Software SRAM: High performance, low latency memory for consolidated real-time systems

Software SRAM enables developers with a new and innovative way to control the contents of cache directly, protecting critical data from eviction by noisy neighbors or even threads within the same application.

Abstract
Shared caches have long presented a challenge in the development of real-time applications. Caches provide significant performance improvements for throughput-oriented applications and can even be beneficial for latency sensitive applications, provided the latency sensitive data can be found in the cache when requested. On consolidated systems where many applications may be sharing the same cache resources, keeping the critical contents in cache can be a challenge. This paper proposes a new solution that enables developers to have control over what contents are placed into and remain in the cache.

Introduction
The transformation of industrial manufacturing represents yet another movement towards consolidation, where formerly dedicated functions are aggregated onto standard IT systems. The first movement was IT and data center systems in the early 2000’s, followed by communications equipment manufacturers and the migration to virtualized network appliances in the mid 2010’s. For industrial manufacturing, Industry 4.0 leverages many of the same underlying principles of the previous consolidation movements, while bringing a unique set of requirements that emphasize determinism and low latency (versus high utilization through over provisioning). Instead of virtualizing an email server or a network firewall, this movement of workload consolidation introduces a new type of virtual appliance: a programmable logic controller (PLC), which can have hard real-time requirements. Unlike an email that arrives 10 milliseconds late and goes unnoticed by the end user, missed deadlines for hard real-time applications running on a PLC (which typically control mechanical devices) have the potential for catastrophic failures.

Table of Contents
Abstract ................................................ 1
Introduction ......................................... 1
Software SRAM ................................... 2
Software Enablement via RTS ........ 4
Conclusion ........................................... 5
Deploying a PLC as a virtual appliance on a consolidated system comes with a set of challenges as the requirements can be mutually exclusive. Customers desire high performance multi-core systems capable of running multiple applications in parallel. Simultaneously, one or more of those applications may have strict temporal requirements and require dedicated resources, Quality of Service (QoS) controls, and other provisions to meet their latency and jitter performance requirements. Balancing these requirements is essential—reserving just enough resources for the real-time application(s) without significantly degrading the performance for the remainder of the system.

Caches, which can be implemented as a shared resource, are one point of contention when tuning real-time applications on consolidated systems. When a real-time application is running standalone on an otherwise idle system, the temporal performance achieved in this configuration (with uncontested access to the cache) often becomes the baseline. When additional applications or virtual machines are introduced to the system, it is difficult to predict cache residency of the real-time application. This is because the contents of the cache are managed by hardware, and the real-time application may be evicted from cache to make room for the other applications on the system. For virtual PLC appliances, where quantifying the worst-case execution time (versus average performance) is foundational, reducing jitter because of shared caches often becomes a part of the optimization process.

Modern Intel processors support Cache Allocation Technology (CAT)\(^1\), which enables developers with QoS over the shared caches. While CAT has proven to be extremely effective in minimizing the noisy neighbor effect (when load applied on the system affects the timing of another application), it does not allow software developers to control the contents of the cache programmatically.

Intel® Time Coordinated Computing (Intel® TCC) introduces a new feature, specifically software SRAM, that goes beyond cache partitioning to provide software developers with the ability to place latency sensitive code and/or data into the cache. Temporal isolation, or the reliable mitigation of the noisy neighbor effect, can be improved with software SRAM technology.

This paper focuses on software SRAM, designed with input from Real-Time Systems GmbH and supported in Real-Time Systems Hypervisor, a technology that is easy to use and satisfies the low-jitter requirements of industrial applications.

Software SRAM
Real-time developers have tried many techniques to reduce the jitter associated with shared caches. Whether attempting cache coloring, cache warming, or the optimization of code and data sizes for increased probability of remaining in cache, these approaches bring complexity, over provisioning (with no guarantees the data remains in cache), or additional development costs.

Software SRAM, an innovative technology enabled as part of Intel® TCC, brings the ability to move from a cache whose contents are managed completely by hardware, to a cache where a portion of the contents can now be managed by software. Software SRAM maps a portion of the physical address space into cache and protects those addresses from eviction. Once a system administrator configures a portion of the cache for software SRAM, developers can place high priority code or data into a designated memory region. Developers are not required to gain an understanding of cache topologies or hardware register interfaces. Once a software SRAM region is established it is accessed in the same manner as the rest of physical memory.

Figure 1 represents the cache hierarchy of an Intel® processor with support for software SRAM.

---

In this example, software SRAM regions have been reserved (orange bars) in the private L2 caches of Core 2 and Core 3, as well as in the shared L3 cache. Once configured, these regions are now addressable by software and provide extremely low latencies and jitter when compared to DRAM. These performance characteristics are beneficial for real-time developers whose applications require bounded worst-case execution time.

To demonstrate the effectiveness of software SRAM regions, a random pointer chasing workload is used to measure the latency of accessing data elements of a buffer. To ensure consistent run-to-run results, the same seed value is used to determine the pseudo-random pattern for accessing the buffer. The buffer can reside in DRAM or the L2/L3 cache with the use of software SRAM. This type of workload is ideal for quickly exposing the worst case as it attempts to defeat the hardware prefetchers that would hide the cache latency observed with a linear access pattern. It is also important to add internal stress in-between accesses of the buffer, to simulate concurrent workloads running on the system that have the potential to evict the measurement buffer from cache. When software SRAM is used to establish the buffer in the L2 or L3 cache, it is protected from eviction by the stressor application.

The implementation of the random pointer chasing workload and internal stress is available as part of Intel® TCC Tools as a cache sample application. Figure 2 illustrates the behavior of the sample application, which implements a random pointer chasing workload over two separate buffers.

Figure 2. Cache sample application that implements random pointer chasing over two buffers

The sample application measures the time it takes to complete 512 accesses to the software SRAM region (orange), and then attempts to pollute the cache by accessing a separate 16MB buffer (gray). It repeats this process until the specified number of iterations are met. The reported latency value is the total time to access all 512 elements of the software SRAM region. Figure 3 illustrates the output of the cache sample application over three levels of memory hierarchy: DRAM, L3 software SRAM, and L2 software SRAM, as measured on the Intel® Core™ i7-1185GRE Processor for 10,000 iterations.2,3

Figure 3. Intel® TCC Tools Cache Sample Application on the Intel® Core™ i7-1185GRE Processor

---


3 Intel® Core™ i7-1185GRE Processor, 2 x 8GB DDR4 3200 MT/s, Yocto Project * Linux 5.4.77-rt43, Intel® TCC Mode BIOS Enabled
The data in Figure 3 demonstrates both a latency and jitter reduction (tighter bands) when utilizing software SRAM regions in the L2 or L3 caches. For hard real-time applications where missing a deadline can be considered a critical failure, software SRAM technology provides another tool that developers can use when optimizing the temporal performance of their applications.

Software Enablement via the RTS Hypervisor

The RTS Hypervisor is the first hypervisor to bring a novel integration of software SRAM technology to market in the form of low-latency, shared memory regions. Shared memory is a simple and effective method of communication between operating systems running on different CPU cores in parallel. It is based on a portion of DRAM easily accessible at any time by any real-time or non-real-time operating system (RTOS). Users can configure one or more shared memory “partitions,” assign names to identify them in software, and optionally define visibility and access permissions. During runtime any application may use a simple API (provided by RTS Hypervisor) to get a pointer to the shared memory, which can then be used as if it were heap memory. Cache coherency is maintained automatically by the processor regardless of whether caches are shared or not.

This makes using shared memory for data exchange convenient and fast, but shared memory regions are subject to the same cache eviction policies as the rest of system memory. The likelihood of shared memory content to remain cached depends on the cache architecture, as well as the memory usage of other processes running in the same OS. For applications with hard real-time requirements, the worst-case scenario is crucial. Reaction and computing times may increase if data must be fetched from DRAM because it was evicted from cache. Not only is accessing DRAM much slower compared to the cache, when this worst case happens, it becomes unpredictable for the designer of the real-time application who must ensure that deadlines are never missed.

When shared memory is combined with software SRAM, developers now have access to low latency memory regions shared between guest operating systems using a familiar and simple API. The concept is illustrated in Figure 4:

![Figure 4. Cache partitioning with software SRAM backed shared memory region](image)

In this example, a quad-core processor is used to run an RTOS on one core and a general-purpose operating system (GPOS) on three cores. Caches are separated because they are either:

- Exclusively used by their connected core (level 1 and level 2 caches in the example)
- Using Cache Allocation Technology in case of the shared level 3 cache

The GPU, which shares the L3 cache with the CPU cores, is restricted to the portion of the cache assigned to the GPOS, protecting the headless RTOS from jitter caused by the graphics engines usage of the shared L3 cache. On top of separating the L3 cache using CAT, which still allows the respective portions of the cache to load and evict data, a shared memory area is created using a dedicated portion of the L3 cache. The majority of the shared L3 cache (light blue & light orange) continues to operate with the standard replacement policy where cache contents are evicted to make room for new requests. However, the contents in the shared memory region combined with software SRAM (dark orange) are no longer subject to eviction from the cache, unless explicitly done so through software. Any OS running on any core may now read from or write to the shared memory and have the data returned from the cache (indicated by the orange arrows). Accessing time critical data put into such memory therefore gets faster and has less jitter.
In the RTS Hypervisor configuration file, one key called `cache_locked` turns a regular shared memory partition into one whose data stays in the cache. Low level details such as the cache architecture and topology are handled by the RTS Hypervisor, making it easy for the user to configure it without the need to comprehend system specific differences. The value assigned to the configuration key defines which cache level the data should be locked in, if applicable. A value of “1” locks the data into the last level cache (as illustrated in Figure 4), “2” into the second-from-last (if this level is shared, otherwise the last level is used) and “3” locks the memory into the shared cache that is closest to the CPU cores, no matter what the actual cache architecture looks like. This method makes the configuration scalable and portable. A typical configuration section looks like this:

```
[...]
# The following section specifies a shared memory area

[/SHM/1]

    "name"     = "data exchange"
    "size"     = uint64: 0x100000 # one megabyte
    "cache_locked" = uint32: 3       # lock data in the fastest shared cache
```

The shared memory API, provided by the RTS Hypervisor and which is available in all operating systems running on top of the RTS Hypervisor:

- Tells applications where the shared memory regions reside in memory
- Tells applications how to identify the shared memory regions
- Provides access to the shared memory regions, which may optionally be backed by software SRAM

Shared memory does not necessarily have to be “shared,” it can also be made available for one OS exclusively or configured “read-only.” Backward compatible interfaces allow existing OS images to benefit from software SRAM as well. If previous designs utilized the RTS Hypervisor shared memory API prior to the introduction of software SRAM, no changes are required to the application to make use of the new software SRAM backed shared memory regions. It is transparent to the application and relies only on configuration file changes in the hypervisor.

With multiple operating systems running in parallel, granting access to the same portion of software SRAM allows for fast and deterministic inter-OS communication at the same time. Throughput and low latency no longer must be exclusive.

**Conclusion**

Software SRAM enables developers with a new and innovative way to control the contents of the cache directly, protecting critical data from eviction by noisy neighbors or even threads within the same application. As cache sizes increase generation to generation, allocating a portion of the cache for low latency memory can be done with lessening impact to residual system performance. Once software SRAM regions have been initialized, they can be used by hypervisors, operating systems, middleware, or user applications to place time sensitive code and/or data into low latency memory regions. The regions are also flexible and can be sized appropriately per design, growing or shrinking as requirements change. RTS Hypervisor’s integration of software SRAM makes it easy for developers to set up and use, receiving the performance benefits of low latency memory without changes to their BSP or operating systems.