ISSN: 2582-8118 ***Volume 2, Issue 5; Sept 2022***

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYSIS IN SCIENCE AND ENGINEERING

Web: https://www.iarj.in/index.php/ijrase/index

6. Nonvolatile Data Memory with A Real-Time Embedded System

Uma G. Kartha
Lecturer in Electronics,
Govt. Polytechnic College,
Chelad, Kothamangalam,
Kerala.

ABSTRACT

Non-Volatile Memory (NVM) technologies are widely used to implement embedded file systems; however, with latencies and write endurance closer to SRAM and DRAM than to Flash, they are positioned as potential replacements for volatile technologies. In many cases, pre-initialized data structures can be used to make the most efficient use of memory. This paper introduces two NVM technologies and discusses their applications as processor registers, caches, and main memory, as well as a possible design of an embedded system with non-volatile memory components.

KEYWORDS:

Non-Volatile Memory, Embedded System, NVM, SRAM DRAM, Memory Cache, Dynamic memory allocation.

Introduction:

Non-Volatile Memory:

Non-volatile memories retain their contents even when the power is turned off, making them ideal for storing data that must be retrieved after a system power-cycle. Nonvolatile memory is typically used to store processor boot-code, persistent application settings, and FPGA configuration data. Although non-volatile memory has the advantage of retaining data even when power is turned off, it is typically much slower to write to than volatile memory and frequently has more complex writing and erasing procedures. Non-volatile memory is also typically only guaranteed to be erasable a certain number of times before failing. Nonvolatile memories include all forms of flash, EPROM, and EEPROM. For non-volatile storage, most modern embedded systems employ some form of flash memory.

It is a type of memory in which data or information is not lost even when the power is turned off. The most common type of non-volatile memory is ROM (Read Only Memory).

It is not as economical and slow in fetch/store as volatile memory, but it stores a larger volume of data. Non-volatile memory is used to store information that must be kept for an extended period of time. Non-volatile memory has a significant impact on the storage capacity of a system. [1]

SDRAM:

Another type of volatile memory is SDRAM. It is similar to SRAM in that it is dynamic and must be refreshed on a regular basis to maintain its content. SDRAM's dynamic memory cells are much smaller than SRAM's static memory cells. This size difference translates into extremely high-capacity and low-cost memory devices. SDRAM has other very specific interface requirements that, in addition to the refresh requirement, typically necessitate the use of special controller hardware. SDRAM divides its memory space into banks, rows, and columns, as opposed to SRAM, which has a fixed set of address lines.

Switching between banks and rows incurs some overhead, so efficient use of SDRAM necessitates careful access ordering. SDRAM also multiplexes row and column addresses over the same address lines, reducing the number of pins required to implement a given size of SDRAM. Higher speed SDRAM variants such as DDR, DDR2, and DDR3 have strict signal integrity requirements that must be carefully considered during the PROCESS CONTROL BLOCK design. SDRAM devices are one of the most popular types of RAM devices because they are among the least expensive and have the highest capacity. SDRAM is used in the majority of modern embedded systems. The SDRAM controller is a critical component of an SDRAM interface. The SDRAM controller handles all address-multiplexing, refresh, row and bank switching, and row and bank switching tasks, allowing the rest of the system to access SDRAM without understanding its internal architecture. [2-3]

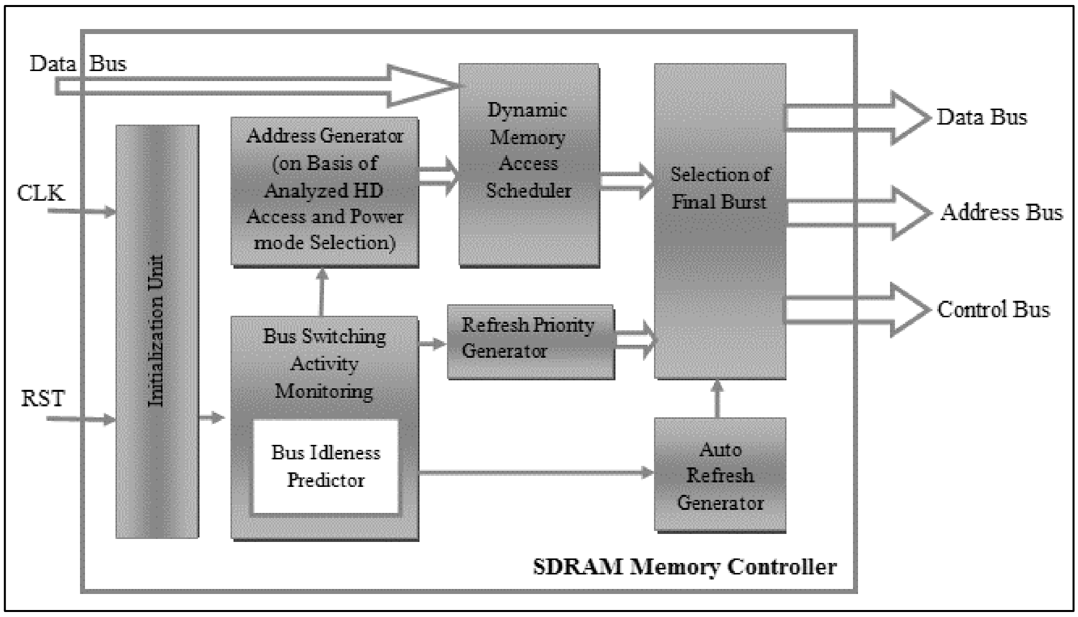
Data Bus
CLK
RST
Initialization Unit
Address Generator (on Basis of Analyzed HD Access and Power mode Selection)
Dynamic Memory Access Scheduler
Selection of Final Burst
Bus Switching Activity Monitoring
Bus Idleness Predictor
Refresh Priority Generator
Auto Refresh Generator
Data Bus
Address Bus
Control Bus
SDRAM Memory Controller

Figure 1: SDRAM Memory Architecture

Dynamic RAM Overview:

Because of the lower cost per bit, larger microcomputer systems use Dynamic RAM (DRAM) rather than Static RAM (SRAM). DRAMs necessitate more complex interface circuitry due to their multiplexed address bus and the need to refresh each memory cell on a regular basis.

A typical DRAM memory is organised as a square array of memory cells with equal numbers of rows and columns. One bit is stored in each memory cell. The bits are addressed by selecting a row with half of the bits (the most significant half) and a column with the other half. Each DRAM memory cell is made up of only a capacitor and a MOSFET switch. As a result, a DRAM memory cell is much smaller than an SRAM cell, which requires at least two gates to implement a flip-flop. [4]

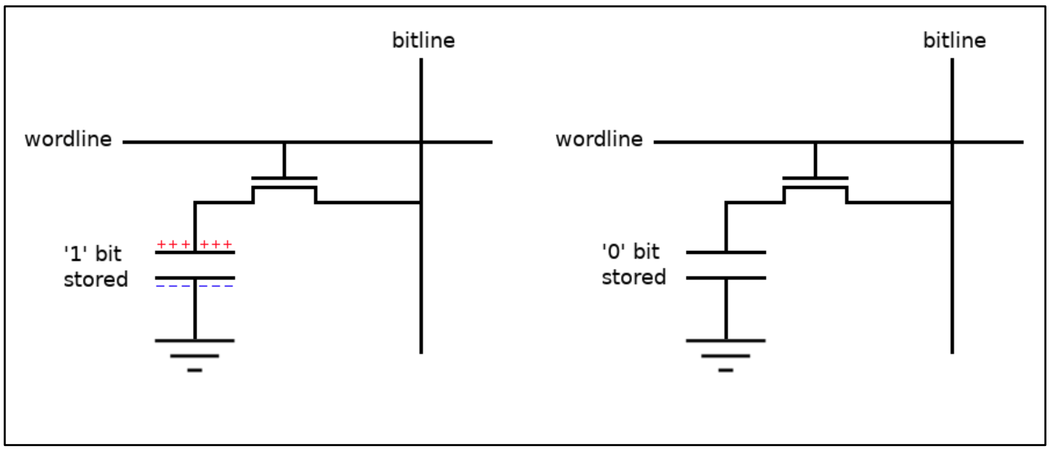

Figure 2: DRAM Inside Structure

The DRAM system will use an architecture similar to that of most previous SRAM systems. One significant distinction is the potential need to manage the refresh function. A second distinction arises from the difficulty of memory size versus IC package size.

NVRAM:

There is a lot of memory in a modern computer system. The majority of it is anachronistically referred to as random access memory (RAM). Because all memory nowadays is random access, the name makes little sense. RAM refers to volatile semiconductor memory that can be written to and read from indefinitely as long as power is applied. This was not always the case. "Core memory" was the most common form of program/data storage in the early days of computers. This was bulky and heavy (not to mention expensive!) by modern standards, but it had a useful feature: it was non-volatile. Power was required to read or write data, but not to keep it. Data would remain unchanged for indefinite periods of time if the core memory was turned off. Dropping or vibrating core memory could corrupt its contents, but this was rarely a cause for concern (except in earthquake zones) because computers were bulky and immobile. [5]

Although RAM is the primary form of working memory in modern computers and most embedded systems, having a quantity of non-volatile RAM (NVRAM) available can be useful. This could be implemented with flash memory or another non-volatile memory technology (like MRAM), or it could be regular RAM with a protected power supply (i.e., a battery). In an embedded system, NVRAM can be used for a variety of purposes, including:

- Programme code and constant data are stored in RAM and are copied into RAM upon startup. Although NVRAM execution is generally an option, the speed (access time) of some NVRAM technologies makes it unappealing.
- Preservation of device configuration parameters between power cycles. Many devices can be configured by the user; this information must be saved somewhere.
- Long-term buffering of acquired data with immunity to power outages. A simple example is the storage of photos in a digital camera.

NVRAM Initialization:

When NVRAM is first powered on, it contains indeterminate data and must be initialised, just like regular RAM. The software must recognise that the NVRAM has been initialised and not overwrite the saved data on subsequent occasions.

The simplest way to accomplish this is to use a signature, which is simply a quickly recognisable sequence of bytes that cannot occur randomly. Of course, this ideal is impossible to achieve because any sequence of bytes, no matter how long, could occur at random. It's just a matter of minimising that possibility while keeping the check quick and simple. If the signature is only four bytes long, it has a 4 billion-to-one chance of occurring at random. That is sufficient for almost any conceivable application. A 32-bit value can also be checked quickly. [6]

NVRAM integrity:

Of course, using a signature does not guarantee the data's integrity. It's a good idea to use a checksum or CRC for error checking, or even a mechanism for data self-correction.

System start-up with NVRAM:

When using NVRAM, the start-up logic must support both signature verification and data integrity checking.

Non-Volatile Embedded System:

The section that follows examines the processor, caches, and main memory to discuss how these components can be rethought using non-volatile memory (NVM) technologies. The positive effects that NVMs can have on an embedded system are investigated, as well as the challenges that come with less beneficial properties and how those can be compensated for. In addition, potential implementations are discussed. This section is only a summary of the research in this area and does not purport to be exhaustive. [7]

1. Non-Volatile Processor (NVP)

All memory components in the processor, from the register file and programme counter (PC) to the buffers, queues, and lists needed for pipelining, are typically implemented in volatile memory technologies. Designing a non-volatile processor (NVP) is a complex task, but using NVM offers the benefit of improved forward progress due to the availability of backups. This feature becomes even more important if the system is powered by a battery charged by energy harvesting. If a power outage occurs, a system with a volatile processor loses its state, as does all data not yet written to an NVM.

2. Design:

As previously stated, the implementation of backups is a critical component when designing an NVP. The location of the backup, the components to be backed up, as well as the timing and frequency of backup operations, must be determined. All components, including the register file, are non-volatile when using only NVMs, allowing them to retain their state during power outages. This design necessitates considerable effort to ensure that the system only moves from one valid state to another. Actual backups would be possible with the addition of a central NVM block. [8]

Review of Literature:

According to Ma et al. [8], processor components are classified as architecture state (register file, PC), microarchitecture state (pipeline latches, reorder buffer, and load/store queue), and performance-enhancing components (e.g., branch history table, branch target buffer). Different backup strategies are developed based on these three categories, with their effectiveness determined by the system's timing constraints. For example, including performance enhancers in the backup significantly improves forward progress because the entire pipeline state is saved and does not need to be recalculated.

The timing of backups is just as important as deciding which components to keep. There are two approaches: backups are performed either on a regular or on-demand basis. Periodic backups, as proposed by Xie et al. [9], can occur at fixed intervals or at statically determined code lines. Their method employs static code analysis to select backup points that store consistent states. They identify load and store instructions to the same address as potential error locations, or rather the code in between those instructions.

Ma et al. [8] propose back-up schemes that are triggered when the power supply falls below a certain threshold. The threshold is set so that the remaining energy barely exceeds the level required to complete the backup routine (On Demand All Back-Up). Alternatively, the remaining energy is sufficient to complete the current PC as well as the entire back-up routine (On Demand Selective Back-Up).

Result and Discussion:

A CPU cache is a cache used by a computer's central processing unit to reduce the average time to access memory. The cache is a smaller, faster memory that stores duplicates of data from frequently accessed main memory locations.

As long as the majority of memory accesses are to cached memory locations, the average latency of memory accesses will be closer to cache latency than to main memory latency. Data is transferred between memory and cache in fixed-size blocks known as cache lines.

A cache entry is created when a cache line is copied from memory into the cache. The cache entry will contain both the copied data and the requested memory location (now known as a tag). When the processor needs to read or write a location in main memory, it first looks in the cache for a corresponding entry. The cache looks for the requested memory location's contents in any cache lines that may contain that address. A cache hit occurs when the processor discovers that the memory location is in the cache. A cache miss occurs if the processor does not find the memory location in the cache. In the event of: When a cache hit occurs, the processor reads or writes the data in the cache line immediately. In the event of a cache miss, the cache creates a new entry and copies data from main memory. The request is then fulfilled using the cache's contents. [10]

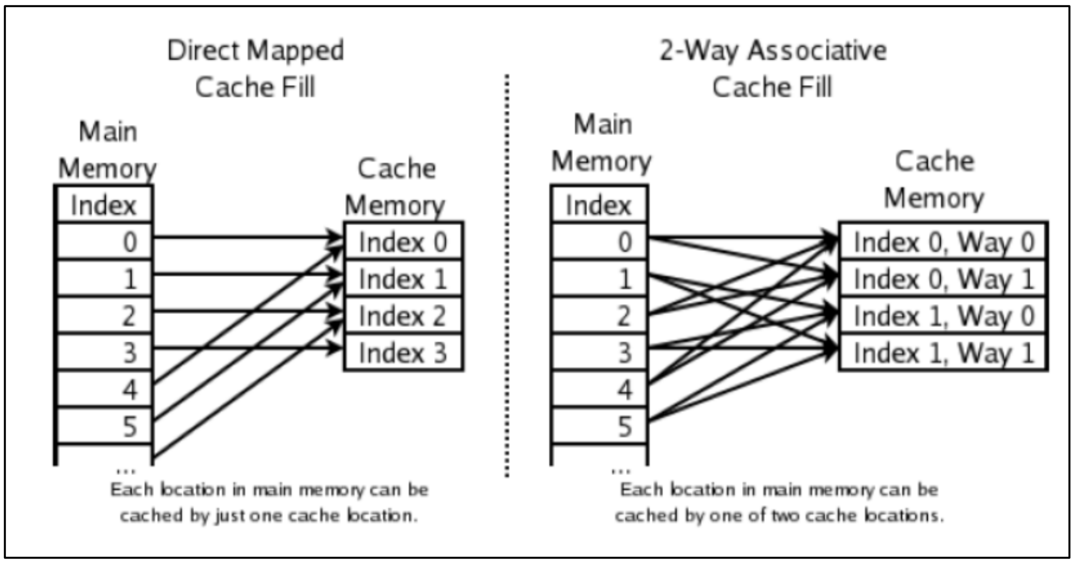

Figure 3: Memory Cache

The replacement policy determines where a copy of a specific main memory entry will go in the cache. The cache is said to be fully associative if the replacement policy is free to choose any entry in the cache to hold the copy.

The cache is direct mapped if each entry in main memory can only go in one place in the cache. Many caches implement an N-way set associative compromise in which each entry in main memory can go to any one of N places in the cache. [11]

One advantage of this scheme is that the cache tags do not have to include the portion of the main memory address implied by the cache memory's index. Because cache tags have fewer bits, they take up less space on the microprocessor chip and can be read and compared more quickly. LRU is also particularly simple because only one bit needs to be stored for each pair.

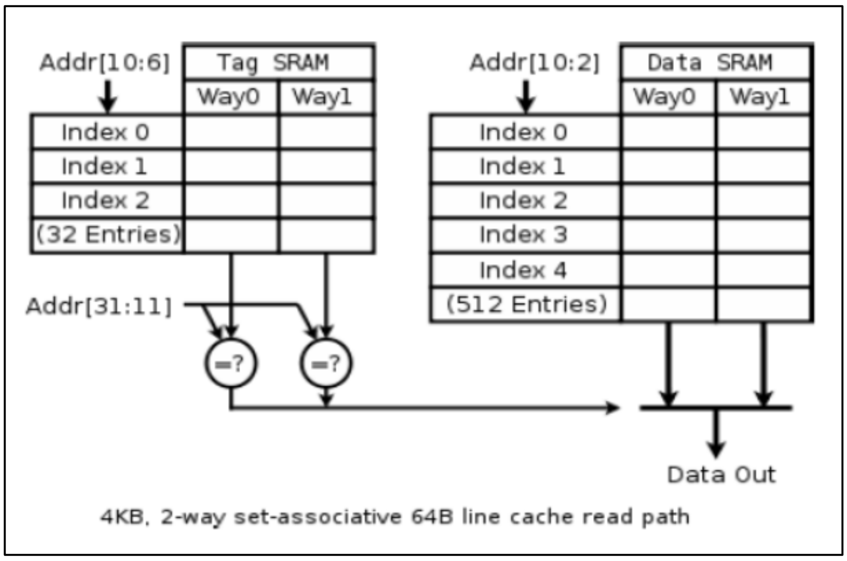

Figure 4: Associative cache

The most common CPU operation that takes more than a single cycle is cache reads. The latency of a level-1 data cache hit has a high impact on programme execution time.

A significant amount of design effort, as well as power and silicon area, is expended to make caches as fast as possible. [12]

Dynamic Memory Allocation:

The act of managing computer memory is known as memory management. Memory management must provide methods for dynamically allocating portions of memory to programmes at their request and freeing it for reuse when no longer required. This is critical to the computer system's operation.

Several methods for improving memory management effectiveness have been developed. Virtual memory systems decouple the memory addresses used by a process from the actual physical addresses, allowing process separation and increasing the effectively available amount of RAM through paging or swapping to secondary storage. The virtual memory manager's performance can have a significant impact on overall system performance. [13-14]

Conclusion:

This paper discussed the current state of research into the use of byte-addressable non-volatile memory technologies and their potential to replace volatile SRAM and DRAM. Using NVRAM in an embedded design is simple, but its functionality must be carefully accommodated, as explained here. The approach of using a global signature and error check is appropriate for a wide range of applications. A separate check on each block of data may be more efficient for very large databases.

References:

1. Katelin Bailey, Luis Ceze, Steven D Gribble, and Henry M Levy. 2011. Operating System Implications of Fast, Cheap, Non-Volatile Memory. In Proceedings of the 13th USENIX Workshop on Hot Topics in Operating Systems (HotOS '11), Vol. 13.
2. Bertrand Le Gal, Emmanuel Casseau, and Sylvain Huet "Dynamic Memory Access Management for High-Performance DSP Applications Using High-Level Synthesis" IEEE TRANSACTIONS ON VLSI SYSTEMS, VOL. 16, Issue NO. 11, NOVEMBER.2008, pp: 1454-1463.
3. Memory Systems: Cache, DRAM, Disk. Bruce Jacob, Spencer W. Ng, and David T. Wang, with contributions by Samuel. ISBN 978-0-12-379751-3. Morgan Kaufmann Publishers, September 2007
4. Hu Hongqi; Sun Jingnan; Xu Jiadong; , "High Efficiency Synchronous DRAM Controller for H.264 HDTV Encoder", 4th IEEE Conference on Industrial Electronics & Applications 2009, pp.2132-2136, 25-27 May 2009.
5. Yongpan Liu, Zewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng-Fan Chang, Sampson John, Yuan Xie, et al. 2015. Ambient Energy Harvesting Nonvolatile Processors: From Circuit to System. In Proceedings of the 52nd Annual Design Automation Conference (DAC '15). 150.
6. Daniel Lohmann, Wolfgang Schroder-Preikschat, and Olaf Spinczyk. 2005. Functional and Non-Functional Properties in a Family of Embedded Operating Systems. In Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '05). 413–420.
7. Kaisheng Ma, Xueqing Li, Shuangchen Li, Yongpan Liu, John Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Nonvolatile Processor Architecture Exploration for Energy-Harvesting Applications. IEEE Micro 35, 5 (2015), 32–40.
8. Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Architecture Exploration for Ambient Energy Harvesting Nonvolatile Processors. In IEEE 21st International Symposium on High Performance Computer Architecture (HPCA '15). 526–537.
9. Mimi Xie, Mengying Zhao, Chen Pan, Jingtong Hu, Yongpan Liu, and Chun Jason Xue. 2015. Fixing the Broken Time Machine: Consistency-Aware Checkpointing for Energy Harvesting Powered Non-Volatile Processor. In Proceedings of the 52Nd Annual Design Automation Conference (DAC '15). 184:1–184:6.
10. Yiran Chen, Weng-Fai Wong, Hai Li, Cheng-Kok Koh, Yaojun Zhang, and Wujie Wen. 2013. On-chip Caches Built on Multilevel Spin-transfer Torque RAM Cells and Its Optimizations. J. Emerg. Technol. Comput. Syst. 9 (2013), 16:1–16:22.
11. Navid Khoshavi and Ronald F Demara. 2018. Read-Tuned STT-RAM and eDRAM Cache Hierarchies for Throughput and Energy Optimization. IEEE Access 6 (2018), 14576–14590
12. Jianxing Wang, Yenni Tim, Weng-Fai Wong, Zhong-Liang Ong, Zhenyu Sun, and Hai Helen Li. 2014. A Coherent Hybrid SRAM and STT-RAM L1 Cache Architecture for Shared Memory Multicores. In Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC '14). 610–615.
13. Zhenyu Sun, Xiuyuan Bi, Hai (Helen) Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. 2011. Multi Retention Level STT-RAM Cache

Designs with a Dynamic Refresh Scheme. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’11). 329–338.

14. Rodrigue Rizk, Dominick Rizk, Ashok Kumar, and Magdy Bayoumi. 2019. Demystifying Emerging Nonvolatile Memory Technologies: Understanding Advantages, Challenges, Trends, and Novel Applications. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS ’19). 1–5.