Memory Controllers for High-Performance and Real-Time MPSoCs

Requirements, Architectures, and Future Trends

Benny Akesson[1], Po-Chun Huang[2], Fabien Clermidy[3], Denis Dutoit[3], Kees Goossens[1], Yuan-Hao Chang[4], Tei-Wei Kuo[2,4], Pascal Vivet[3], and Drew Wingard[5]

[1] Eindhoven University of Technology, the Netherlands
[2] National Taiwan University, Taiwan
[3] CEA LETI, France
[4] Academia Sinica, Taiwan
[5] Sonics Inc., USA

ABSTRACT

Designing memory controllers for complex real-time and high-performance multi-processor systems-on-chip is challenging, since sufficient capacity and (real-time) performance must be provided in a reliable manner at low cost and with low power consumption. This special session contains four presentations that describe these challenges and proposed solutions for DRAM and flash memory controllers, respectively. The first presentation discusses performance and reliability issues in flash memories, while the second identifies challenges in providing DRAM access to memory clients with mixed time-criticality. The third presentation proposes an integrated approach to optimize cost and performance of the DRAM subsystem, and the last one describes how wide DRAM interfaces enabled by 3D technology improve DRAM performance and reduces power.

Categories and Subject Descriptors: B.8.2 [Performance and reliability]: Performance Analysis and Design Aids

General Terms: Design, Performance, Reliability, Verification

1. INTRODUCTION

Memories are key components in any modern computer system, and the performance of these devices is critical in Multi-Processor Systems-on-Chips (MPSoCs) in both the real-time and high-performance domains. Contemporary systems have complex memory hierarchies with diverse types of volatile and non-volatile memories, such as DRAM and flash. It is the daunting task of the memory controllers in the systems to manage these devices such that sufficient capacity and (real-time) performance are provided to the memory clients in a reliable manner, while limiting cost and power consumption. To address this challenge, memory controllers have advanced architectures, policies, and scheduling algorithms that continue to evolve as we move closer to the memory and power walls.

This special session comprises four presentations about DRAM and flash memory controllers for real-time and high-performance MPSoCs, covering technology trends, requirements, and architectures. The remainder of this paper is structured in four sections, each corresponding to one of the presentations. Section 2 starts by presenting trends and challenges in consumer flash-memory devices. The architecture of a typical flash controller is presented along with potential solutions to improve reliability and performance in consumer products. In particular, technologies for address translation, striping, real-time performance, and wear-leveling are discussed. Section 3 then presents the requirements of mixed time-criticality systems that have a mix of memory clients with firm, soft, and no real-time requirements. These requirements are complex and conflicting, and cannot be efficiently satisfied by current memory controllers. Future research challenges are identified to design memory controllers suitable for these systems. Section 4 explains how to optimize the cost and performance of a DRAM subsystem, where the memory clients have conflicting quality-of-service (QoS) requirements. An integrated approach is proposed that manages memory traffic from the processor interface, through the interconnect, to the DRAM controller. Lastly, Section 5 describes how the transition from a single off-chip DRAM to DRAM stacked on top of logic using 3D technology improves performance and reduces power consumption, hopefully enabling us to take a step back from the impending memory and power walls.

CODES+ISSS'11, October 9–14, 2011, Taipei, Taiwan.

2. CHALLENGES AND SOLUTIONS FOR CONSUMER FLASH-MEMORY DEVICES

(Huang, Chang, and Kuo)

Flash memories have been widely adopted in various consumer and even enterprise products in recent years because of its non-volatile, shock-resistant, and power-economic characteristics. With the density advancement of semiconductors, high-density and low-price multi-level-cell (MLC) flash chips have replaced the role of single-level-cell (SLC) flash chips in many applications and dominated the market of consumer flash-memory devices, such as solid-state drives (SSDs) and flash cards. However, compared to SLC flash memory, MLC flash memory has lower read/write performance, higher bit error rate, and lower endurance. This situation is exacerbated, as the development trend increases the cell density of MLC flash memory to further reduce the

unit cost. In addition, the fast growing capacity of both MLC and SLC flash memory also introduces new challenges to the flash management designs with respect to the scalability issue. As a result, retaining the performance and reliability of consumer flash-memory devices will become more challenging in the near future.

In this presentation, the characteristics, system architecture, and development trends of consumer flash-memory devices are discussed in Section 2.1. Then, performance/reliability issues and their solutions are investigated and discussed in Section 2.2. Section 2.3 presents conclusions.

2.1 Architecture and design issues

A flash memory chip usually contains one or more sub-chips, each of which is composed of planes. A plane consists of blocks, and each block includes a fixed number of pages. A block is the unit for erase operations and a page is the basic unit for read/write operations, where a page could not be overwritten unless its residing block is erased. Due to cost considerations, MLC flash memory has surpassed the market share of SLC flash memory and has become the major storage media in consumer devices, where a cell of SLC flash memory stores 1-bit data and an MLC$_{\times n}$ flash memory cell stores n-bit data. As shown in Table 1, the low-price MLC flash memory (e.g. triple-level-cell (MLC$_{\times 3}$) flash memory) has lower access performance and reliability, compared to its counterpart SLC flash memory. In addition, it also introduces two new write constraints. That is, a page cannot be partially programmed and pages in a block must be written sequentially from the first one. In order to obey the write constraints, the write performance of MLC-based flash-memory devices is thus exacerbated.

Table 1: Comparisons of flash chips. [37–39]

Cell type	SLC	MLC$_{\times 2}$	MLC$_{\times 3}$
Price (USD/32 Gb)	25.1	4.08	3.87
Page size (KB)	4	8	8
Block size (pages)	128	256	384
Page read time (μs)	35	75	100
Page write time (ms)	0.3	1.3	2.5
Block erase time (ms)	0.7	3.8	3
Endurance (P/E cycles)	$\geq$ 10000	$\approx$ 3000	$\leq$ 1000

As shown in Figure 1, a consumer flash-memory device usually consists of a controller, ROM for the storage of the firmware, and RAM to store the address translation and management information. It might consist of one or more flash chips, and could be classified into two major types. One is the SSD that is designed for the replacement of hard drives and is usually consists of several chips (e.g. 8 chips). The other is the flash card (e.g. USB flash drive and SD card) that is commonly used as storage for user's data and is usually composed of one or two chips. Both types have the characteristic of fast growing capacity to challenge the performance of flash management designs, in terms of space scalability. Since SSDs usually have huge capacity and require high access performance, chips of an SSD are usually distributed to multiple channels, and each channel can be accessed independently to increase the degree of parallelism when accessing flash chips. In contrast, flash cards usually require low unit cost. They usually include a controller with limited computing power, a small size of RAM, and low-cost MLC flash chips. Based on the above observations, the technical issue is how to improve the inherent performance and reliability issues of MLC flash memory, considering the characteristics of SSDs and flash cards.

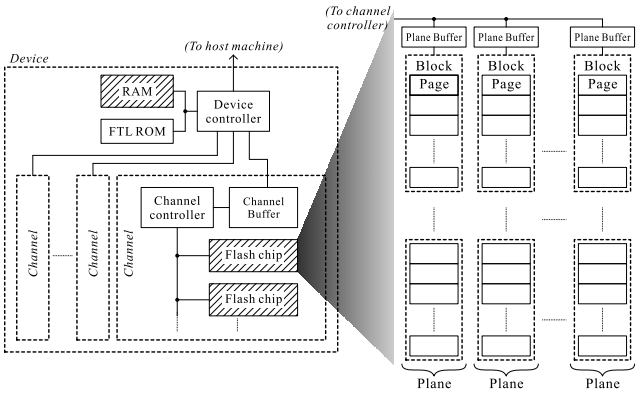


Figure 1: A flash-memory device architecture.

2.2 Challenges and solutions

2.2.1 Performance issues

Performance is an important factor of a storage device. There are three key factors that affect the performance of a flash-memory device. These are the address translation, garbage collection, and the access time to the flash chips.

The address translation to translate the logical block addresses (LBAs) to their corresponding physical block addresses (PBAs) is required because *out-place update* is adopted to update data to a free page instead of writing to the original place. The rationale behind this is to prevent erasing blocks on every data update to improve the write performance. In order to perform the address translation, flash translation layer (FTL) is first proposed by Intel to manage the translation information at the page level [26]. As the capacity of flash-memory devices grows, the size of the address translation table of FTL is too large to fit in the limited RAM space of flash-memory devices. In order to solve this issue, BL is proposed to reduce the size of the mapping information by adopting a block-level address translation mechanism that is to map each LBA to a physical block and an offset in the block [13]. However, BL could cause excess live-data copying, because a write to update existing data requires a free block to store the updated data and the valid data in the same block of the existing data. In order to compromise between performance and RAM space, various hybrid mapping mechanisms, such as NFTL and FAST, were proposed to manage the translation information at the block level, while the updated data are temporarily managed with the page-level translation mechanism [4,33]. Due to the fast growing capacity of flash-memory devices, the limited RAM space could eventually not accommodate the block-level address translation information, so that many new management strategies (e.g., DFTL and CBMS) [14,21] are proposed to write the translation information to flash memory and some others even try to design address translation mechanisms with scalable (or adaptive) translation granularities [52].

The performance of a flash-memory device is also seriously affected by the efficiency of its garbage collection design. The garbage collection is needed when there is not enough free space in the flash memory. This is because the out-place update writes updated data to free pages (called *live pages*) and, therefore, invalidate pages (referred to as *dead pages*) of

the existing data. When the garbage collection is activated, it needs to select a victim block, copy live pages of the victim block to free pages, and then erase the victim block. In order to reduce the overheads on the selection of victim blocks, a greedy policy is usually adopted to scan the blocks and select the one whose dead page count and/or live page count exceed a predetermined threshold as the victim block. To reduce the overhead of garbage collection, many special policies are designed to either minimize the live-page copying (or block erases) or reduce the time on selecting a victim block [17, 32]. In particular, the over-provisioning method is proposed to reduce the overhead of live-page copies by preserving an over-provisioning area (i.e. preserved area) for the free space allocation to reduce the frequency of garbage collection activities. In addition, if the response time of the device is concerned, the phase garbage collection could be adopted to improve the response time of the device by splitting the live-page copying process into several phases, each of which could be interrupted by a read/write request (from the host) before its execution [18]. As for the storage devices in real-time systems, the real-time garbage collection [12] is designed to recycle a block in each time period with guaranteeing enough free pages for each real-time task to prevent any real-time task from being blocked over the allowed slack time and missing its deadline.

Because the access time of MLC flash chips keeps increasing and the development trend of SSDs is to adopt multiple MLC flash chips on more channels, maximizing the degree of parallelism on accessing flash chips has become an important design issue to improve the performance of SSDs. Thus, the adaptive striping [11] proposes to distribute data of each write request to each chip/channel evenly, to improve the write performance of the flash-memory device. However, adaptive striping requires a page-level address translation mechanism to support its striping at the page level. In order to support the access to multiple chips/channels with smaller RAM space for the address translation information, a set-based mapping [14] is designed to configure pages at the same offset of every chips a read/write unit as well as the address translation unit, so that data could be read from or written to a page set simultaneously to maximize the parallelism degree with the size of the mapping information reduced.

2.2.2 Reliability issues

Due to the high error rate of MLC flash memory and its popularity in various products with high update frequencies, research on the reliability of consumer flash-memory devices, as well as the wear-leveling issue, has received a lot of attention in recent years. Note that flash chips are tested before shipment, and various efficient testing algorithms, such as march-based algorithms, were designed to detect various flash-memory faults, e.g. stuck-at-fault, address decoder fault, and word-line/bit-line disturb fault [31].

To correct data errors after flash chips are shipped out, MLC flash chips support a large spare area in each page to store the redundancy generated by error correction codes (ECCs) that could correct more error bits so that the hardware ECC, e.g. BCH and RS [7, 8], might need to be redesigned to enhance the capability in bit-error correction. On the other hand, many software methods were proposed to improve the reliability of flash-memory devices. One intuitive approach is to generate the parity-check (or XOR) information over pages of the same block and store the parity-check information in the last one or two pages of the same block to "commit" it [14]. When the data error of a page fails to be corrected by the ECC redundancy stored in its spare area, the parity-check information with data of other pages in the same block are loaded to recover the error page data.

Another direction is to adopt RAID technology to provide a better capability on the correction of burst errors. However, the adoption of RAID technology could be complicated and introduce significant management overhead, due to the need of out-place updates and the read-modify-write operation of the RAID technology to update partial data of a stripe. To solve this problem without sacrificing the data reliability, a scalable striping methodology for SSDs is designed to stripe data to different numbers of channels/chips according to the size of the written data, to reduce the risk of updating partial data of stripes, where a stripe of a different size also includes a different error correction code to reduce the space overheads. Different from the hardware ECCs and the RAID technologies, a two-version strategy is proposed for flash file systems to maintain two correct page versions for each chunk of a user data file, so that the file could be recovered to a consistent version after any one chunk of the file is corrupted and fails to be corrected by existing ECCs or RAID technologies [25].

The reliability of a flash chip could be indirectly improved by the adoption of wear-leveling, which distributes block erases evenly over a flash memory to prevent any block from wearing out, since the error rate of a block increases as the block endures more erases. Thus *dynamic wear-leveling* was proposed to achieve wear-leveling by moving hot data (i.e. the frequently updated data) around to prevent them from staying in any block for a long period of time [11]. Although dynamic wear-leveling does have great improvement on wear-leveling, the endurance improvement is constrained because most data are cold (i.e. the infrequently updated data) and likely to stay intact, regardless of how updates of hot data wear out other blocks. In addition, the development trend of MLC flash chips could endure fewer and fewer erases over each block. Thus *static wear-leveling* is proposed to ultimately prolong the lifetime of MLC flash chips by moving both the hot and cold data around with limited extra block erases. One intuitive solution is to keep track of the erase count of each block with some well-designed data structures (e.g. multiple lists) so as to distribute block erases evenly [41]. In order to reduce the required RAM space to track the block erases, some approximate solutions are proposed to let the distribution of block erases achieve a certain level of evenness by maintaining the ratio of the number of block erases to the number of blocks being erased within a given time period of time [13].

2.3 Conclusions

The objective of this presentation is to discuss the development trends of consumer flash-memory devices, as well as their challenging issues and solutions. In particular, the key factors of the performance issue, such as address translation and garbage collection, are discussed along with their solutions. For the reliability issue, various error correction technologies are presented to improve the reliability of flash-memory devices. The wear-leveling is also discussed for the reliability enhancement of flash-memory devices. With the rapid dropping cost and fast growing capacity of MLC flash memory, it is clear that flash-memory devices is an excellent choice for the storage of consumer electronics, yet new challenging issues will also keep coming up when the flash-memory devices are adopted in new application scenarios.

3. SDRAM CONTROLLERS FOR MIXED TIME-CRITICALITY SYSTEMS

(Akesson and Goossens)

Complex contemporary systems are implemented as heterogeneous MPSoCs with distributed shared memory hierarchies to strike a good balance between performance, cost, power consumption and flexibility [6, 27, 30]. These systems concurrently execute multiple *mixed time-criticality* applications, which means they have a mix between firm, soft, and no *real-time requirements*. The real-time requirements of the applications are reflected in the requirements of the memory clients (IP blocks) in the SDRAM controller. However, although there are suitable SDRAM controller designs for firm and soft/no real-time requirements, respectively, there are no good solutions for systems with all types.

This presentation starts in Section 3.1 by introducing the architecture and timing behavior of SDRAM memories. Section 3.2 then explains the requirements of memory clients in mixed time-criticality systems. Sections 3.3 and 3.4 proceed by reviewing the state of the art in SDRAM controllers for firm and soft/no real-time memory controllers, respectively. Section 3.5 then explains why these controllers are unable to efficiently satisfy the requirements of mixed time-criticality systems, and identifies research challenges that must be addressed to evolve towards a mixed time-criticality SDRAM controller design. Lastly, conclusions are presented in Section 3.6.

3.1 SDRAM overview

An SDRAM memory comprises a number of banks, each containing a memory array with a matrix-like structure, consisting of rows and columns. Each bank has a row buffer that can hold one open row at a time, and read and write operations are only allowed to the open row. Once a row has been opened, *read* and *write* bursts can be issued to access the columns in the row buffer. These bursts have a programmable *burst length* (BL) of either 4 or 8 words for DDR2/DDR3 SDRAM that are transferred from/to the memory with a data rate of two words per clock cycle.

The SDRAM architecture makes the response time of requests and the provided bandwidth highly variable for three reasons: 1) a request targeting an open row can be served immediately (locality), while it otherwise needs the current row to be closed and the required row to be opened (no locality), 2) the data bus is bi-directional and requires several cycles to switch from read to write and vice versa, and 3) to prevent data loss, the memory must occasionally be refreshed before executing the next request, which results in several cycles without data transfer. The impact of these three factors may cause the execution time of an SDRAM burst to vary by an order of magnitude from a few clock cycles to a few tens of cycles.

The bandwidth to and from a memory ideally corresponds to the product of the width of the memory interface, the clock frequency of the memory, and the data rate. This is referred to as the *peak bandwidth* of the memory. However, for the previously mentioned reasons, the peak bandwidth of SDRAMs cannot be fully utilized. This is captured by the concept of *memory efficiency*, which is the percentage of clock cycles with useful data on the data bus [2]. The product of the memory efficiency and the peak bandwidth determines the *net bandwidth*, which is the bandwidth that is useful to the memory clients after considering all types of overhead. SDRAM bandwidth is a scarce and expensive resource due pin and power constraints on the chip. Memory controllers hence try to maximize memory efficiency to reduce cost with common targets being in the range of 70-90%, depending on the application.

3.2 Memory client requirements

Applications may have different types of real-time requirements, depending on the nature of the processing and the degree of pipelining in the processing cores. These application requirements are reflected in the requirements of their corresponding memory clients. For memory clients of throughput-driven applications, such as streaming media decoders, bandwidth requirements are most important. Conversely, clients of latency-driven applications, e.g. in the control domain, primarily have response time requirements. The requirements exist in three different classes, depending on the criticality of the application. Applications, such as a Software-Defined Radio [40], have *firm real-time requirements* (FRT). Failure to satisfy the requirements of their memory clients is highly undesirable and may result in failure to comply with a given standard, or even violate the functional correctness of the MPSoC [49]. Other applications, such as media decoders, have *soft real-time requirements* (SRT). Missing a soft deadline results in quality degradation of the application output, such as causing visual artifacts in decoded video. Although this is perceived as annoying by the user, it may be acceptable as long as it does not occur too frequently. There are also applications with *no real-time requirements* (NRT), such as a graphical user interface. Their memory clients do not have well-defined requirements, but must still be served fast enough for the application to be perceived as responsive by the user.

A key challenge with respect to memory client requirements is that many modern MPSoCs are *mixed time-criticality systems*, which implies that the memory clients have a mix of FRT, SRT, and NRT requirements [49, 50]. This is a problem as there are suitable memory controller designs for FRT and SRT/NRT requirements, respectively, but no good solutions for mixes between these types.

3.3 Firm real-time controllers

This section discusses memory controllers suitable for memory clients with FRT requirements, whose bandwidth and/or response time requirements must be satisfied even in the worst-case scenario. The goals of these memory controllers are typically to maximize the guaranteed net bandwidth to all memory clients and/or minimize their guaranteed response times. This is challenging, as both net bandwidth and response times are highly variable and difficult to bound tightly.

A problem for FRT memory controllers is that they are typically unable to exploit locality, since locality is difficult to guarantee even for the memory accesses of a single application, and more or less impossible for interleaved memory accesses from a set of concurrently executing applications. This results in bounds on net bandwidth and response times that are far away from their average values, making it expensive to provide FRT performance guarantees for general applications. To minimize this cost, some FRT memory controllers employ a *close-page policy*, which means that rows are closed immediately after a burst [43] or set of bursts [1] is finished to minimize the overhead of opening another row in the same bank. However, this policy reduces the best-case memory efficiency, since rows are opened and closed for every access. As an example, the controller in [1] has a best-case memory efficiency of only 80% for a 16-bit DDR3-800 memory [28] that interleaves 64 B requests over four banks with $BL = 8$. This case happens when all requests are reads.

Most SDRAM controllers can be classified as either statically or dynamically scheduled, depending on if the sequence of SDRAM commands sent to the memory is determined at design time or at run time. Statically scheduled controllers, such as [5], execute precomputed schedules of SDRAM commands that have been computed at design time. These controllers are suitable for memory clients with FRT requirements, since response times and net bandwidth can be bounded at design time by analyzing the schedule. The main drawback of these controllers is that their applicability is limited to single applications whose memory behavior can be exactly specified at design time, which is typically not the case in MPSoCs. However, note that such an application-specific controller can exploit locality, since knowledge about the exact memory access trace enables rows to be opened and closed in an optimal manner, resulting in a highly efficient, although very restrictive solution.

Dynamically scheduled memory controllers, on the other hand, schedule SDRAM commands at run time based on available requests. The challenge with dynamically scheduled FRT controllers, such as [43], is to analyze the behavior of the command scheduler for requests with arbitrary and variable sizes, which is required to bound the net bandwidth and response times. This is why this controller and other FRT controllers assume requests to have fixed size. To guarantee high memory efficiency, the fixed request size is furthermore required to be large enough to exploit bank-level parallelism by interleaving requests across multiple banks. This requires sizes in the range of 64-256 B, depending on the number of banks and the width of the memory interface.

A hybrid memory controller is proposed in [1] that combines aspects of both statically and dynamically scheduled approaches. The hybrid concept is based on *predictable memory patterns*, which are statically computed sequences (sub-schedules) of SDRAM commands. These patterns are dynamically scheduled at run time, based on the incoming requests. The memory patterns exist in five flavors: 1) read pattern, 2) write pattern, 3) read/write switching pattern, 4) write/read switching pattern, and 5) refresh pattern. The benefits of this solution is that it lifts the abstraction level from individual highly inter-dependent SDRAM commands to patterns, which are sets of commands, that are relatively independent. This makes both memory scheduling and performance analysis easier. Similarly to [43], the hybrid controller requires large requests that are interleaved over the memory banks to provide high bounds on memory efficiency. As an example, 63% memory efficiency is guaranteed in the worst case for a 16-bit DDR3-800 memory interleaving 64 B requests over four banks with $BL = 8$. This case happens when all requests are writes.

3.4 Soft/no real-time controllers

The same memory controllers are typically used for both SRT and NRT requirements. They are dynamically scheduled controllers that target maximizing net bandwidth to optimize cost by accommodating more memory clients with a given memory, and minimize response times to satisfy requirements of latency-driven memory clients. Note that this type of memory controller considers *average* efficiency and response times rather than the worst-case counterparts. This is because SRT requirements are typically verified by extensive simulation instead of using the formal analysis techniques employed for FRT requirements. It is hence sufficient to assert that application-level deadlines, which may be at the granularity of thousands of memory requests, are satisfied with high probability, rather than providing absolute guarantees at the level of individual requests [49].

Although this class of controllers can have a variety of policies for opening and closing rows [47], open-page policies are common, since locality in the memory traffic does not have to be guaranteed. There only has to be enough locality to offset the penalty of row misses with high probability. The average memory efficiency of an SRT/NRT controller is hence typically much higher than for their FRT counterparts. The best-case memory efficiency for these controllers happens if all requests are either reads or writes to the same row in a bank and is as high as 98%, the only losses being attributed to mandatory refresh operations.

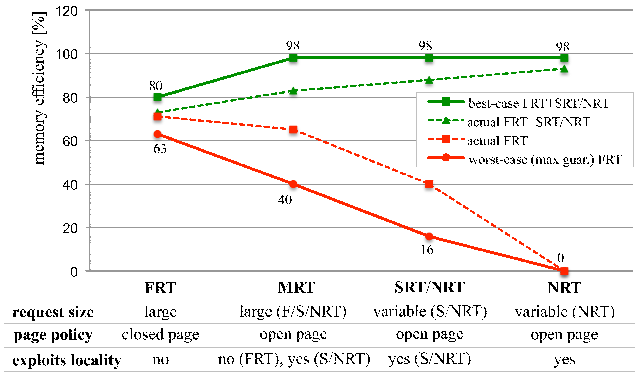
A benefit of SRT/NRT controllers is that they support any type of memory traffic. They are hence very general components, applicable to many systems. The flexibility of these controllers stems from that they are not concerned about formal analysis and providing bounds on memory efficiency, and dynamically schedule memory accesses at the level of single SDRAM bursts, rather than as sets of interleaved bursts. Scheduling at this finer level of granularity also reduces the average response times for high-priority clients by reducing blocking effects from clients with lower priorities. However, since no bank-parallelism is guaranteed between consecutive bursts, these memory controllers are unable to guarantee high memory efficiency in the worst case. For example, a typical controller for a 16-bit DDR3-800 memory with $BL = 8$ only guarantees 16% efficiency for in case all requests are writes and there is a row miss for every burst. This bound is determined by the minimum time between opening two pages in the same bank (tRC). It is hence derived from the memory specification rather than the specifics of the memory controller, and applies by default to many SRT/NRT memory controllers.

SRT/NRT memory controllers often feature sophisticated mechanisms to improve the average memory efficiency or reduce average response times. Examples that improve memory efficiency involve preference for requests that target open rows in the memory banks [34, 36, 42, 48, 51], or that fit with the current direction (read/write) of the data bus [9, 22, 34, 36, 51]. Example mechanisms that reduce average response times of applications prefer reads over writes [48], which is beneficial if reads are blocking while writes are posted, or let high-priority memory clients preempt lower priority clients [34]. Although these mechanisms help SRT/NRT memory controllers fulfill their goals, the interactions between these mechanisms and dynamic command schedulers are complex, making it difficult to derive useful bounds on response times, and even to show that the 16% bound on memory efficiency applies.

3.5 Mixed time-criticality controllers

Mixed time-criticality systems concurrently execute a set of applications with a mix of FRT, SRT, and NRT requirements. Currently, there are no memory controllers that deal well with a set of clients that have conflicting FRT and SRT/NRT requirements. *Mixed real-time* (MRT) controllers are likely to evolve from existing FRT or SRT/NRT controller designs. Extending current FRT controllers to include SRT/NRT requirements requires solving the following open issues:

1) FRT controllers maximize the guaranteed net bandwidth and minimize the maximum response times by using close-page policies, since locality cannot be assumed. However, an MRT controller must guarantee *just enough* net bandwidth (and response times) to satisfy FRT requirements and then *maximize the average* net bandwidth and minimize the average response times for the SRT/NRT clients. It hence makes sense to move from the traditional close-page policy



	FRT	MRT	SRT/NRT	NRT
request size	large	large (F/S/NRT)	variable (S/NRT)	variable (NRT)
page policy	closed page	open page	open page	open page
exploits locality	no	no (FRT), yes (S/NRT)	yes (S/NRT)	yes

Figure 2: Best-case and worst-case memory efficiency for different types of memory controllers.

(FRT in Figure 2) that cannot exploit locality to a *predictable open-page policy* that does use locality to improve the average memory efficiency and reduce average response times for SRT/NRT applications. For example, by introducing hit and miss patterns to the controller in [1] for a 16-bit DDR3-800 with $BL = 8$ and 64 B requests, the best-case efficiency (for all traffic) can increase from 80% to 98%, although the guaranteed memory efficiency (and hence net FTR bandwidth) decreases from 63% to around 40%.

2) FRT applications must have well-specified behaviors, such as worst-case execution times, memory access patterns, and request sizes to enable formal analysis, but SRT and NRT applications typically do not. An MRT controller must hence guarantee that FRT clients receive their requested service in a robust manner, regardless of the behaviors of other clients. This can be achieved by: a) chopping and rounding all requests to fixed-length atomic service units, e.g. as done by the atomizer block in [1], and b) by scheduling the atomic service units using a predictable arbiter, such as [3].

3) Current FRT controllers either have assumptions on or restrict client traffic. This must be extended with support for variable request sizes of dynamic SRT/NRT applications that may not have well-specified behaviors. This involves both generalization of the memory controllers themselves and of their corresponding performance analyses.

4) Support for multiple use-cases is required to support concurrently executing FRT applications that may start or stop dynamically at run time. This requires partial reconfiguration for applications that change, but without disrupting FRT service of persistent applications.

5) FRT power management strategies are required to reduce system power consumption, which is identified as a grand challenge of the coming decade [27].

Conversely, evolving existing SRT controllers to MRT controllers requires deriving bounds on net bandwidth and response times of requests. This requires the following challenges to be addressed:

1) Features, such as reordering, read/write grouping, and preemption, must be restricted to make controllers less dynamic. This is necessary to allow analysis of these features in terms of their worst-case impact on FRT clients. Unfortunately, it may also reduce the average memory efficiency.

2) Even though SRT/NRT controllers support any type of traffic, this results in very pessimistic worst-case memory efficiency. By restricting and increasing the scheduling granularity beyond a single burst, it may be possible to improve on the 16% memory efficiency limiting many controllers today and reach the 40% we project for MRT controllers.

The worst-case efficiency in Figure 2 is indicative of the maximum guaranteed FRT bandwidth. The actual bandwidth required to offer it, shown by the (imagined) actual FRT dashed line, depends on the locality of the traffic. A close-page policy destroys all locality, and the minimum and maximum actual bandwidths are 63-80%, respectively. An open-page SRT/NRT controller exploits all locality. However, in the worst-case with small bursts and no locality, it uses 98% of the actual total bandwidth to offer a maximum guaranteed (FRT) efficiency of 16%. A promising direction for MRT controllers is to allow rows to remain open to improve the best-case total (FRT + SRT/NRT) bandwidth, while still guaranteeing sufficient FRT bandwidth.

3.6 Conclusions

SDRAM memory clients in complex MPSoCs have mixed time-criticality, meaning that they have a mix of firm, soft and no real-time requirements. Although there are controllers for firm or soft/no real-time requirements, there are no good solutions for mixes between these.

This presentation discusses the requirements of mixed time-criticality systems and identifies research challenges to evolve existing memory controller to satisfy them. We conclude that firm real-time controllers must: 1) transition to predictable open-page policies that exploit locality, trading worst-case memory efficiency and response times for an improvement in the average-case counterparts, 2) guarantee a minimum bandwidth and maximum response times to FRT clients, regardless of the behaviors of other clients. 3) relax assumptions and restrictions on memory traffic by generalizing both controllers and performance analysis, 4) support starting and stopping applications at run time, while providing undisrupted real-time service to remaining applications, and 5) support predictable power management strategies to reduce power consumption. Conversely, an evolution of soft/no real-time controllers requires: 1) restrictions on the use of dynamic performance enhancing features to enable bounds on memory efficiency and response times to be derived, and 2) restricting and increasing the scheduling granularity to guarantee sufficient memory efficiency for FRT memory clients.

4. HIGH-PERFORMANCE MEMORY SUBSYSTEMS FOR CONSUMER MULTI-CORE SOCS

(Wingard)

Consumers have demonstrated an almost insatiable desire for devices that can produce, consume and communicate ever higher quantities and qualities of media content. This demand drives semiconductor providers to design MPSoC devices that leverage the superior performance, area and power characteristics of heterogeneous multi-core architectures. The biggest challenge in these architectures is aggressively optimizing the cost and performance of the off-chip DRAM system, because the desired bandwidths grow faster than the underlying DRAM technology and consumer price points demand the lowest cost total system. In this presentation, we show that optimizing the DRAM subsystem performance requires an integrated approach that manages the memory traffic from the interfaces of the processors, through the interconnect network, to the DRAM scheduler and controller. We describe methods for balancing conflicting QoS requirements that exploit the increasing parallelism both among the heterogeneous cores and in the DRAMs themselves.

5. 3D TECHNOLOGIES: SOME PERSPECTIVES FOR MEMORY INTERCONNECT AND CONTROLLER

(Dutoit, Clermidy, and Vivet)

Due to increasing flexibility, performance, and power consumption constraints, multi-core architectures have to face many issues. MPSoC communication with a fast high-storage memory (DRAM or SDRAM) has always been a challenge for multi-core architectures. Such memory is off-chip due to density versus price reasons. High resolution TV, image processing, and augmented reality are typical examples of applications that require drastically increasing bandwidth to the working memory. Fast links have emerged for solving this issue as well as their low-power counterparts, such as DDR and LPDDR standards. However, these standards have extrinsic limitations when the bandwidth must be further increased, mainly due to high frequency in a noisy (variable) environment. Reducing this frequency is possible if the bus width can be enlarged. Current MPSoCs cannot afford such a solution due to limitations on the number of pads.

This presentation explains why 3D technologies using Through Silicon Vias (TSV) provide a unique opportunity to propose new connection schemes and interfaces based on a wide data bus between the MPSoC and the memory. First, a brief overview of 3D TSV technology choices is presented in Section 5.1. Section 5.2 then explains how the wider and slower memory interfaces enabled by 3D technology improve bandwidth and reduce power compared to existing memory interfaces. Integration of wide interfaces and its impact on multi-core architectures is then discussed in Section 5.3, before conclusions are drawn in Section 5.4.

5.1 3D TSV technology choices

In 3D stacking technologies, TSVs are the basic building elements, providing connections between the different stacked dies. The idea is very simple and consists of making a hole in the silicon, which will be filled in another process step. However, multiple technological options can be investigated. In this section, we present a short overview of various 3D technologies [46].

On a top-level fabrication process, three main techniques can be used to stack integrated circuits: Wafer-to-Wafer (W2W), Die-to-Wafer (D2W) and Die-to-Die (D2D). In W2W, two complete wafers are stacked together and dies are extracted after assembly. This solution provides a high-throughput process, although with the severe constraint that the dies of the different wafers must have the same dimensions. In the D2W technique, a die from one wafer is picked and placed on the top of another die integrated in a wafer. This solution allows testing dies individually before 3D assembly. This is commonly called the Known-Good-Die assembly process (KGD), which provides a better yield compared to the W2W technique. Moreover, it removes the constraint of identical dimensions for the two dies and enables heterogeneous technologies (distinct CMOS nodes, analogue, memories, MEMs...) to be used. D2D is the last solution, more costly in terms of fabrication, but mandatory if the assembly step is done by the manufacturer of the lower size die or by an external assembler.

Three principal manufacturing steps are usually used for the 3D process: via drilling and filling, wafer thinning and backside metallization. These three steps may be processed either together or separately. Depending on the position of these steps in the global manufacturing process, three technology options are possible: TSVs first, TSV middle, or TSV last. TSV are referred to as "firsts" if they are processed before the front-end CMOS steps (transistors masks) [23]. The so-called "Via Middle" TSVs [44] are processed after the front-end transistor process, but before metal layer fabrication. Wafer thinning and backside metallization can be processed during the assembly steps. Finally, "Via Last" [15] (or post backend TSVs) are realized after the metallization stage. In such a case, TSVs can be processed by packaging fabrication lines.

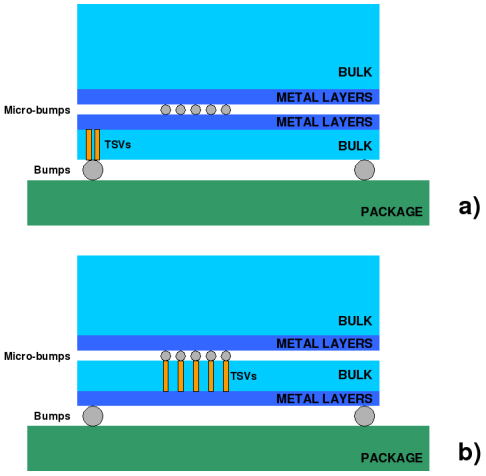


Figure 3: Stacking Options: a)Face-to-Face, b)Face-to-Back

Different TSV integration densities are available today. Low density TSVs have a pitch larger than 100 μm. Medium density TSVs [16] have a pitch of about 50 μm, which represents an integration density of about 500 TSV/mm^2. High Density TSV [10] consists of really high aspect ratio TSVs with a diameter lower than 10 μm, offering an interesting integration density of 10 000 TSV/mm^2.

Once the process is defined, different options can be used for the assembly step. The most natural one comes from flip-chip techniques. It consists of solder micro-balls, which perform both electrical conductivity and mechanical links between dies. In order to reduce existing flip-chip bump dimensions to be compatible with TSV sizes, copper pillars, also known as micro-bumps, have been developed [24]. Like BGA and flip-chip packaging techniques, copper pillars are fabricated using a brazing (soldering) process operation. Copper-pillar diameters have been demonstrated down to 25 μm [16]. A more advanced packaging solution consists in direct copper bonding between two dies without any brazing element by using molecular copper bonding. This type of process is still active R&D and is not yet available due to surface roughness constraints, but will allow higher integration density, probably less than 10 μm [19].

Finally, depending on the side of the two dies (back side is the passive silicon, front side is the active layer), different assembly can be realized (Figure 3). For the Face-to-Face (F2F) connection, two dies ended by a high-level metal interconnection (called "Alu Cap") can be assembled without a thinning step. Nevertheless, TSV process steps are required to interconnect the stack with the package and the thinning process is performed on the assembled dies. For the Face-to-Back (F2B) solution, one of the two dies must be thinned before assembly to reach the TSVs contact. Note that this last solution is the only one which allows stacking more than two dies.

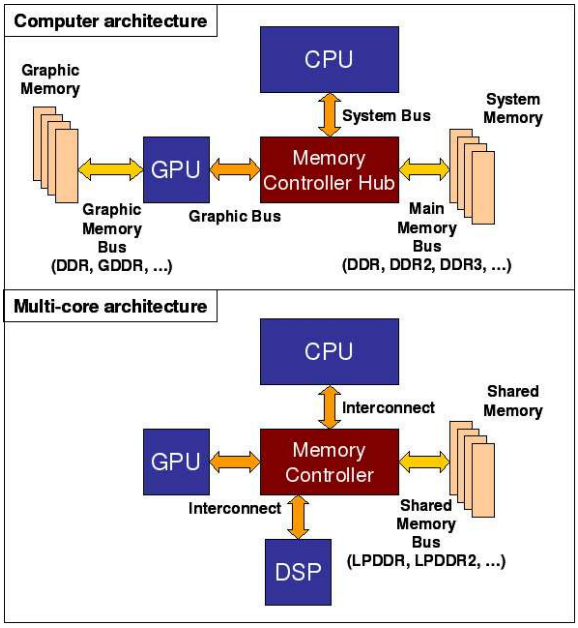


Figure 4: System architectures and memory interconnects

5.2 Improving memory bandwidth

With the increasing demand for processing power in MPSoCs, memory-interconnect bandwidth is becoming the performance bottleneck of the overall system. Tera-scale memory bandwidth will likely be required in the coming years.

5.2.1 The memory-link bottleneck

A typical computer architecture [45], shown in Figure 4, consists of a microprocessor (CPU), a chipset and the main memories connected through the system bus (interconnect). The system and processing memories are split to provide the required bandwidth. In contrast, architectures targeting mobile applications must fulfill hard area and power consumption constraints. As a consequence, modern systems are integrating the chipset (memory controller, graphic component) as IP blocks with the CPU and the modem processor in a single MPSoC. Such organization puts more pressure on the memory bus as all the data and control traffic is merged on the same physical bus.

5.2.2 Bandwidth and power consumption requirements

Peak memory bandwidth is defined as the product of the number of data bits in the memory bus and the speed of a single bit [45]. A typical value reached by the current generation of MPSoCs used for handheld devices is a 32 bit wide bus with a JEDEC standard LPDDR2 with a frequency of 533 MHz. Consequently, the memory bandwidth is:
4 Bytes × 533 MHz × 2 (Double Data Rate) = 4.264 GB/s
The power consumption of such an interface is proportional to the interconnect technology capacitance, the interconnect activity (linked to the frequency of the memory bus), and the square of the voltage excursion.

In a typical MPSoC, many operators are requesting data from the same shared memory: the CPU, the multimedia processors and the modem processor. Complex use cases could occur in such a computing scheme and lead to a heavy demand on memory bandwidth. It is foreseen that the terabyte-per-second [20] memory bandwidth will soon be reached. The following section will see the impact on actual memory interfaces and their evolutions.

5.2.3 High-bandwidth memory interfaces overview

In computer architecture, a widely used memory interconnect is the DDR3 protocol. It is a parallel dual clock edge (DDR) synchronous protocol. The maximum frequency is 1066 MHz and the IO-voltage is 1.5 V. This interface achieves approximately 10 GB/s bandwidth for 32 bit memory with a power consumption evaluated at 30 mW per Gb/s. The devices (processor and memories) are placed in different packages on a PCB. The package and PCB technologies used in computers permit a good transmission line quality to ensure the required signal integrity for 1.5 V-range signaling level at 1 GHz.

In mobile computing MPSoCs, the cost-driven PCB does not allow the required signal integrity for such signaling level and frequency. A dedicated protocol has been standardized by JEDEC to address mobile computing specific environment, targeting lower cost and lower consumption than computer architecture. The most widely used memory interconnect is the LPDDR2 interface. It is derived from the DDR3 protocol and keeps the main features, such as the parallel dual clock edge (DDR) synchronous protocol. The main differences are coming from the signaling voltage (1.2 V) and the lower frequency (533 MHz). These differences aim at reducing power consumption at the expense of lower memory bandwidth.

5.2.4 Memory-interface evolution

As previously noted, the next generations of memory interfaces have to cope with increasing memory-bandwidth demands, while keeping the power consumption at a reasonable limit. The memory bandwidth definition expressed above shows two solutions to improve the bandwidth: increase the number of data bits or increase the frequency. The power consumption can be reduced in three ways: by limiting the interconnect capacitance, by limiting the signal voltage range (quadratic impact), or by limiting the frequency. Finally, each memory interface evolution has to keep the latency stable for performance reasons. With these simple equations in mind, an evolutionary trend is emerging for high bandwidth: wide data memory interface.

5.2.5 Wide data interface

For this interface proposal, the bandwidth improvement is achieved through an unprecedented extension of the bus width between the MPSoC and the memory up to 512 bits. This evolution has been made possible with the introduction of 3D-stacked memories [35]. This 3D technology provides a much higher connection density that makes the overall silicon cost (area) of the wide data memory interface in the same range as the typical LPDDR2 interface located in the MPSoC pad ring.

Lower power consumption is mainly achieved through the lower capacitance of the TSV interconnect. In conjunction with wider data bit width, a Single-Data Rate (SDR) is used and the frequency reduced from 533 MHz for the LPDDR2 interface down to 200 MHz. The lower bit rate significantly simplifies the design of the pad interface. Signal-integrity issues are singularly minimized with SDR signals at 200 MHz through TSVs between dies. On the whole, the bandwidth improvement provided by the wider data bus interface (16 times higher than LPDDR2) compensates and exceeds the performance loss coming from the lower bit rate (5.33 times lower than LPDDR2). The resulting wide data interface throughput is three times that of the LPDDR2.

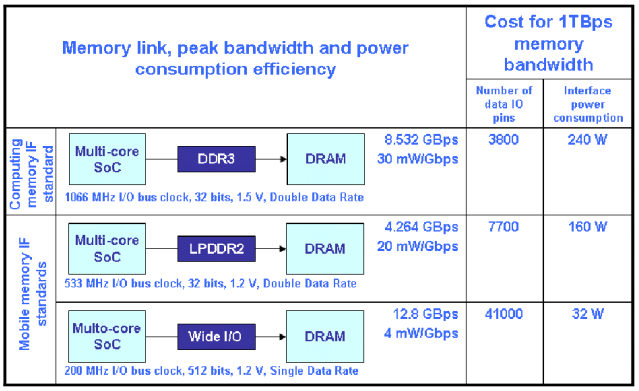


Figure 5: Main features of different high-bandwidth memory interfaces

5.2.6 *Memory-interface performance comparison*

Figure 5 gives a summary of the three different memory interfaces described in this section for the 1 TB/s target. It clearly proves the need to go to large bit-width interfaces for power consumption reasons. In this context, 3D stacking has a clear advantage compared to other interface solutions. DRAM memory on processor vertical integration with wide data interconnects through TSV is seen as the driving 3D technology for improving the multi-core computing performances.

5.3 Wide data interface design integration

5.3.1 *Wide I/O memory architectural specification*

Memory vendors are currently adopting the wide data interface (also called Wide I/O) for their memory devices [29]. The chip architecture of a Wide I/O DRAM is made up of four independent memory partitions with their own 128-bit wide interface. The key distinctive features versus a conventional SDRAM are the following:

- 4-channel SDRAM x 128 b, 200 MHz type interface, 12.8 GB/s peak bandwidth, around 500 mW power
- Maximum number of stacked dies is 4 (1, 2, or 4)
- Wide I/O interface channel pitch of 50 μm x 40 μm
- TSV diameter of $\sim$10 μm
- around 1200 micro-bump connections between chips
- 4-channel micro-bump and TSV arrays located at the center of the chip to ensure electrical connectivity as well as mechanical stability, all channels are symmetric with respect to the chip center
- Boundary scan test mode to detect bump connection failure between chips

5.3.2 *Package integration*

Figure 6 is showing an example of a 2 layer stack with one processor SoC and one Wide I/O memory using the technological assumptions summarized in Table 2. The processor SoC is packaged with a flip-chip technology to improve the power integrity and consequently allows high-performance operating conditions. The memory is stacked on top of the processor SoC in a Face-to-Back configuration. The interconnect, located in the center area of the memory die, consists of an array of micro-bumps whose pitch is in the range

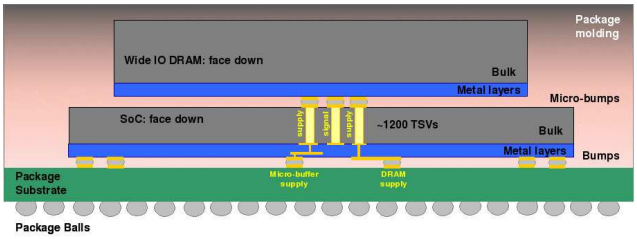


Figure 6: Proposed stacked integrated circuits for Wide I/O

of 40 μm to 50 μm. With such a configuration, TSVs are required within the processor SoC. Some of them are dedicated to Wide I/O memory supply distribution from the package balls up to the memory. The other TSVs are used for the interconnect signals.

Table 2: Technology assumptions

Assembly	W2D, D2D
Stacking	F2B
TSV process	Via Middle
TSV density	10 μm diameter
TSV xy pitch	50 μm x 40 μm
Copper Pillars	25 μm diameter

5.3.3 *MPSoC architecture impact*

Today, we see a trend towards MPSoCs with a throughput-oriented memory hierarchy. As a result, an MPSoC platform is a highly heterogeneous system. It integrates a general-purpose multi-core CPU, and a number of domain-specific many-core subsystems. In such an architecture, the main memory is shared between all sub-systems and is physically located outside the SoC. All instruction and data transfers between the local memories and the external memory device have to go through a centralized memory controller which deals with several roles:

1. It arbitrates between the different initiators and acts as an arbiter or a scheduler
2. It converts the transaction initiated by the sub-systems into a protocol compatible with the memory device
3. It converts the digital protocol into precise timed signals relative to the memory clock edges. It handles all timing delay variability through the physical path made of the SoC package, the board and the memory package. This role is called "PHYsical or PHY interface"

The Wide I/O memory interconnect scheme impacts the memory controller architecture and its integration into the processor SoC versus conventional architectures. To address the four independent partitions of the memory, the arbiter could be either centralized and considered as a unique resource shared among the sub-systems or split into four independent controllers, managing parallel traffic streams to the memory partitions, thus reducing the congestion in the SoC interconnect. A memory controller and a PHY interface are required for each Wide I/O partition and have to be physically placed close to the channels for signal integrity and power integrity reasons. This deeply impacts the floorplan of the processor SoC.

5.4 Conclusions

3D stacking of multi-core heterogeneous system opens a new era of architecture exploration with new partitioning of the overall System-on-Chip. After a description of the 3D stacking technologies, this presentation reviews the existing memory interfaces and demonstrates that they need to evolve to new protocols in order to achieve the terabyte-per-second bandwidth with reasonable power consumption. 3D integration is a unique opportunity enabling memory-interconnect evolution to higher bandwidth. DRAM memory on processor vertical integration with wide data interconnects through TSVs is probably the most advanced scheme for providing high throughput memory connection.

6. REFERENCES

[1] B. Akesson and K. Goossens. Architectures and modeling of predictable memory controllers for improved system integration. In *Proc. DATE*, 2011.

[2] B. Akesson *et al.* Classification and Analysis of Predictable Memory Patterns. In *Proc. RTCSA*, 2010.

[3] B. Akesson *et al.*, "Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration," in *Proc. RTCSA*, 2008.

[4] B. Amir. US Patent 5937425: Flash file system optimized for page-mode flash technologies. 1999.

[5] S. Bayliss and G. Constantinides. Methodology for designing statically scheduled application-specific SDRAM controllers using constrained local search. In *Proc. FPT*, 2009.

[6] C. van Berkel. Multi-core for Mobile Phones. In *Proc. DATE*, 2009.

[7] E. R. Berlekamp. *Algebraic Coding Theory, Revised Edition.* Aegean Park Press, 1984.

[8] R. C. Bose and D. K. R. Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3, 1960.

[9] A. Burchard *et al.* A real-time streaming memory controller. In *Proc. DATE*, 2005.

[10] L. Cadix et al. Integration and frequency dependent electrical modeling of Through Silicon Vias (TSV) for high density 3DICs. In *Proc. IITC*, 2010.

[11] L.-P. Chang and T.-W. Kuo. An adaptive striping architecture for flash memory storage systems of embedded systems. In *Proc. RTAS*, 2002.

[12] L.-P. Chang et al. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Trans. Embed. Comp. Syst.*, 3, 2004.

[13] Y.-H. Chang et al. Improving flash wear-leveling by proactively moving static data. *Comp., IEEE Trans. on*, 59(1), 2010.

[14] Y.-H. Chang and T.-W. Kuo. A commitment-based management strategy for the performance and reliability enhancement of flash-memory storage systems. In *Proc. DAC*, 2009.

[15] J. Charbonnier et al. Wafer level packaging technology development for CMOS image sensors using Through Silicon Vias. In *Proc. ESTC*, 2008.

[16] P. Chausse et al. Polymer filling of medium density through silicon via for 3D-packaging. In *Proc. EPTC*, 2009.

[17] M.-L. Chiang and R.-C. Chang. Cleaning policies in mobile computers using flash memory. *J. Syst. Softw.*, 48, 1999.

[18] S. Choudhuri and T. Givargis. Deterministic service guarantees for NAND flash using partial block cleaning. In *Proc. CODES+ISSS*, 2008.

[19] L. Di Cioccioa et al. Enabling 3D Interconnects with Metal Direct Bonding. In *Proc. ICICDT*, 2007.

[20] P. Franzon. Creating 3D-Specific Systems-Architecture, Design, CAD. In *Proc. DATE*, 2010.

[21] A. Gupta et al. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proc. ASPLOS*, 2009.

[22] S. Heithecker and R. Ernst. Traffic shaping for an FPGA based SDRAM controller with complex QoS requirements. In *Proc. DAC*, 2005.

[23] D. Henry et al. Via First Technology Development Based on High Aspect Ratio Trenches Filled with Doped Polysilicon. In *Proc. ECTC*, 2007.

[24] D. Henry et al. 3D integration technology for set-top box application. In *Proc. 3DIC*, 2009.

[25] P.-H. Hsu et al. A version-based strategy for reliability enhancement of flash file systems. In *Proc. DAC*, 2011.

[26] Intel Corporation. Understanding the flash translation layer (FTL) specification. 1998.

[27] International Technology Roadmap for Semiconductors (ITRS), 2009.

[28] JEDEC Solid State Technology Association. *DDR3 SDRAM Specification*, JESD79-3E edition, 2010.

[29] Jung-Sik Kim et al. A 1.2V 12.8GB/s 2Gb Mobile Wide-I/O DRAM with 4x128 I/Os Using TSV-Based Stacking. In *Proc. ISSCC*, 2011.

[30] P. Kollig *et al.* Heterogeneous Multi-Core Platform for Consumer Multimedia Applications. In *Proc. DATE*, 2009.

[31] T.-W. Kuo et al. An efficient fault detection algorithm for nand flash memory. *SIGAPP Appl. Comp. Rev.*, 11, 2011.

[32] O. Kwon et al. An efficient garbage collection policy for flash memory based swap systems. In *Proc. ICCSA*, 2007.

[33] S.-W. Lee et al. FAST: An efficient flash translation layer for flash memory. In *Lect. Notes in Comp. Sci.*, volume 4097, 2006.

[34] K. Lee *et al.* An efficient quality-aware memory controller for multimedia platform SoC. *IEEE Trans. Circuits Syst. Video Technol.*, 15(5), 2005.

[35] G. H. Loh. 3D-Stacked Memory Architectures for Multi-Core Processors. In *Proc. ISCA*, 2008.

[36] C. Macian *et al.* Beyond performance: Secure and fair memory management for multiple systems on a chip. In *Proc. FPT*, 2003.

[37] Micron Technology. FNNB74A NAND Flash Memory Datasheet, 2010.

[38] Micron Technology. MT29F16G08ABACA NAND Flash Memory Datasheet, 2010.

[39] Micron Technology. MT29F64G08CBAAA NAND Flash Memory Datasheet, 2010.

[40] O. Moreira *et al.* Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proc. EMSOFT*, 2007.

[41] M. Murugan and D. Du. Rejuvenator: A static wear leveling algorithm for nand flash memory with minimal overhead. In *Proc. MSST*, 2011.

[42] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enabling High-Performance and Fair Shared Memory Controllers. *IEEE Micro*, 29(1), 2009.

[43] M. Paolieri *et al.* An Analyzable Memory Controller for Hard Real-Time CMPs. *Embedded Systems Letters, IEEE*, 1(4), 2009.

[44] G. Pares et al. Mid-process through silicon vias technology using tungsten metallization: Process optimization and electrical results. In *Proc. EPTC*, 2009.

[45] L. A. Polka et al. Package Technology to Address the Memory Bandwidth Challenge for Tera-scale Computing. *Intel Technology Journal*, vol. 11, no. 3, 2007.

[46] G. Poupon et al. 3D Integration : a technological toolbox. In *Proc. IMPACT*, 2008.

[47] S. Rixner *et al.* Memory access scheduling. In *Proc. ISCA*, 2000.

[48] J. Shao and B. Davis. A burst scheduling access reordering mechanism. In *Proc. HPCA*, 2007.

[49] L. Steffens *et al.* Real-Time Analysis for Memory Access in Media Processing SoCs: A Practical Approach. *Proc. ECRTS*, 2008.

[50] P. van der Wolf and J. Geuzebroek. SoC Infrastructures for Predictable System Integration. In *Proc. DATE*, 2011.

[51] W.-D. Weber. *Efficient Shared DRAM Subsystems for SOCs.* Sonics, Inc, 2001.

[52] C.-H. Wu et al. An adaptive flash translation layer for high-performance storage systems. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 29, 2010.