wr* represents yield of the CSR without redundancy, curves *1c, 2c* and *15c* represent models with 1D-redundancy 1, 2, 15 redundant columns and curves *1c+1r, 2c+2r* represent models with two dimensional redundancy with 1 column and 1 row, and 2 columns and 2 rows respectively.

Comparing models with 2 redundant columns (1D- redundancy) and 1 redundant column + 1 redundant row (2D- redundancy), it is obvious that yield is better for two dimensional redundancy. The case, when there are 4 redundant columns (1D- redundancy) compared to 2 redundant columns + 2 redundant rows (2D- redundancy) gives even better results for yield. It is clearly seen from these results that the yield is much better for two dimensional redundancy than for one dimensional, not because of the quantity of the redundancy, but because of the quality of it. One might also notice that the area in space between curves *4c* and *2c+2r* is much larger than area in space between curves *2c* and *1c+1r*. This shows that even little increase in quantity of two dimensional redundancy results in dramatic improvement of yield. Two dimensional redundancy repairs much

more defect types than one dimensional redundancy with the same space occupied.

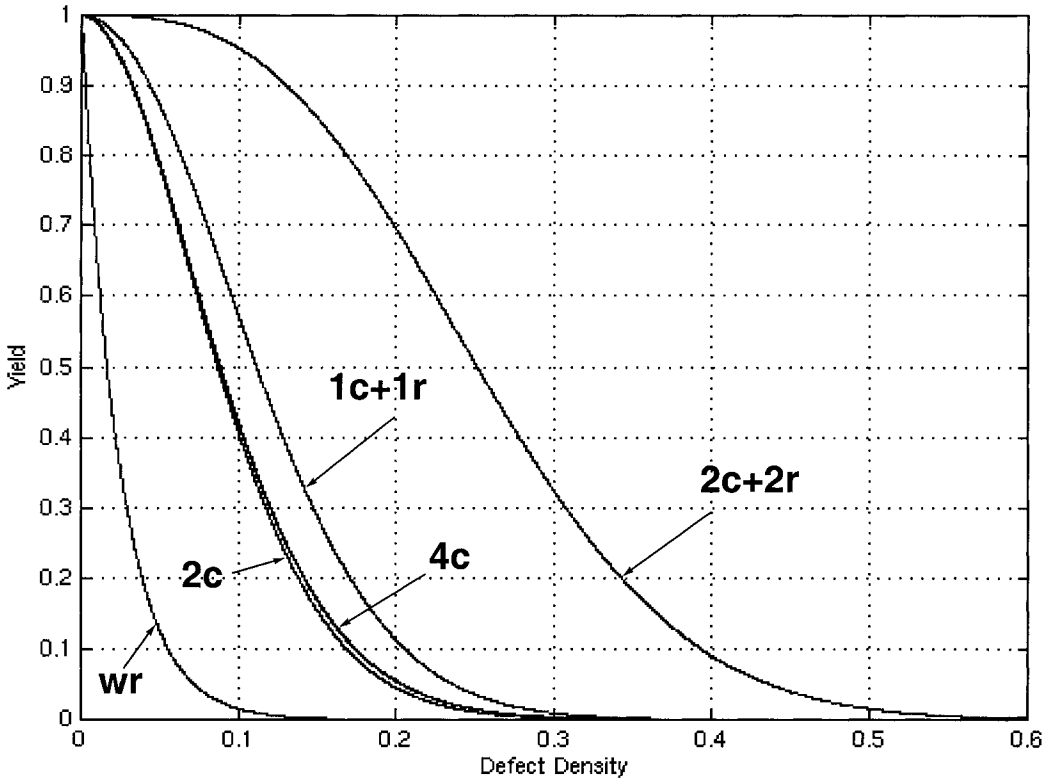


Figure 5.4: One dimensional versus two dimensional redundancy.

Another interesting observation is that after certain value, the yield of the model with single redundancy converges to some point and does not improve further. As is shown in Figure 5.5, comparing curves *wr, 1c, 2c, 4c, 15c,* yield improvement eventually stops when approximately 4 redundant columns are employed and curves *4c* and *15c* converged to the same line, and are seen as one single line with this scale. This happens because single redundancy alone is not sufficient to repair all types of faults in a core. Increasing the redundancy, which is not able to repair certain defects gives nothing, and memory is still going to fail.

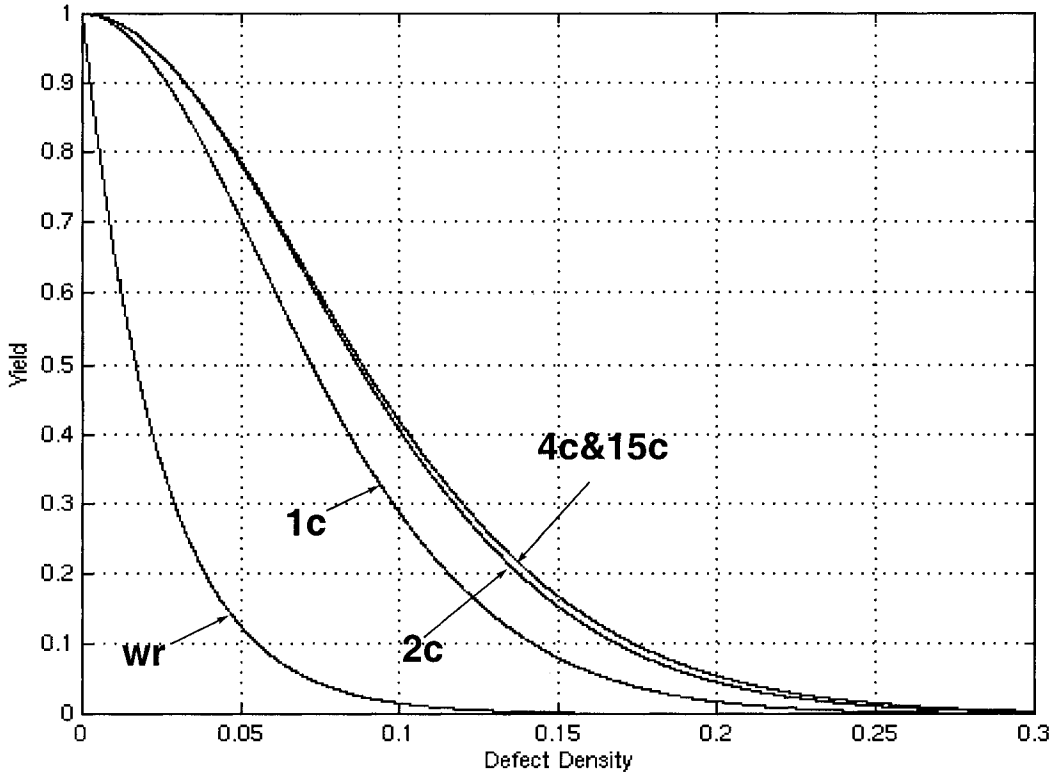


Figure 5.5: Limitations of one dimensional redundancy

5.6 - Conclusions

Selection of suitable redundancy (number of redundant columns, rows, combination of ECC and 2D-redundancy, redundant memory controllers) for specific memory model can significantly increase the yield of embedded memory. Comparing to other existing models [20], [22], [23], [25] the CSR model proves that in the future deep-submicron technologies yield might be improved without excessive performance penalty.

The CSR model achieves better yield and performance due to cross-shared redundancy positioning and additional memory controllers. Power consumption is kept within bounds for the memory without redundancy because of the placement of redundant columns and rows, which allows to charge less wires to access necessary cells.

References

[1] B. Polianskikh, Z. Zilic, "Induced Error-Correcting Code for 2bit-per-cell Multi-Level DRAM", *44th Midwest Symp. on Circuits and Systems, 2001.*

[2] B. Polianskikh, Z. Zilic, "New Embedded Memory Architecture for Enhanced Yield, Performance and Power Consumption", *8th Int. Conf. on Electronics, Circuits and Systems, 2001.*

[3] Semiconductor Business News, *25 October 2000.*

[4] EDTN network news release, *3 July 2001.*

[5] T. P. Haraszti, "*CMOS Memory Circuits*", Kluwer Academic Publishers, Dordrecht, 2000.

[6] S. Gandemer, B. C. Tremintin and J.-J. Charlot, "Critical Area and Critical Levels Calculation in I.C. Yield Modeling", *IEEE Trans. on Electronic Devices, Vol: 35, No. 2, February 1988.*

[7] J. A. Cunningam, "The Use and Evaluation of Yield Models in Integrated Circuit Manufacturing", *IEEE Trans. on Semiconductor Manufacturing, Vol.: 32, May 1990, Page(s): 60-71.*

[8] G. Battaglini and B. Ciciani, "An Improved Analytical Yield Evaluation Method for Redundant RAM's", *Proc. Int. Workshop on Memory Technology, Design and Testing, 1998, Page(s): 117-123.*

[9] B. W. Johnson, "Design and Analysis of Fault-Tolerant Digital Systems." Addison-Wesley, 1957.

[10] C. H. Stapper and H.-S. Lee, "Synergistic Fault-Tolerance for Memory Chips", *IEEE Trans. on computers, Vol. 41, No. 9, Sept. 1992, pp. 1078-1087.*

[11] Microchip Technologies Inc., *PIC12C5XX 8-Pin, 8Bit CMOS Microcontrollers*, Data Sheet DS40139E, 1999.

[12] K. W. Ouyang and M. Marmet, "Voltage Level Sensing Power-up Reset Circuit", *U. S. Patent Serial Number 4,717,840 (14 March 1986).*

[13] A. R. Desroches, "Power Supply Detect Circuit Operable Shortly After on/off Cycle of the Power Supply", *U. S. Patent Serial Number 5,552,736 (3 September 1996).*

[14] A. M. Love and R. D. Norwood, "Method for Generating Power-up Pulse", *U. S. Patent Serial Number 5,203,867 (20 April 1993).*

[15] A. S. Sedra and K. C. Smith, *"Microelectronic Circuits 4th edition"*, Oxford University Press, Oxford, 1998.

[16] D.Lammers and Y.Hara "Memory designs embrace the exotic." EDTN Network(www.ednmag.com), April 22, 2001.

[17] M.Redecker, B. F. Cockburn and D. G. Elliot, "Fault-models and tests for a 2-bit-per-cell MLDRAM", IEEE Design & Test of Computers 1999, pp. 22-31.

[18] C.Wickman, A.Chan, T. Brandon, Y. Xiang, J. Koob, P. Bartosec, B. Cockburn and D. Elliot "Semiconductor File Memory" Micronet Annual Workshop, Ottawa 2001, pp-23-24.

[19] J. M. Rabaey "Digital Integrated Circuits, a Design Perspective", Prentice Hall, upper Saddle River, New Jersey 07458.

[20] Z.Horen and I.Horen, "A model for enhanced manufacturability of defect tolerant integrated circuits", *Proc. Defect and Fault Tolerance on VLSI systems, 1991, Pages: 81-92.*

[21] Y. Zorian, "Yield Improvement and Repair Trade-Off For Large Embedded Memories", *Proc. Conf. Design, Automation and Test in Europe, 2000, Page(s): 69-70.*

[22] S. M. Domer, S. A. Foertsch and G. D. Raskin, "Model for Yield and Manufacturing Prediction on VLSI Designs for Advanced Technologies, Mixed Circuitry and Memories", *IEEE Journal of Solid-State Circuits, Volume: 303, March 1995, Page(s): 286-294.*

[23] K. N. Ganapathy, A. D. Singh, D. K. Pradhan, "Yield Modeling and Optimization of Large Redundant RAM's", *Proc. [2nd] Int. Conf. on Wafer Scale Integration, 1990, Page(s): 273-287.*

[24] K.-I. Imamiya, J.-I. Miyamoto, N. Ohtsuka, N. Tomita and Y. Iyama, "Optimum Redundancy Design for New-Generation EPROM's Based on Yield Analysis of Previous Generation", *VLSI Test Symp. on Design, Test and Application: ASIC's and Systems-On-Chip', 1992. Page(s): 182-187.*

[25] M. Rudack and D. Niggemeyer, "Yield Enhancement Considerations for A Single-Chip Multiprocessor System with Embedded DRAM", *Int. Symp. on Defect and Fault Tolerance in VLSI Systems, 1999. Page(s): 31-39.*