ETM^2: Empowering Traditional Memory Bandwidth Regulation using ETM

Alexander Zuepke, Ashutosh Pradhan, Daniele Ottaviano, Andrea Bastoni, Marco Caccamo
Technical University of Munich
Email: {alex.zuepke, ashutosh.pradhan, daniele.ottaviano, andrea.bastoni, mcaccamo}@tum.de

***Abstract*—The Embedded Trace Macrocell (ETM) is a standard component of Arm's CoreSight architecture, present in a wide range of platforms and primarily designed for tracing and debugging. In this work, we demonstrate that it can be repurposed to implement a novel hardware-assisted memory bandwidth regulator, providing a portable and effective solution to mitigate memory interference in real-time multicore systems.**

ETM^2 requires minimal software intervention and bridges the gap between the fine-grained microsecond resolution of MemPol and the portability and reaction time of interrupt-based solutions, such as MemGuard. We assess the effectiveness and portability of our design with an evaluation on a large number of 64-bit Arm boards, and we compare ETM^2 with previous works using a setup based on the San Diego Vision Benchmark Suite on the AMD Zynq UltraScale+.

Our results show the scalability of the approach and highlight the design trade-offs it enables. ETM^2 is effective in enforcing per-core memory bandwidth regulation and unlocks new regulation options that were infeasible under MemGuard and MemPol.

***Index Terms*—real-time system, multicore, memory bandwidth regulation**

I. Introduction

The problem of memory contention under real-time constraints has become increasingly critical on modern embedded platforms, which rely on shared memory hierarchies to achieve high performance and energy efficiency.

Despite the increasing availability of architecture-defined Quality of Service (QoS) mechanisms such as Arm's Memory Partitioning and Monitoring (MPAM) [1], these features are not primarily designed for real-time workloads [2] and still exhibit many implementation-defined and platform-specific characteristics [3], which limit their predictability and portability across different systems-on-chips (SoCs).

In contrast, memory bandwidth regulation implemented in software has been actively researched by the real-time systems community over the past decade. These techniques only rely on timers, interrupts, and the Performance Monitoring Unit (PMU), and are applicable to a wide range of platforms.

MemGuard [4] first implemented an approach leveraging Performance Monitoring Counters (PMCs) to estimate memory activity and throttle cores using an interrupt-based approach. This design allows MemGuard-like techniques to promptly react upon reaching per-PMC memory bandwidth thresholds, but enforcing small bandwidth values and short regulation periods incur overheads [5]–[8].

MemPol [8] removes the burden of interrupt management from the regulated cores and achieved microsecond-scale granularity by moving the regulation logic to an external processing element that periodically polls the PMCs. Despite the much higher resolution and the possibility of monitoring multiple PMCs at the same time, MemPol's polling-based approach cannot immediately react when bandwidth thresholds are reached, leading to *overshooting*. The recently proposed MemCoRe [9] extended MemPol's approach by moving the control logic to an FPGA fabric and exploiting cache-coherence events to achieve nanosecond-scale regulation. Yet, this precision depends on platform-specific FPGA integration, limiting its portability.

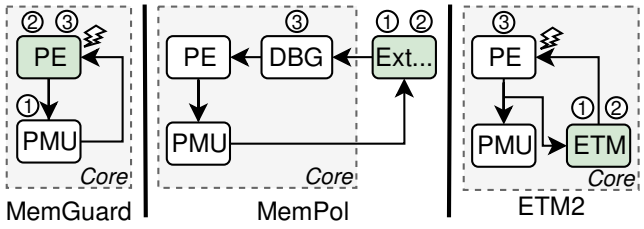


Fig. 1: High-level comparison among traditional memory bandwidth regulation mechanisms MemGuard and MemPol, and the presented ETM^2. High-lighted in green is the regulation logic. Lightning arrows indicate interrupts.

Contrary to these approaches, in this work we show that to achieve *both* fine-grained regulation resolution and prompt reaction times, additional computational resources external to the cores are not required. This allows us to apply our strategy to a large number of Arm-based platforms.

We observe that the debug and trace infrastructure on 64-bit Arm processors already provides (in hardware) all the building blocks required to implement the basic components of software-based memory bandwidth regulation techniques. Specifically: ① integration with architectural elements such as PMCs, ② a control logic, and ③ throttling mechanisms.

As illustrated in Fig. 1, MemGuard relies on the PMU for bandwidth monitoring and regulation interrupts ①. The control logic is implemented by the Processing Elements (PEs) using two interrupt handlers ②, ③ to throttle cores on PMC overflows and handle budget replenishment. In MemPol, the external core implements the control logic by polling the PMU ①, ②, as well as the throttling strategy by using the debug infrastructure to stall and restart the PE ③.

We show that the Embedded Trace Macrocell (ETM) [10], a standard element of the Arm CoreSight [11] architecture

and available on almost all commercial off-the-shelf (COTS) Arm platforms, can be cleverly exploited to perform most of memory bandwidth regulation activities with minimal software support. Our approach leverages the ETM's access to low-level events in the PE to monitor memory bandwidth by tracking memory transactions (see ② in Fig. 1). It also uses the ETM state-machine capability for the regulation logic, generating interrupts to throttle cores when bandwidth thresholds are exceeded ①. The only software support required is a minimal interrupt handler ③ (at operating system or hypervisor level) to acknowledge ETM interrupts and stall the core. Notably, no control logic is implemented in the software handler. We termed our approach ETM^2: *Empowering Traditional Memory Bandwidth Regulation using ETM*.

ETM^2 inherits the strengths of existing regulators. Like MemPol, it achieves microsecond-scale precision and can combine multiple monitoring dimensions (cache refills and write-backs). Like MemGuard, it is interrupt-based and free from overshooting caused by periodic polling of PMCs like in MemPol. Despite lacking the fine-grained physical-address awareness of FPGA-based approaches (*e.g.,* MemCoRe), it preserves full compatibility with commercial Arm-based systems, requiring no hardware modification or FPGA integration.

In summary, the key contributions of this paper are:

- ETM-based regulation: We leverage the CoreSight ETM to implement a hardware-assisted, memory bandwidth regulator directly within the existing debug infrastructure.
- Interrupt-based fine-grained regulation: We demonstrate microsecond-scale regulation intervals with an interrupt-driven approach that avoids PMC-polling overshooting.
- Two regulator designs, *periodic replenishmen (PR)* (*à la* MemGuard) and *token-bucket (TB)* (*à la* MemPol).
- Wide platforms coverage: ETM^2 has been implemented on a broad range of 64-bit Arm SoCs, relying solely on standard CoreSight components.

The remainder of this paper is structured as follows. Sec. II reviews background on memory-bandwidth regulation, focusing on software-based approaches, and provides the necessary context on Arm CoreSight and the ETM. Sec. III presents the design of ETM^2, detailing how CoreSight components are repurposed for regulation. Sec. IV describes our implementation across multiple 64-bit Arm platforms. Sec. V evaluates ETM^2 in terms of correctness, resolution limits, and its comparison against MemGuard and MemPol, using both micro-benchmarks and SD-VBS workloads. Finally, Sec. VI concludes the paper and outlines directions for future work.

II. Background and Related Work

Memory bandwidth regulation has been extensively studied to improve timing predictability and reduce interference, especially in multiprocessor systems-on-chip (MPSoC) systems.

Most software-based approaches use PMCs to measure per-core memory accesses and subsequently trigger regulation activities. Starting with MemGuard [4] (discussed in detail in Sec. II-A), several other studies have experimented with variants of the same technique in the context of hypervisors [6], [12], [13], or to enact regulation based on reads or writes [14].

Compared to MemGuard, the approach proposed by MemPol [8] (see Sec. II-B) implements a finer-grained regulation using a polling-based strategy. PMU-based memory-regulation policies can only approximate the effective utilization of the DRAM controller [15], but very few platforms allow to precisely quantify memory bandwidth using performance counters co-located within the DRAM [7].

Orthogonally to PMC-monitoring, bandwidth regulation has also been implemented using the quality of service (QoS) infrastructure available on some MPSoC [16]–[20], or architectural features such as Arm's MPAM [1] or Intel's RDT [21]. Nonetheless, the limited availability of these mechanisms in embedded systems and their platform-specific, often unpredictable behavior [3], [22], continue to hinder their adoption.

Although this work focuses on hardware-assisted, software-based bandwidth regulation, several prior studies have proposed dedicated FPGA- or hardware-extensions (*e.g.,* [23]–[26]) that implement regulation directly in hardware, or that improve worst-case latency of contended memory transactions with a dedicated memory-controller design (*e.g.,* [27], [28]). On PS-PL platforms (*e.g.,* [29]), following [30], MemCoRe [9] uses the FPGA to monitor cache-coherency traffic and throttle cores similarly to MemPol. In contrast, [31] only monitors cache-miss from FPGA using the CoreSight infrastructure.

A. MemGuard

MemGuard [4] enforces memory bandwidth regulation by assigning each core a budget of memory transactions within a fixed regulation period. The budget is expressed in terms of PMC events, such as last-level cache refills or write-backs. The memory budget is consumed when cores perform memory transactions and is replenished according to the regulation period. MemGuard relies on two interrupts: one is triggered when the PMU counter reaches the assigned budget, upon which the core is temporarily throttled; the other is a periodic timer interrupt that resets the PMU counter and resumes the core if it was previously throttled. This design provides per-core bandwidth control without requiring hardware support beyond standard PMCs, but using more than one counter requires individual budgets for each PMU-based metric. As such, MemGuard allows only limited combinations (*i.e., min*, depending on which PMC fires first) of the contributions of different counters [14]. The interrupt-based mechanism adopted by MemGuard introduces an overhead that increases with smaller regulation periods (more timer interrupts), or with smaller budget assignments (more PMU interrupts due to smaller leeway on bursts) [4], [7], [8]. As a result, MemGuard's periods are coarse-grained (typically 1 ms [8]), limiting the achievable temporal resolution and allowing unregulated bursts of memory-intensive activity.

B. MemPol

MemPol [8], [32] addresses the granularity limitations of MemGuard by decoupling monitoring and enforcement

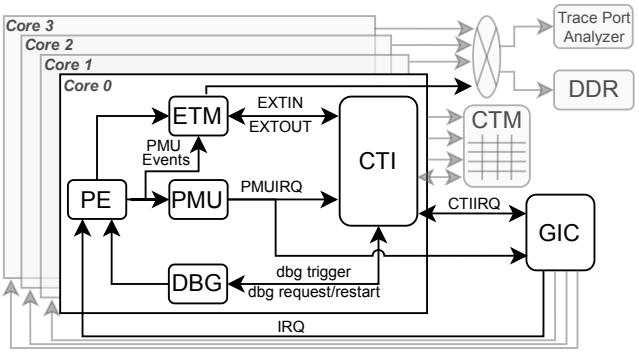


Fig. 2: Arm CoreSight components, highlighting the modules used by ETM[2].

from the regulated cores. Instead of PMU-triggered interrupts, MemPol periodically polls PMCs from an external control core (*e.g.,* one of the real-time cores available on MPSoCs [29]) and uses debug interfaces (*e.g.,* Arm's CoreSight [11]) to halt and resume the target cores when their budget is exceeded. MemPol guarantees a minimum bandwidth over time, using equivalently a sliding window [8] or a token-bucket [9] regulation strategy. The polling-based design enables regulation at microsecond-scale periods, while avoiding any overhead on the regulated workloads (excluding the throttling if the budget is exceeded). MemPol can also flexibly combine multiple PMCs at runtime to derive more accurate estimates of memory pressure than single-counter schemes [8], [20]. However, due to its polling behavior, MemPol cannot prevent *overshooting* the target budget before the regulation can be enacted.

C. Regulation Granularity

Memory bandwidth regulation at small scales (*e.g.,* microseconds, function calls) allows for better integration with hardware-based QoS schemes at interconnect level [16]. However, regulation at small scales is inherently more pessimistic than regulation at coarse scales (*e.g.,* milliseconds, task scheduling): due to the much smaller budgets, a fine-grained regulation has less leeway on short bursts of memory accesses, *e.g.,* `memcpy` or `memset` operations, and throttles cores for a longer time. MemPol favors small scales to reduce overshooting, but allows to compensate bursts with its sliding window or token-bucket mechanisms. Thus the choice of regulation mechanism and granularity eventually depends on the timing requirements of a system, *e.g.,* fine-grained regulation can be beneficial for tasks with periods of 100 µs or below, or when co-regulating with hardware-based QoS.

D. Arm CoreSight

Arm defines *CoreSight* for debugging of its cores and specifies *invasive* and *non-invasive* debug [11]. Invasive debug gives a hardware debugger or a self-hosted software debugger full access to a core's internal state. Non-invasive debug comprises performance monitoring and tracing.

Fig. 2 compactly describes the CoreSight components of 64-bit Arm Cortex-A processors [11]. The debug interface (DBG) offers low-level access to the core's registers for debugging purposes. The PMU provides six counters that can be programmed to count a larger number of PE-specific events and signal interrupts. The ETM monitors control flow-specific information of the PE and has access to the same PE-specific events as the PMU. This is used to generate trace events for other components outside of the core that collect the traces and make them available for further processing. The Cross-Trigger Interface (CTI) provides access to low-level control signals to synchronize DBG, PMU, and ETM events among cores in multicore setups through the Cross-Trigger Matrix (CTM) and raises interrupts to the Generic Interrupt Controller (GIC). Two important interrupts are PMUIRQ to signal PMC overflows and CTIIRQ to forward selected events at the CTI.

In Cortex-A processor cores, DBG, PMU, ETM, and CTI are always part of the core and run in the same clock domain and at the same speed as the PE. CoreSight components are accessible through core internal registers (DBG and PMU only) or as memory-mapped devices. The memory-mapped interfaces are accessible through a low-bandwidth Debug Advanced Peripheral Bus (APB). Unless debugging is disabled or locked down for security purposes, both hardware debuggers and the cores can access the CoreSight components.

Note that CoreSight access is typically disabled in production deployments, as it allows to bypass security perimeters of the kernel or hypervisor [33]. System integrators must therefore carefully consider the risks of giving the cores access to non-invasive CoreSight components for their systems.[1]

E. ETM Background

The ETM monitors internal signals of the PE and a diverse set of resources to implement *filters* and *triggers* for tracing [10]. Filters define *"what"* to trace, while triggers define events to start and stop tracing (*"when"*). Additional registers configure frequency and level of detail of the generated trace data. Fig. 3 shows the ETM resources relevant for this work, which are the same on all 64-bit Cortex-A processors.

External inputs: The ETM is connected to four external input signals from the CTI and a set of low-level signals from the PE that also feed the PMU. The ETM can monitor up to four of these signals *OR*-ed together. Unfortunately, on out-of-order cores such as Cortex-A72, A76 and A78, events related to cache accesses have multiple low-level signals that can become active at the same time.

Address comparators: Eight address comparators match configured addresses or can be used in pairs as address range comparators. Cortex-A cores implement only matches on instruction fetches and for virtual addresses.

Resource selectors: 16 resource selectors function as multiplexer from specific *groups* like external inputs, single or pairs of address comparators, counters, or sequencer states. From the selected group, a selector can combine (logical *OR*) up to 16 specific resources. Selectors can be used in pairs with a Boolean operation, *e.g.,* AND or OR, and can be configured

[1]Access for hardware debuggers (*e.g.,* JTAG) is managed by TAP controllers that typically implement security gateway functionality in the SoC.

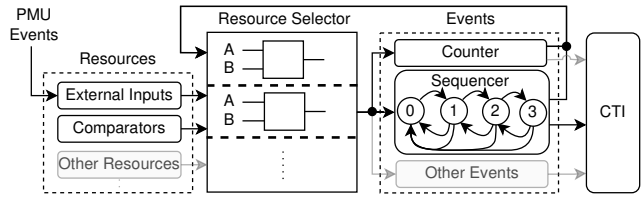


Fig. 3: Internal ETM architecture, highlighting the modules used by ETM[2].

to negate their outputs. The first two selectors are hardwired to always provide *TRUE* (1) and *FALSE* (0) signals respectively.

Counters: The ETM contains two 16-bit counters that count *down* when its configured input signal (resource selector) is active. The counter "fires" when it reaches the value zero. Each counter has a programmable reload value and can be configured to self-reload or on a trigger signal from a resource selector. Depending on whether a counter reloads itself, the counter outputs a pulse (self-reload mode) or level signal (not self-reloading). Additionally, the two timers can be chained together into a single 32-bit counter.

Sequencer: The ETM provides a state machine with four states, $0 \Leftrightarrow 1 \Leftrightarrow 2 \Leftrightarrow 3$. The sequencer allows configuring resource selectors for the three forward and the three backward transitions, and to reset to state 0. The sequencer can change multiple states within the same cycle, as long as the conditions for state transitions evaluate to true. Forward movements take precedence over backward movements, and resets take precedence over both. The sequencer stays in its state when no other conditions evaluate to *TRUE* in the current cycle. Sequencer states can be configured as inputs for resource selectors (*OR*-ed together) with a level signal as output.

External outputs: The ETM can drive four ETM output signals to the CTI depending on the configured resource selectors. The CTI can be configured to route these signals into the PE (to halt or resume the core for debugging purposes), to other cores' CTIs via the CTM, to raise the CTIIRQ interrupt to the GIC, or even back into the ETM.

Programming model: The ETM is programmed by configuring all described resources, using the *FALSE* selector to disable unused resources. In this work, we configure all trace resources, but we disable the generation of trace data, as it is not relevant for bandwidth regulation. Programming the ETM can be particularly cumbersome due to the mixed use of pulses and level outputs and edge- and level-triggered inputs.

III. Design

The core logic of software-based memory bandwidth regulators like MemGuard and MemPol (see Fig. 1) is straightforward. These regulators a) keep track of memory bandwidth usage monitoring last-level cache (LLC) activity, and b) throttle cores (*e.g.,* via interrupt) that exceed their assigned budget until they can resume normal operations (*e.g.,* the budget is *replenished*). These steps could be implemented in hardware *near* the core where the required data is already available.

While researching the ETM's capabilities for bandwidth regulation, we observed that the ETM provides the minimum amount of resources to implement such a regulator, *i.e.,* configurable access to PMU data, two counters to account for LLC activity and time, and two different mechanisms to halt the core, debug halt and CTIIRQ. The driving research questions then became where the limitations and advantages of the ETM are, and which types of regulator designs are possible.

A. External Inputs to the ETM

Like the PMU, the ETM has access to the same low-level signals in the core regarding last-level cache activity. But unlike the PMU, the ETM is limited to monitor four low-level input signals that are logically *OR*-ed together rather than the PMU's six counters and proper adders.[2]

On first generations of Arm cores such as Cortex-A53, A57, and A72, memory bandwidth regulation relies on two PMU events, *i.e.,* L2 cache refills and L2 cache write-backs, that are available as two input signals to the ETM [34]–[36]. However, newer generations, such as Cortex-A76 and A78, require different PMU events [20] that expose three or more low-level signals to drive the PMU counters [37], [38]. This raises the important question of whether accounting *OR*-ed input signals is a sufficient metric for regulation. We will demonstrate feasibility and discuss this in detail in Sec. V-B.

Furthermore, the monitoring of LLC activity available to the ETM cannot be leveraged to realize a global regulation policy like in MemPol. Given a typical four-core cluster with four input and four output signals, propagating each core's individual throttling state through the CTM to the other cores in the cluster requires three of the four inputs and would reduce the LLC activity information to a single input signal.

B. Counters

The ETM provides two 16-bit counters (Sec. II-E) that we use to count PMU events (Sec. III-A) and time using the *TRUE* input selector. We program both timers in self-reloading mode with the core's *budget* (number of cache lines) and *period* for budget replenishment (CPU cycles). Two limitations become obvious: (i) 16-bit counters provide an upper bound for the duration of the period, *e.g.,* 32.7 µs for a 2000 MHz core, and (ii) self-reloading may cause overflows for small budget values when the counter keeps incrementing after the core's budget was exceeded, *e.g.,* due to overheads of an interrupt handler or flushing of internal queues in the core's memory subsystem. We look at the latter problem in detail in Sec. V-D.

C. External Outputs of the ETM

The ETM's external output can drive the low-level input pins to stop (EDBGRQ) or resume (DBGRESTART) the core for debug purposes, or raise an interrupt (CTIIRQ).

A fast low-level stop/resume mechanism would be favorable for memory bandwidth regulation, however a debug halt requires an explicit acknowledgment signal (DBGACK) before the core can be resumed [8], and this signal is not connected

[2]For an event comprising N signals, the PMU employs a combinatorial network and adds the resulting value in the range 0 to 2^{N-1} to the PMC.

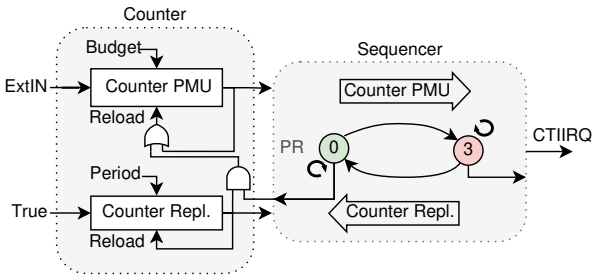


Fig. 4: ETM^2 PR regulation: The design follows MemGuard by using two counters for budget accounting and periodic replenishment. The sequencer turns the counter transitions (edge) into stable output signals (level). State 3 indicates throttling to the OS or Hypervisor via CTIIRQ.

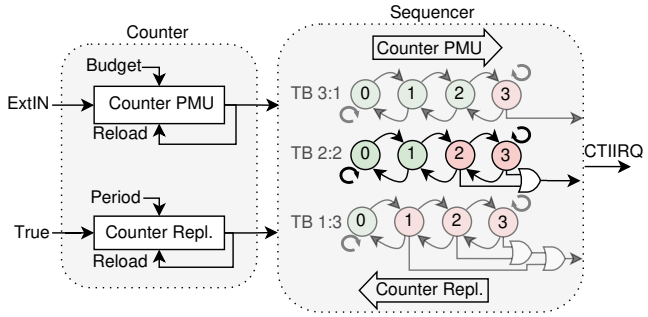


Fig. 5: ETM^2 TB regulation: The design follows MemPol's token-bucket approach with the two counters driving the state machine transitions back and forth. Depending on the TB variant, states 1 to 3 indicate throttling. ETM^2 TB 2:2 provides robustness for both small and large regulation budgets.

to the CTI. Instead, the debug halt state must be acknowledged by a write to a 32-bit register in the CTI's memory-mapped region. This would require busmaster capabilities not available to the ETM, and this obviously cannot be combined with an interrupt handler on the core to write to the acknowledgement register either, as the core is already halted. We therefore did not follow this approach, but opted instead for triggering the CTIIRQ interrupt to throttle the core in the interrupt handler.

Unfortunately, the routing of the CTIIRQ signal to the GIC can be problematic: on some platforms (*e.g.,* the Nvidia Orin) the CTIIRQ is not available, while on others (*e.g.,* the NXP i.MX8 and i.MX93) it is shared among the cores. In this paper, we did not further attempt to find a solution for these systems.

D. Sequencer

When ETM counters reach zero, they provide a pulsed output signal before reloading on the input event. Such a pulsed output is not observable by an interrupt handler. Therefore, we have to use sequencer states to indicate that the core is within or exceeds its configured budget. With two states, we can design a MemGuard-like regulation which we present in Sec. III-E. However, as the sequencer has four states, we evaluated the use of the additional states for regulation purposes, and also designed a MemPol-like regulation which we present in Sec. III-F.

Sequencer states must also be used for address-based filtering when using the address range comparators, *e.g.,* to disable regulation in specific memory regions, or in the OS kernel or hypervisor. The address range comparators emit pulses at different times than PMU input signals, thus the two signals cannot be combined by *OR*-ing them together as a counter input. We tackle these challenges with a design to enable accounting and regulation only in user space (Sec. III-G).

E. Periodic Replenishment (PR) Regulation

The ETM^2 PR regulator provides *periodic replenishment* of the core's budget. Fig. 4 sketches the design. The two counters drive the sequencer state machine in either forward (budget exhausted) or backward direction (budget replenished). We use just two states, 0 and 3, and raise CTIIRQ in state 3. The interrupt handler in the OS or hypervisors polls the sequencer state status register and waits for the state to change back to zero before returning. Important in this design is that the replenishment also resets the budget counter, but we do this only when the sequencer is in state 0. This allows us to accumulate any overuse of bandwidth while in state 3, as the PMU counter resets on the transition $0 \Rightarrow 3$, and reduce the budget from the next regulation period accordingly.

The general behavior of this regulator mimics MemGuard's behavior, except that we don't require interrupts to replenish the budgets, as the ETM-based design implements the replenishment directly in hardware. This solves one of the major performance issues when MemGuard is used with a small regulation period, as there is no overhead for replenishment for the core under regulation. Compared to MemGuard, a limitation of the presented regulator is that the length of the period is limited by the 16-bit counter.

F. Token Bucket (TB) Regulation

The ETM^2 token bucket (TB) regulator uses the additional two sequencer states to model a *token bucket* regulation. Such a design allows the application to compensate for memory access bursts over multiple regulation periods. In this model, the bucket's fill counter can be interpreted as a rational number, where the sequencer state represents the integer part and the counter value represents the fractional part, and replenishment occurs in integer steps, *e.g.,* from 2.427 to 1.427. The two counters drive the state machine forwards resp. backwards. An important difference to the PR regulator is that TB does not reset the budget counter, as this would break the logical bucket model (the replenishment counter could reset the budget, for example, from 2.427 to 1.0).

Fig. 5 shows the design for three variants of the TB regulator. TB 3:1 uses three states to model the acceptable range and one state to indicate that the budget is exceeded. TB 2:2 uses two states for each range. TB 1:3 uses a single state for the acceptable range and three states for budget overuse. The motivation for the three different designs is their behavior *w.r.t.* counter overflows at very small budgets or very large budgets, which we will discuss further in Sec. V-D.

G. Address-based Regulation (PR-user)

We also evaluated the feasibility of address-based regulation, *i.e.,* providing fine-grained control for accounting and/or replenishment when the core executes in specific memory regions. Such an approach can for example distinguish between execution in user mode or kernel mode resp. hypervisor mode [9]. The address range comparators in the ETM provide filtering of instruction execution based on virtual address ranges, the core's current execution state (*i.e.,* EL0 to EL3), and the currently scheduled task or VM IDs based on internal registers (*CONTEXTIDR* for task IDs and the *VMID* portion in *VTTBR* for VM IDs). However, address range comparators only activate (pulse) on instruction execution, therefore we need sequencer states to get a stable level output to combine this information with the pulses from PMU events or counters in resource selector pairs.

For accounting of memory accesses only in user space, dubbed PR-user, we configured the address range comparators to follow the split in the virtual address space between user and kernel space, and use the following four states: 0: <budget/kernel; 1: <budget/user; 2: ≥budget/user; 3: ≥budget/kernel. Transitions $1 \Leftrightarrow 2$ follow the PR regulator design. Transitions $0 \Leftrightarrow 1$ resp. $2 \Leftrightarrow 3$ happen when the core changes execution modes. Budget replenishment additionally drives the state machine $(2 \vee 3) \Rightarrow 0$, and the sequencer adjusts to 1 on the next instruction fetch from user space.

IV. Implementation

ETM^2 needs access to the memory-mapped CoreSight registers from the OS or hypervisor. Consequently, the security implications identified in MemPol [8] also apply: the CoreSight infrastructure can bypass TrustZone protections, and system integrators must account for this.

Each core needs a dedicated CTIIRQ line at the GIC (see Fig. 2). This interrupt must be configured as level-triggered on Cortex-A53 to A72 and edge-triggered for newer cores such as Cortex-A55 or A76. In older SoCs, the interrupt is often connected as shared peripheral interrupt (SPI). Newer SoCs follow Arm's recommendation and provide CTIIRQ as private peripheral interrupt (PPI) 24 to the GIC.

We observed that the ETM stops counting when the core enters an idle state by executing *wait for interrupt (WFI)* or *wait for event (WFE)* instructions. In idle state, the core is not causing interference, but also the automatic replenishment stops. The kernel or hypervisor should therefore enter idle state only during replenished state, which is likely the case when the core has no other pending activity and pending interrupts.

Lastly, as the regulator derives its internal timing for budget replenishment from the *TRUE* resource selector, and therefore indirectly from the core's current clock speed, the core should be configured to a fixed frequency.

A. Software Components

The ETM^2 implementation consists of two software components: (i) a component to set up CoreSight devices and program the core's regulation values, and (ii) an interrupt handler for CTIIRQ that polls the ETM sequencer until the ETM leaves the regulation-specific throttling state. It afterwards acknowledges the CTIIRQ by writing to the CTI interrupt acknowledge register *CTIINTACK* and in the GIC.

Our reference implementation[3] consists of an application to program the ETM from user space and a Linux kernel module that implements the interrupt handler. The application opens */dev/mem* and configures the memory-mapped CoreSight devices for the desired regulator design and the cores' bandwidth settings. The kernel module is configured through board-specific settings for CTIIRQ in the DTB (interrupt number and type, and core affinity) and regulation-specific settings to indicate throttling states.

B. Programming of DBG, PMU, ETM, and CTI

CoreSight devices are programmed by first unlocking the DBG, CTI, ETM, and PMU devices via their memory-mapped interfaces and programming the devices into an initial state with disabled functionality (see Arm specification [11]).

We program the CTI to connect the ETM external outputs 1, 2, and 3 to the CTIIRQ input signal of the CTI. We leave all other signals of the core in a disconnected state.

For the ETM, we program the external inputs to monitor core-specific signals (see Sec. IV-C), the address range comparators (if needed), the resource selectors, the counters, and the sequencer state transitions. We connect ETM outputs to the respective sequencer states that indicate throttled state, and finally enable the ETM.

For the ETM to observe PMU events and thus the regulation to work, the PMU must also be initially programmed to enable the export of PMU events to the ETM by setting the *X* bit (export) in the PMU's PMCR control register. Afterwards, the OS or hypervisor can configure and use the PMU independently from the ETM, as they are separate hardware units.

The ETM^2 PR regulator requires six resource selectors, with free resources to implement address-based filtering. The TB regulator uses one resource selector to configure the external inputs for the PMU counter and six pairs of resource selectors to define the signals that drive the sequencer state machine back and forth. Extensions to the TB design are not possible, as there are no more ETM resources available.

C. Core-Specific Regulation Settings

For memory bandwidth regulation, we consider four memory access operations: *Read* refers to cache refills from load operations or instruction fetches. *Prefetch* refers to cache refills from prefetching and allow stressing the read capabilities of the memory subsystem: the core does not have to wait for the result and handles prefetches in an out-of-order manner, even on in-order cores. Arm cores do not fetch a cache line from memory when the full content of a cache line is overwritten, *e.g.,* in `memset`. Instead, the core allocates a cache line that is eventually written back. We denote this type of access as *write*

[3] https://github.com/rtsl-cps-tum/rtas2026-etm2

operation. Lastly, *modify* describes a partial write to a cache line, requiring the cache line to be fetched from memory and eventually be written back, causing twice the memory activity.

For early generations of 64-bit Arm cores, namely Cortex-A53, A57 and A72, which have private L1 instruction and data caches and use a shared L2 cache as LLC, we monitor the core's L2 activity as

$$B_{A53} = B_{A72} = \textit{L2D\_CACHE\_REFILL} + \textit{L2D\_CACHE\_WB}.$$

This is in line with previous work (*e.g.,* [8], [9]) and was shown to accurately track a core's memory activity.

Cortex-A53: The PMU events are provided as external PMU inputs 21 and 22 to the ETM.

Cortex-A57 and A72: Cortex-A57 and A72 cores have the same programming model for PMU and ETM. The events are provided on ETM PMU inputs 24 and 25.

For newer *Dynamic Shared Unit* (DSU)-based cores, namely Cortex-A55, A76 and A78, Arm extended the memory subsystem to three cache levels with private split L1 caches, private unified L2 caches, and a shared L3 cache as LLC. As no per-core PMU event tracks the LLC activity accurately, we use the same PMU settings for bandwidth monitoring as in [20]. Note that having a private L2 cache is optional for Cortex-A55 cores. Our regulation scheme remains valid regardless of the presence of an L2 cache, as the DSU cache always uses the position of the L3 cache in the programming model [39].

Cortex-A55: For this core, we use three models:

$$B_{A55}^{\textit{pessimistic}} = 2 \times \textit{L3D\_CACHE\_ALLOC},$$

$$B_{A55}^{\textit{moderate1}} = \frac{1}{4} \times \textit{BUS\_ACCESS},$$

$$B_{A55}^{\textit{moderate2}} = \textit{L3D\_CACHE\_ALLOC} + \textit{L3D\_CACHE\_REFILL}.$$

ETM PMU inputs are 33 ($B_{A55}^{\textit{pessimistic}}$), 23 ($B_{A55}^{\textit{moderate1}}$), resp. 33 and 34 ($B_{A55}^{\textit{moderate2}}$) [39]. The models progressively decrease the overestimation of *writes* and *reads* [20].

Cortex-A76 and A76AE: Here we use:

$$B_{A76}^{\textit{pessimistic}} = 2 \times \textit{L2D\_CACHE\_WR},$$

$$B_{A76}^{\textit{moderate1}} = \textit{L2D\_CACHE\_WR} + \textit{L3D\_CACHE\_REFILL},$$

$$B_{A76}^{\textit{moderate2}} = \textit{L2D\_CACHE\_WR} + \textit{L3D\_CACHE\_ALLOC}.$$

Similarly to the A55, these models are progressively more accurate in estimating *writes* and *reads* [20]. ETM PMU inputs for *L2D_CACHE_WR* comprises two signals 73 and 74, for *L3D_CACHE_REFILL* also comprises two signals 158 and 159, and *L3D_CACHE_ALLOC* uses one signal 157 [37].

We note that MemGuard is restricted to using the *pessimistic* and *moderate1* settings on Cortex-A55 and the *pessimistic* setting on Cortex-A76 due to the combination of PMUs required by the other settings.

Cortex-A78 and A78AE: On this core type, we use

$$B_{A78}^{\textit{pessimistic}} = 2 \times \textit{L2D\_CACHE\_WR}.$$

The model regulates *modifies* accurately, but overestimates *reads* and *writes* by 100% [20]. ETM PMU inputs are 103, 104, and 105 [38]. We cannot use better models from [20], as

$$B_{A78}^{\textit{moderate1}} = \textit{L2D\_CACHE\_WR} + \textit{L3D\_CACHE\_REFILL},$$

requires monitoring of six ETM PMU inputs. Likewise,

$$B_{A78}^{\textit{moderate2}} = \frac{1}{4} \times \textit{BUS\_ACCESS\_WR} + \textit{L3D\_CACHE\_REFILL}$$

cannot be used due to the fractional factor in the sum.

V. Evaluation

The evaluation addresses the following questions.

Correctness: Is ETM^2 correctly accounting for LLC refills/write-backs and hitting the bandwidth regulation target.

Boundaries: What are the resolution boundaries of ETM^2 regulation, and what are the most appropriate settings.

Comparison: How does ETM^2 perform in comparison to state-of-the-art approaches, namely MemGuard and MemPol.

A. Setup

We evaluated the applicability of ETM^2 across multiple SoCs and core types, *i.e.,* Cortex-A53, A72, A55, A76 and A78. Table I summarizes tested boards and their settings.

To evaluate the correctness and the boundaries of ETM^2, we run micro-benchmarks on an RTOS, which provides a low-noise environment and allows us to perform measurements close to the hardware without unwanted interference of unrelated tasks, like on a complex OS such as Linux. We use the *bench* memory bandwidth benchmark [40] as load generator and to measure the behavior of different memory access types, *i.e., prefetch*, *read*, *write*, and *modify*.

We compare ETM^2 with MemGuard and MemPol using the same setup that was also used in previous studies like [9], [32]. Specifically, we use a ZCU102 board featuring a Zynq UltraScale+ SoC with four Cortex-A53 cores. Our test system runs Debian 12 with a Linux v6.1 vendor kernel from AMD/Xilinx. We use the *San Diego Vision Benchmark Suite* (*SD-VBS*) [41] in the context of *RT-Bench* [42]. To test the isolation properties of different regulation techniques, we use *IsolBench* [43] as an interference generator (`bandwidth` *write* of 16 MiB size).

B. Correctness of ETM-based Budget Accounting

To verify the correctness of the proposed regulation mechanisms, we need to first verify that the ETM observes the correct sum of the cache-related counters used for regulation on the specific cores, as the ETM can only aggregate the low-level cache signals into one *OR-ed* input. For this, we configure the ETM into a pure counter mode, *i.e.,* we chain the two counters into a single 32-bit counter and disable generation of interrupts, and compare both ETM and PMU counters for various memory access operations over a 32 MiB-sized memory region. For ETM and PMCs, we use the counters listed in Sec. IV-C. Memory access variants with *ldnp* or *stnp* indicate that non-temporal loads and stores were used, which might bypass caching. *Write_dczva* zeros a full cache line with the Arm *DC ZVA* instruction.

Table II shows the results on five representative SoCs to cover each evaluated core type. The table compares the ratio between the memory bandwidth observed by PMU and ETM counters, versus the one observed by the *bench* benchmark. A value of 1.0 for *prefetch*, *read*, and *write* variants resp. 2.0 for

TABLE I: Characterization of the set of platforms used to test ETM^2.

Board	Full Name	Cores	Speed [MHz]	CTIIRQ	IRQ Latency [cycles]	Status
ZCU102	AMD Zynq UltraScale+	4xA53	1200	SPI	81	Works
i.MX8M	Coral AI dev board	4xA53	1200	Shared SPI	-	Not supported
i.MX8MP	Debix Model A	4xA53	1200	Shared SPI	-	Not supported
i.MX93	Debix Model C	2xA55	1692	Shared SPI	-	Not supported
S32G2	MicroSys S32G274AR2SBC2	2x (2xA53)	1000	PPI	80	Works
LX2160A	Microsys miriac SBC-LX2160A	8x (2xA72)	2000	PPI	259	Works
TI AM62x	BeaglePlay	4xA53	1250	PPI	131	Works
TI AM67x	BeagleY-AI	4xA53	1250	PPI	162	Works
TI TDA4VM	TI J721EXCPXEVM	2xA72	2000	PPI	162	Works
TI AM69x	TI AM69 Starter Kit	2x (4xA72)	2000	PPI	163	Works
Raspberry Pi 4	Raspberry Pi 4 Model B	4xA72	600	-	-	CoreSight broken
RK3566	Lubancat Zero N	4xA55	816	PPI-edge	217	Works
RK3568	Youyeetoo YY3568	4xA55	816	PPI-edge	217	Works
RK3588	Firefly ITX-3588J	4xA55+4xA76	1120+1200	PPI-edge	272+308	Works
Orin	NVIDIA Jetson AGX Orin	3x (4xA78)	2000	-	-	No CTIIRQ

TABLE II: ETM^2 correctness evaluation for Cortex-A53/A72/A55/A76/A78. For each operation (*e.g., read*), the table shows the ratio between the memory bandwidth observed by PMU and ETM counters, versus the one observed by the *bench* benchmark.

Board	Core	prefetch_l1		prefetch_l2		prefetch_l3		read		read_ldnp		write		write_dczva		write_stnp		modify		modify_prefetch		modify_stnp	
		PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM	PMU	ETM
ideal		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00	2.00	2.00	2.00	2.00
ZCU102	A53	1.00	1.00	1.00	1.00	-	-	1.00	1.00	1.00	1.00	0.97	0.97	1.00	1.00	1.00	1.00	2.00	2.00	2.00	2.00	2.00	2.00
TI AM69x	A72	1.76	1.74	1.76	1.73	1.76	1.73	1.76	1.74	1.76	1.74	1.00	1.00	1.00	1.00	1.00	1.00	1.76	1.75	1.76	1.74	1.00	1.00
RK3588	A55	1.93	1.91	1.94	1.90	0.00	0.00	1.92	1.81	1.92	1.80	1.00	1.00	1.00	1.00	1.00	1.00	1.94	1.93	1.94	1.91	1.93	1.88
RK3588	A76	0.30	0.29	0.27	0.27	0.06	0.06	2.00	1.90	1.21	1.19	1.00	1.00	1.00	1.00	1.00	1.00	2.00	1.94	2.00	1.93	1.00	1.00
Orin	A78	0.04	0.04	0.01	0.01	0.01	0.01	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00

modify (as it contains both reads and writes) indicates a perfect match between PMU resp. ETM counters and the observed bandwidth, as shown in row *ideal*. Values < 1 show that PMU or ETM counters undercount the memory bandwidth; the regulator is too optimistic. In contrast, values of > 1 show overcounting, *i.e.,* the accounted value is pessimistic.

The Cortex-A53 (Zynq UltraScale+) offers a good match of the selected PMU events for regulation. The ETM counter also matches. The Cortex-A53 core implements prefetches from L3 cache as NOP.

The Cortex-A72 (AM69x) overcounts *reads* and some *modify* operations by 75% (leading to undercounting for *modify*). Here, the ETM counter misses a small percentage of the events observed by the PMC.

The Cortex-A55 (RK3588, $B_{A55}^{moderate2}$) expectedly overcounts *reads* and slightly undercounts *modify* operations. The ETM counter misses a small percentage of events.

On the Cortex-A76 (RK3588, $B_{A76}^{moderate2}$), *prefetches* are not correctly accounted for, and *reads* are expectedly overcounted. The ETM counter misses about 7% of events.

For the Cortex-A78 (Nvidia Orin, $B_{A78}^{pessimistic}$), we see the same problem for *prefetches* as on the Cortex-A76. However, the measurement shows a good match of both PMCs and ETM accounting to the ideal baseline.

Overall, the measured results are in line with previous results [20]. This shows that the accounting of PMU events by the ETM is appropriate for memory bandwidth regulation.

A second important aspect for the regulation is the latency *from* observing an event at the ETM *to* throttling in the interrupt handler. To measure this latency, we program the ETM to trigger an interrupt when it observes an unaligned load/store operation that crosses two cache lines, and we can observe that an interrupt becomes pending to the core by polling the *ISR_EL1* register. We then measure the time in CPU cycles from the unaligned memory access to the arrival of the interrupt at the core, effectively bypassing any operating system overheads. As shown in Table I (column *IRQ latency*), the latency is up to a quarter of a microsecond on all platforms.

C. Regulation Behavior at Small Regulation Periods

We evaluate the regulation behavior at small regulation periods in Fig. 6, and explain two design decisions in ETM^2. We construct two modified variants of ETM^2 PR, PR-stop and PR-user, and compare the regulation behavior for a replenishment period of 5 µs and small budget values.

PR (Fig. 6 (left)) is shown as reference and regulates as expected after a minimum bandwidth is achieved (see also Sec. V-D). For low bandwidth targets, the regulation misses its target by going over the two diagonal dashed lines (top dashed line for *prefetch*, *read*, and *write*; bottom dashed line for *modify*—recall from Sec. IV-C that modify comprises both read and write parts and achieves only half of the bandwidth target, however the regulator observes the sum of the PMCs, as the purple dashed line shows), but eventually converges for bandwidths $\geq$350 MB/s. We discuss this effect in Sec. V-D.

PR-stop (Fig. 6 (center)) *stops accounting* PMU events if the budget is exceeded, like MemGuard. The regulation misses its target and shows a systematic *regulation error* that can be attributed to: (i) the latency of CTIIRQ, (ii) buffering of outstanding transactions in the memory subsystem, and (iii) the fact that the operating system also performs memory accesses during interrupt handling. We observe that *read* and *prefetch* exceed the target by up to eight cachelines (the number of cachelines is shown on top and applies to the y-axis as well). We attribute this error to latency effects, as the Cortex-A53 core has limited out-of-order capabilities on loads. For *write* and *modify* the absolute error increases to about 20 cachelines

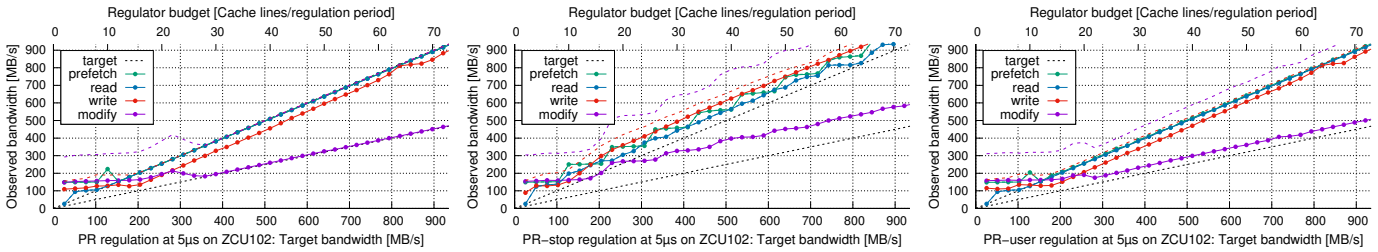


Fig. 6: ETM^2's PR regulator variants on ZCU102 show different types of *regulation errors* relevant at small regulation periods. PR (left, reference) regulates as expected at $\geq$350 MB/s. PR-stop (center) *stops accounting* of PMU events if the core's budget is exceeded, but ongoing memory transactions finish later. PR-user (right) accounts PMU events in *user space* only. Cache-line eviction by *e.g.* interrupt handlers is not accounted for. In all three graphs, colored dashed lines show observed PMC values.

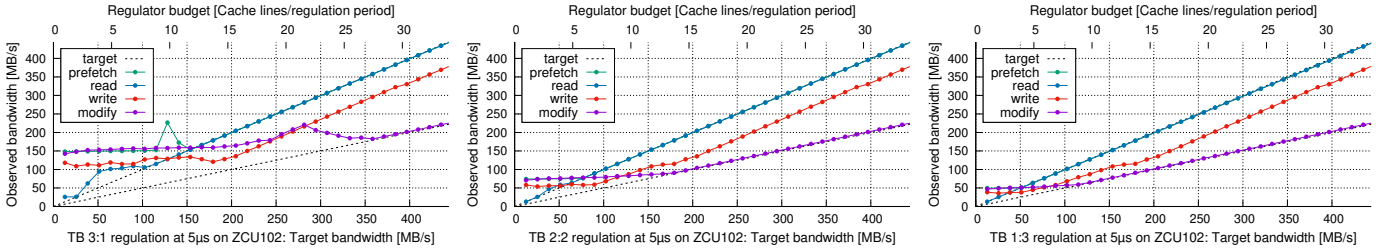


Fig. 7: ETM^2's TB regulators on ZCU102 for small regulation values (bandwidth targets). TB 3:1 (left) shows problems of PMU counter overflows for small budget values as its PMU counter overflows multiple times. TB 2:2 (middle) is more robust to counter overflows and allows smaller regulation values. TB 1:3 (right) improves further.

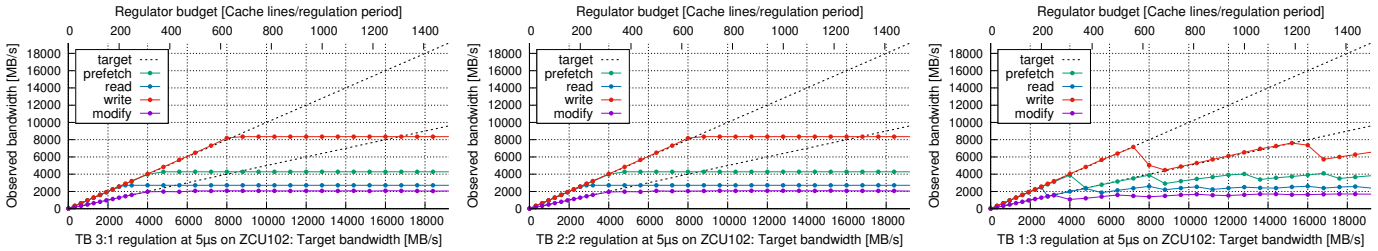


Fig. 8: ETM^2's TB regulators on ZCU102 for large regulation values (bandwidth targets). TB 3:1 (left) and TB 2:2 (middle) regulate correctly at large regulation values. TB 1:3 (right) oscillates at peak bandwidth levels.

due to the additional buffering of transactions. We conclude that regulation at small scale must never stop accounting.

PR-user (Fig. 6 (right)) accounts for PMU events in *user space* only. In our experiment, which uses an RTOS with timer and regulation interrupts as kernel activities, only *modify* shows a regulation error, which is likely caused by delays in accounting of buffered transactions, as code and data is hot in the cache. However, any kernel activity would contribute to regulation errors if accounting is excluded for the kernel.

Note that these effects are mostly relevant for regulation at microsecond scale. For larger periods, these effects have less impact on regulation quality, as their absolute error is small.

D. Boundaries and Limitations of the ETM² Regulation

We show the limitations of ETM^2 when using budget values that are too small. Fig. 7 (left) shows the behavior of the TB 3:1 regulator for a replenishment period of 5 µs, similar to Fig. 6. We observe that the regulation misses its target, but eventually converges when the given budgets become larger. This effect is caused by *overflows of the PMC counter* in the ETM (recall from Sec. III-F that ETM^2 accounts all the time). Shifting more accounting capabilities toward the throttling state mitigates the regulation error, and the regulation hits the bandwidth target much earlier, as in Fig. 7 (middle) and (right) for TB 2:2 resp. TB 1:3. Note that for such peak memory accesses at small budgets, we can observe the same limitation in the TB regulator and on all platform, as Sec. V-E shows.

However, TB 1:3's better regulation at small values comes at a cost, as Fig. 8 shows. TB 1:3 (right) starts to oscillate once we reach peak bandwidth levels, as the TB regulator triggers the throttling state due to the missing reset of the ETM counter in the design, see Sec. III-F. We therefore recommend using only the TB 3:1 and TB 2:2 variants of the TB regulators, and the former only for larger bandwidth values.

E. Large-scale Evaluation

Figs. 9 to 14 show the PR regulation on a wide range of platforms. Due to space constraints, results for further boards are presented in the extended version of the paper [44]. As in Figs. 6 to 8, the top graph shows the regulation behavior

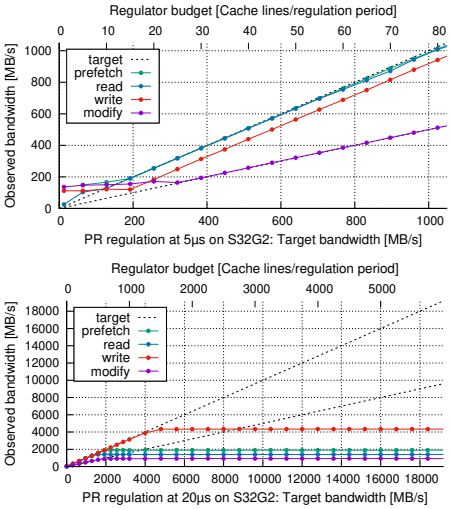


Fig. 9: S32G2 with 4x Cortex-A53.

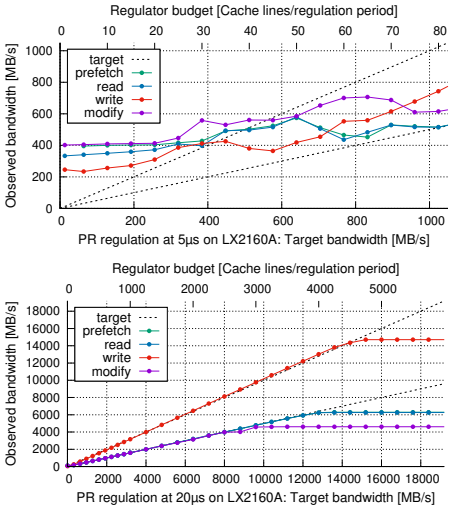


Fig. 10: LX2160A with 16x Cortex-A72.

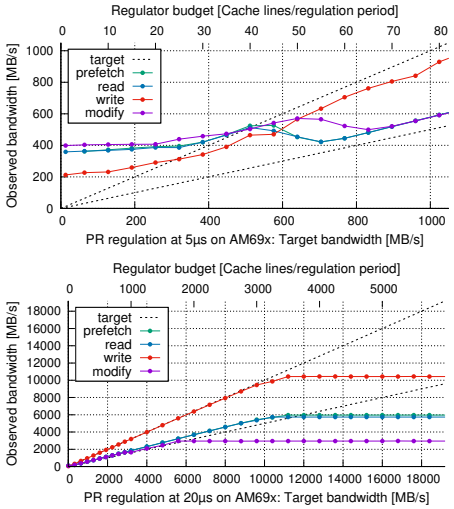


Fig. 11: AM69x with 8x Cortex-A72.

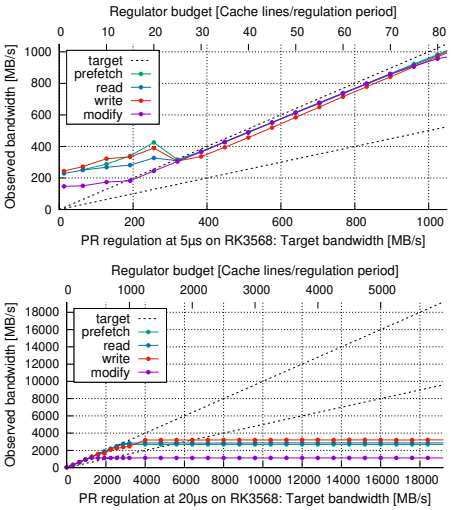


Fig. 12: RK3568 with 4x Cortex-A55.

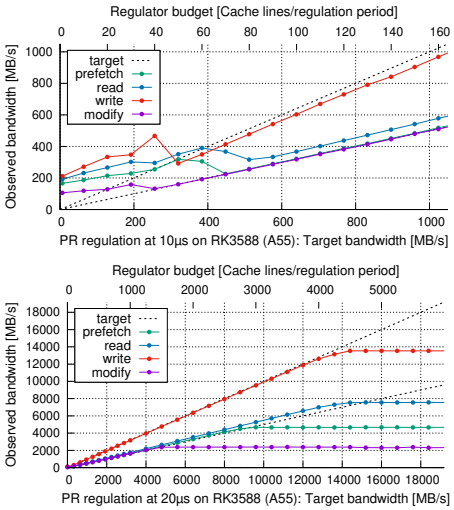


Fig. 13: RK3588 with 4x Cortex-A55.

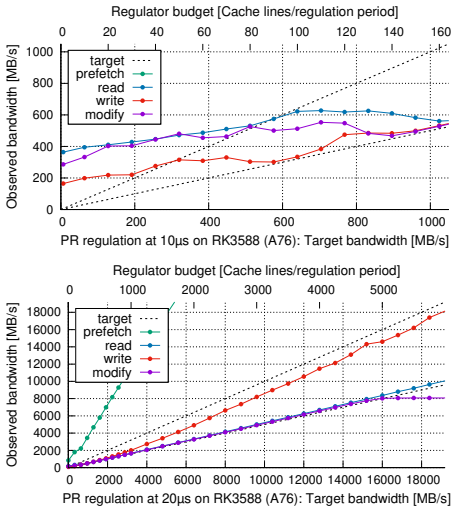


Fig. 14: RK3588 with 4x Cortex-A76.

for typical memory budgets on embedded platforms, *i.e.,* less than 1000 MB/s, for a small replenishment period of 5 µs (10 µs on RK3588). The bottom graph shows the regulation behavior for a larger replenishment period of 20 µs and over a wider range of bandwidth targets and allows for comparison of platform-specific factors. As noted in Sec. V-D, these PR trends also apply to TB 3:1 and TB 2:2. The bottom graphs in each figure primarily illustrate the limitations of each platform. The top graphs show the minimum budget that is required for the regulation to hit the target bandwidth.

On Cortex-A53-based SoCs (Fig. 9), we observe similar trends at low memory bandwidth targets as previously discussed in Sec. V-C (Fig. 6) and Sec. V-D (Figs. 7 and 8).

On Cortex-A72-based SoCs (Figs. 10 and 11), the trend is similar. In general, the Cortex-A72 cores can sustain a higher memory bandwidth, as the platforms have more powerful memory controllers. The regulation can keep the bandwidth target for sufficiently large budgets and replenishment periods.

On Cortex-A55-based SoCs (Figs. 12 and 13), we see similar trends. However, the RK3588 has a much more powerful memory controller, so the regulation stabilizes later. On the same SoC, we can see that the memory subsystem of the Cortex-A76 is much more powerful than the A55 (Fig. 14). Note that the top graphs in Figs. 13 and 14 show the regulation at 10 µs; otherwise, the interesting range would be out of range. The regulation follows the trends discussed in Table II, and the regulation slightly undercounts for both Cortex-A55 and A76.

We were not able to evaluate ETM^2 on Cortex-A78, as our platform, the Nvidia Orin, does not route CTIIRQ to the GIC.

F. Comparison of ETM², MemGuard, and MemPol

We compare ETM^2 with MemGuard and MemPol using SD-VBS on the Zynq UltraScale+ ZCU102 and on the RK3588. For MemGuard and MemPol, we use the reference implementations provided by the authors. MemGuard is implemented as a kernel module [4], [5]. The MemPol regulator runs on the first Cortex-R5 real-time core in its default settings

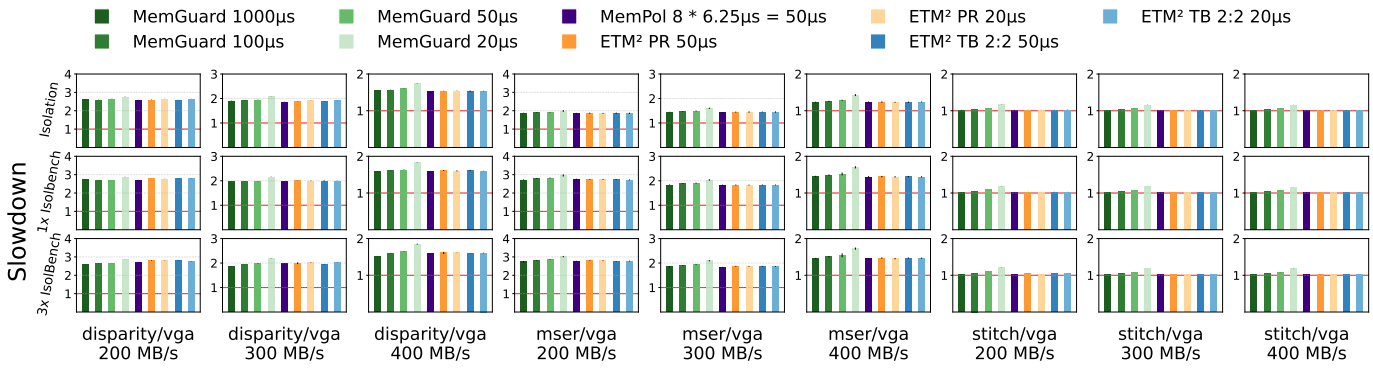


Fig. 15: Slowdown of SD-VBS benchmarks at different bandwidth targets (columns) and increasing interference (rows) regulated by MemGuard, MemPol, and ETM^2 using a single PMC (*L2D_CACHE_REFILL*) on ZCU102. See Sec. V-F for details.

with a 6.25 µs polling interval and a sliding window size of 8 for regulation over a period of 50 µs [32].

We use two experimental settings: the first (similar to [32]) uses a single PMC to track *reads* and is the baseline for comparing with MemGuard (Fig. 15). The primary goal is to show that the best variants of each regulation strategy achieve comparable results. The second setup (Fig. 16) uses different core-specific regulation settings (see Sec. IV-C) to illustrate the effects of pessimistic and moderate regulation strategies. In both experiments, we run SD-VBS benchmarks with *vga* image size under regulation at different bandwidth settings.[4] The experiments distribute fractions of the, empirically determined, bandwidth available under worst case conditions (1000 MB/s on the ZCU102 [32], 2000 MB/s on the RK3588 [20]) between benchmarks and interference tasks. The bandwidth targets for the benchmarks are 20%, 30%, and 40%. The interference cores use and share the remaining bandwidth, *i.e.,* 80%, 70%, and 60%. The first row shows the results for execution of SD-VBS benchmarks in isolation. The middle and bottom rows show the results with increasing cores running IsolBench [43] as load generator and causing memory interference. The graphs show the normalized *slowdown* of execution times compared to unregulated execution without interference (thin red line). The colored bars indicate the average over 10 runs. Small vertical black lines show min/max.

Fig. 15 shows the results for regulation based on cache refills. Here, the regulation provided by ETM^2 nearly matches previously reported results for MemGuard and MemPol [32]. For MemGuard, we observe that for small regulation periods like 50 µs and 20 µs, the overhead increases and often exceeds the other regulators. ETM^2 and MemPol show slightly higher slowdown on memory intensive benchmarks like *disparity* and *mser* than MemGuard. Note that MemGuard only regulates user space code and does not account for kernel activities like system calls or interrupt handling, see Sec. V-C. The memory intensive benchmarks also expose that ETM^2 has slightly higher overhead than MemPol. We attribute this to ETM^2's interrupt-based throttling, but we have not further investigated this effect nor the timing differences to MemGuard. Otherwise, all regulators behave similarly on compute intensive benchmarks like *stitch* (see also [44]). Differences between ETM^2's PR and TB regulations are minimal, with TB 50 µs being slightly faster. All regulators show also a similar behavior in providing isolation from interference, as the comparison between the top row (isolation, no interference) and the bottom rows (interference from IsolBench) shows. Also, the regulation behavior of ETM^2 nearly matches MemPol's. The results provide empirical evidence that *OR*-ing is not a problem for both real-world applications and micro-benchmarks.

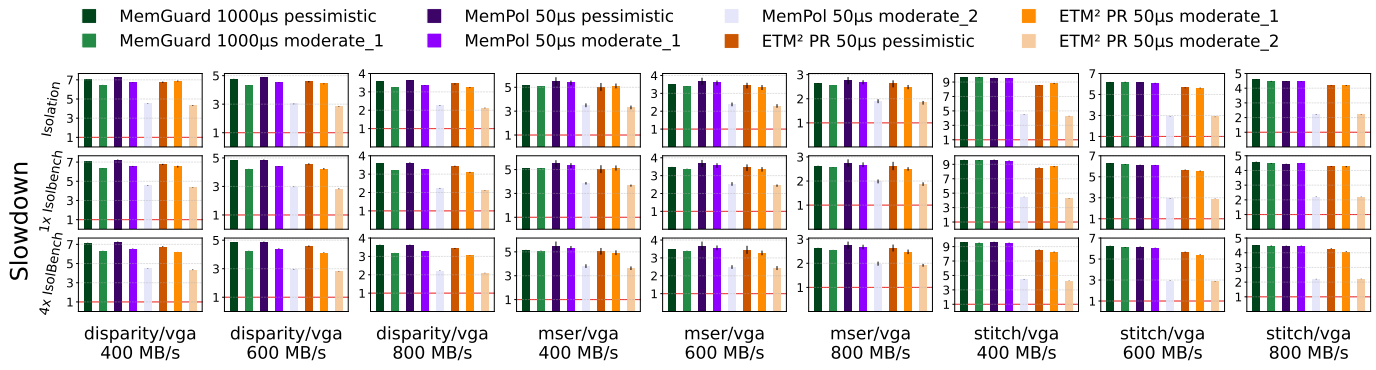
Fig. 16 shows the results on newer Cortex-A55 and A76 cores (RK3588) for regulation with the settings from Sec. IV-C. Here, in addition to the *pessimistic* regulation settings, both ETM^2 and MemPol can use the less pessimistic *moderate* ones.[5] As expected, more accurate accounting is beneficial over *pessimistic* regulation models. Similarly to Fig. 15, under pessimistic settings, MemGuard behaves comparably or slightly better than the other regulators, but, when available on the target platform, moderate settings proves much more effective in achieving lower slowdowns. We also note that on newer cores, the impact of enforcing a reduced bandwidth is much more pronounced that on older cores. This can be observed on, *e.g., disparity* on the Cortex-A76, where the slowdown reaches up to factor 80. Lastly, we can observe that memory bandwidth regulation alone does not protect from other types of interference, *e.g.,* in the shared LLC, as the differences between the top row and the bottom rows show.

G. Discussion

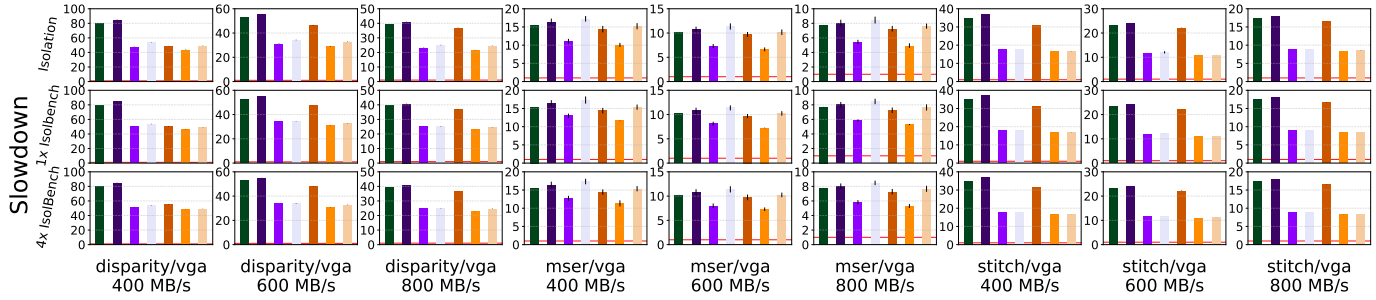
Overall, ETM^2 behaves as intended, delivering regulation performance that closely matches MemGuard and MemPol. For small regulation periods, ETM^2 shows less overhead than MemGuard, as no timer interrupts for periodic replenishment are needed. Like MemPol, ETM^2 depends on the CoreSight debugging infrastructure. However, ETM^2 does not require a dedicated core for regulation purposes, as the ETM state machine implements the regulation.

[4]For space reason, Figs. 15 and 16 only show the most relevant trends. Results for *sift* and *tracking* benchmarks are available in [44].

[5]Due to space constraints, we compare only the best configurations of the regulators and limit interference to 1x IsolBench running on 1x Cortex-A55 and 4x IsolBench running on 2× Cortex-A55 and 2× Cortex-A76.



(a) Cortex-A55 (RK3588)



(b) Cortex-A76 (RK3588)

Fig. 16: Slowdown of SD-VBS benchmarks at different bandwidth targets (columns) and increasing interference (rows) regulated by MemGuard, MemPol, and ETM^2 using the regulation settings from Sec. IV-C on the RK3588. See Sec. V-F for details.

Although the use of *OR*-ed inputs can lead to undercounting, our experiments indicate that this effect is minimal in practice and does not affect regulation accuracy. The effect of overflowing the PMU event counter in the ETM regulation, leading to errors in the regulation, can be mitigated by using larger budgets, larger periods, or by using the TB regulator instead of PR. In any case, our experiments provide baseline thresholds for safe operation of the regulation on each platform.

The leeway of the token-bucket regulation in the PR regulator to compensate for memory bursts is quite limited, but sufficient to extend the regulator's period over a longer time span than what is possible with a single 16-bit cycle counter. We would have wished for more states in the ETM sequencer or additional counters to cope with this limitation.

Another technical challenge for ETM^2 is the lack of precise events to monitor cores' LLC activity on modern DSU-based Arm cores like Cortex-A55 and A76. The bandwidth regulators proposed in [20] were instrumental for the implementation and evaluation on the large scale of SoCs, but are limited in the precision *w.r.t.* monitoring of cores' memory activity. Table II shows that ETM^2, similar to MemPol, can use the *optimistic* bandwidth regulation schemes, while MemGuard must use the *pessimistic* ones [20]. Overall, MemPol allows for the greatest flexibility in combining multiple PMCs, as it allows arbitrary factors for each PMC.

As the comparison to MemGuard shows, long regulation periods compensate for memory bursts better than short regulation periods. This is a known, positive effect, especially when bandwidth regulation is only needed to balance memory accesses between cores at the software-level and with a resolution that matches the tasks' execution times. Instead, the fine-grained regulation in ETM^2 and MemPol allows co-regulation at a much lower scale and at the hardware level of other busmasters in the system, similar to QoS features of Arm interconnects [16]. Ultimately, the selection of the regulator depends on the application.

VI. Conclusion

In this paper, we have shown that the Arm's CoreSight ETM can be repurposed to implement an effective hardware-assisted memory-bandwidth regulator for real-time multicore systems. Our ETM^2 design achieves microsecond-scale regulation intervals and fast reaction times without relying on PMC polling or dedicated external control cores. Our novel approach complements the strengths of prior regulators: it offers the fine-grained precision and multi-dimensional monitoring of MemPol, while preserving the responsiveness of interrupt-driven approaches like MemGuard. We evaluated the correctness and capabilities of ETM^2 on a broad range of 64-bit Arm SoCs, demonstrating its wide portability. Furthermore, we compared with previous results from MemGuard and MemPol in a similar evaluation context based on ZCU102 and SD-VBS.

Overall, ETM^2 can scale efficiently and enables regulation capabilities that were previously infeasible with MemGuard, without requiring dedicated cores as in MemPol.

Future work will focus on evaluating similar hardware support capabilities on new generations of Arm v9 processors.

ACKNOWLEDGMENTS

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

REFERENCES

[1] Arm, "Arm Memory System Resource Partitioning and Monitoring (MPAM) System Component Specification," https://developer.arm.com/docs/ihi0099/ Accessed: 2025-11-11.

[2] M. Zini, D. Casini, and A. Biondi, "Analyzing Arm's MPAM from the perspective of time predictability," *IEEE Trans. Computers*, vol. 72, no. 1, pp. 168–182, 2023. [Online]. Available: https://doi.org/10.1109/TC.2022.3202720

[3] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger, "A Closer Look at Intel Resource Director Technology (RDT)," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS)*, 2022, p. 127–139. [Online]. Available: https://doi.org/10.1145/3534879.3534882

[4] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013, pp. 55–64. [Online]. Available: https://doi.org/10.1109/RTAS.2013.6531079

[5] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory Bandwidth Management for Efficient Performance Isolation in Multi-Core Platforms," *IEEE Transactions on Computers*, vol. 65, no. 2, p. 562–576, 2016. [Online]. Available: https://doi.org/10.1109/TC.2015.2425889

[6] N. Dagieu, A. Spyridakis, and D. Raho, "Memguard: A memory bandwith management in mixed criticality virtualized systems memguard KVM scheduling," in *10th Int. Conf. on Mobile Ubiquitous Comput., Syst., Services and Technologies (UBICOMM)*, 2016.

[7] A. Saeed, D. Dasari, D. Ziegenbein, V. Rajasekaran, F. Rehm, M. Pressler, A. Hamann, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Memory Utilization-Based Dynamic Bandwidth Regulation for Temporal Isolation in Multi-Cores ," in *2022 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, p. 133–145. [Online]. Available: https://doi.org/10.1109/RTAS54340.2022.00019

[8] A. Zuepke, A. Bastoni, W. Chen, M. Caccamo, and R. Mancuso, "Mempol: Policing core memory bandwidth from outside of the cores," in *29th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2023, San Antonio, TX, USA, May 9-12, 2023*. IEEE, 2023, pp. 235–248. [Online]. Available: https://doi.org/10.1109/RTAS58335.2023.00026

[9] I. Izhbirdeev, D. Hoornaert, W. Chen, A. Zuepke, Y. Hammad, M. Caccamo, and R. Mancuso, "Coherence-aided memory bandwidth regulation," in *IEEE Real-Time Systems Symposium, RTSS 2024, York, UK, December 10-13, 2024*. IEEE, 2024, pp. 322–335. [Online]. Available: https://doi.org/10.1109/RTSS62706.2024.00035

[10] Arm, "Embedded Trace Macrocell Architecture Specification ETMv4.0 to ETM4.6," https://developer.arm.com/docs/ihi0064/ Accessed: 2025-11-09.

[11] ——, "Arm CoreSight Architecture Specification," https://developer.arm.com/docs/ihi0029/ Accessed: 2025-11-09.

[12] P. Modica, A. Biondi, G. Buttazzo, and A. Patel, "Supporting temporal and spatial isolation in a hypervisor for ARM multicore platforms," in *2018 IEEE International Conference on Industrial Technology (ICIT)*, 2018, pp. 1651–1657. [Online]. Available: https://doi.org/10.1109/ICIT.2018.8352429

[13] J. Martins, A. Tavares, M. Solieri, M. Bertogna, and S. Pinto, "Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems," in *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, ser. OpenAccess Series in Informatics (OASIcs), M. Bertogna and F. Terraneo, Eds., vol. 77. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, pp. 3:1–3:14. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/11779

[14] M. Bechtel and H. Yun, "Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019, p. 357–367. [Online]. Available: https://doi.org/10.1109/RTAS.2019.00037

[15] P. Sohal, R. Tabish, U. Drepper, and R. Mancuso, "E-WarP: A System-wide Framework for Memory Bandwidth Profiling and Management," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020. [Online]. Available: https://doi.org/10.1109/RTSS49844.2020.00039

[16] Arm, "ARM CoreLink QoS-400 Network Interconnect Advanced Quality of Service," https://developer.arm.com/docs/dsu0026/ Accessed: 2025-11-11.

[17] P. Houdek, M. Sojka, and Z. Hanzálek, "Towards predictable execution model on ARM-based heterogeneous platforms," in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, 2017, pp. 1297–1302. [Online]. Available: https://doi.org/10.1109/ISIE.2017.8001432

[18] A. Serrano-Cases, J. M. Reina, J. Abella, E. Mezzetti, and F. J. Cazorla, "Leveraging Hardware QoS to Control Contention in the Xilinx Zynq UltraScale+ MPSoC," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. B. Brandenburg, Ed., vol. 196. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 3:1–3:26. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13934

[19] M. Zini, G. Cicero, D. Casini, and A. Biondi, "Profiling and controlling I/O-related memory contention in COTS heterogeneous platforms," *Software: Practice and Experience*, vol. 52, no. 5, pp. 1095–1113, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3053

[20] A. Pradhan, D. Ottaviano, Y. Jiang, H. Huang, J. Zhang, A. Zuepke, A. Bastoni, and M. Caccamo, "Predictable Memory Bandwidth Regulation for DynamIQ Arm Systems," in *31st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2025, Singapore, August 20-22, 2025*. IEEE, 2025, pp. 126–137. [Online]. Available: https://doi.org/10.1109/RTCSA66114.2025.00022

[21] Intel, "Resource Director Technology," https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html Accessed: 2025-11-11.

[22] A. Pradhan, D. Ottaviano, A. Zuepke, A. Bastoni, and M. Caccamo, "Work-in-Progress: A First Practical Look at Arm's MPAM for Real-Time Systems," in *IEEE Real-Time Systems Symposium, RTSS 2025, Boston, MA, USA, December 2-5, 2025*. IEEE, 2025, pp. 592–595. [Online]. Available: https://doi.org/10.1109/RTSS66672.2025.00056

[23] Y. Zhou and D. Wentzlaff, "MITTS: Memory Inter-Arrival Time Traffic Shaping," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 532–544. [Online]. Available: https://doi.org/10.1109/ISCA.2016.53

[24] J. Cardona, C. Hernández, J. Abella, and F. J. Cazorla, "Maximum-contention control unit (MCCU): resource access count and contention time enforcement," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, J. Teich and F. Fummi, Eds. IEEE, 2019, pp. 710–715. [Online]. Available: https://doi.org/10.23919/DATE.2019.8715155

[25] F. Farshchi, Q. Huang, and H. Yun, "BRU: Bandwidth Regulation Unit for Real-Time Multicore Processors," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 364–375. [Online]. Available: https://doi.org/10.1109/RTAS48715.2020.00011

[26] N.-J. Wessman, F. Malatesta, J. Andersson, P. Gomez, M. Masmano, V. Nicolau, J. Le Rhun, G. Cabo, F. Bas, R. Lorenzo, O. Sala, D. Trilla, and J. Abella, "De-RISC: the first RISC-V space-grade platform for safety-critical systems," in *2021 IEEE Space Computing Conference (SCC)*. IEEE, 2021, pp. 17–26.

[27] R. Mirosanlou, M. Hassan, and R. Pellizzoni, "DRAMbulism: Balancing Performance and Predictability through Dynamic Pipelining," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020, pp. 82–94. [Online]. Available: https://doi.org/10.1109/RTAS48715.2020.00-15

[28] P. K. Valsan and H. Yun, "MEDUSA: A Predictable and High-Performance DRAM Controller for Multicore Based Embedded Systems," in *2015 IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*, 2015, pp. 86–93. [Online]. Available: https://doi.org/10.1109/CPSNA.2015.24

[29] AMD, "Zynq UltraScale+ Device Technical Reference Manual," https://docs.amd.com/r/en-US/ug1085-zynq-ultrascale-trm/Zynq-UltraScale-Device-Technical-Reference-Manual Accessed: 2025-11-11.

[30] D. Hoornaert, S. Roozkhosh, and R. Mancuso, "A Memory Scheduling Infrastructure for Multi-Core Systems with Re-Programmable Logic," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. B. Brandenburg, Ed., vol. 196. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 2:1–2:22. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13933

[31] M. Baryshnikov, "FPGA-based support for predictable execution model in multi-core CPU," Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Prague, Czech Republic, May 2018.

[32] A. Zuepke, A. Bastoni, W. Chen, M. Caccamo, and R. Mancuso, "Mempol: polling-based microsecond-scale per-core memory bandwidth regulation," *Real Time Syst.*, vol. 60, no. 3, pp. 369–412, 2024. [Online]. Available: https://doi.org/10.1007/s11241-024-09422-8

[33] Z. Ning, C. Wang, Y. Chen, F. Zhang, and J. Cao, "Revisiting ARM Debugging Features: Nailgun and Its Defense," *IEEE Transactions on Dependable and Secure Computing*, no. 01, pp. 1–16, 2021. [Online]. Available: https://doi.org/10.1109/TDSC.2021.3139840

[34] Arm, "Arm Cortex-A53 MPCore Processor Technical Reference Manual," https://developer.arm.com/docs/ddi0500/ Accessed: 2025-11-11.

[35] ——, "Arm Cortex-A57 Core Technical Reference Manual," https://developer.arm.com/docs/ddi0488/ Accessed: 2025-11-11.

[36] ——, "Arm Cortex-A72 MPCore Processor Technical Reference Manual," https://developer.arm.com/docs/100095/ Accessed: 2025-11-11.

[37] ——, "Arm Cortex-A76 Core Technical Reference Manual," https://developer.arm.com/docs/100798/ Accessed: 2025-11-11.

[38] ——, "Arm Cortex-A78 Core Technical Reference Manual," https://developer.arm.com/docs/101430/ Accessed: 2025-11-11.

[39] ——, "Arm Cortex-A55 Core Technical Reference Manual," https://developer.arm.com/docs/100442/ Accessed: 2025-11-11.

[40] A. Zuepke, "Memory Benchmark," https://gitlab.com/azuepke/bench.

[41] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "SD-VBS: The San Diego vision benchmark suite," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2009, pp. 55–64. [Online]. Available: https://doi.org/10.1109/IISWC.2009.5306794

[42] M. Nicolella, S. Roozkhosh, D. Hoornaert, A. Bastoni, and R. Mancuso, "Rt-bench: An extensible benchmark framework for the analysis and management of real-time applications," in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, ser. RTNS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 184–195. [Online]. Available: https://doi.org/10.1145/3534879.3534888

[43] CSL-KU, "Isolbench," https://github.com/CSL-KU/IsolBench.

[44] A. Zuepke, A. Pradhan, D. Ottaviano, A. Bastoni, and M. Caccamo, "ETM2: Empowering Traditional Memory Bandwidth Regulation using ETM," 2026. [Online]. Available: https://arxiv.org/abs/2603.16490