Reducing Power Consumption of microcontroller sub-systems using a Cache Controller

Olivier Montfort. Dolphin Integration

Introduction

Strong market requirements for decreasing the power consumption of embedded systems and increasing complexity of systems-on-chip require innovative solutions addressing such needs. In microcontrollers systems using memories like Flash, OTP, EPROM or EEPROM, the memories can consume a large part of the consumption of the whole system. Using a cache with such memories offers a double advantage. It first enables to increase speed performances by reducing memory latency and increasing throughput, which is the usual approach and, much more interesting if the cache is well parameterized and designed with appropriate features, it enables to significantly reduce the power consumption due to optimization of memory accesses.

For example, the power consumption of an embedded Flash in 180 nm process can be reduce up to 4 times and its average access time divided by 2 when using it with Dolphin I-Stratus-LP cache controller.

Cache concepts

A cache, meaning here a cache memory, the associated controller and a tag memory, is a local, high speed and low consuming memory which stores copies of frequently-used data from a background memory so that most accesses don't need to retrieve data from the slow and power consuming background memory.

However, the cache can only store a limited number of data due to its small size. As a result, various data of the background memory compete for space in the cache and evict each other. The cache structure determines how data will be stored in cache memory and which data will replace data already available in the cache.

The Figure 1 shows two memories. Each location in each memory contains a block of data called "line" or "block". Each location in each memory also has an index, which is a unique number used to refer to that location. The index for a location in main memory is called an address. Each location in the cache has a tag that contains the index of the block of data in background memory that has been cached.

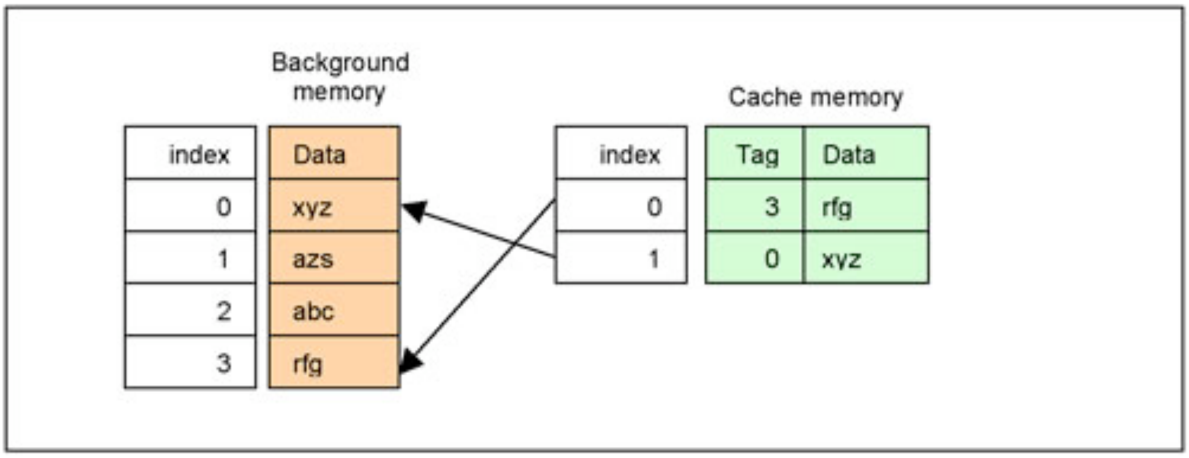

Figure 1: Diagram of a CPU cache

When a data needs to be read from or written at a location in background memory, the cache controller first checks whether that memory location is in the cache. This is accomplished by comparing the tag address of the memory location to all tags in the cache that might contain that address. (see Figure 2)

If the memory location is found in the cache, we say that a **cache hit** has occurred; otherwise, we speak of a **cache miss**.

In the case of a cache hit, the data is immediately read from or written to the cache line. The proportion of accesses that result in a cache hit is known as the hit rate, and is a measure of the effectiveness of the cache. In the case of a cache miss, the cache controller allocates a new entry, which comprises the tag just missed and a copy of the new block from the background memory. Generally the block copied from background contains more data that the word required. It enables to prepare future accesses to neighbor address locations. The size of the block is named cache line size and is selected by the low significant bits of data address called "offset" (see Figure 2). Increasing cache line size enables to improve cache hit, but implies that accesses on a cache miss will take more time and be more consuming. As the background memory is already a slow and consuming memory, the choice of the cache line size is critical for memory system performances.

In order to make room for the new entry on a cache miss, the cache generally has to evict one of the existing entries. The heuristic that it uses to choose the entry to evict is called the replacement policy. The fundamental problem with any replacement policy is that it must predict which existing cache entry is least likely to be used in the future. There are a variety of replacement policies to choose from and no perfect way to decide among them.

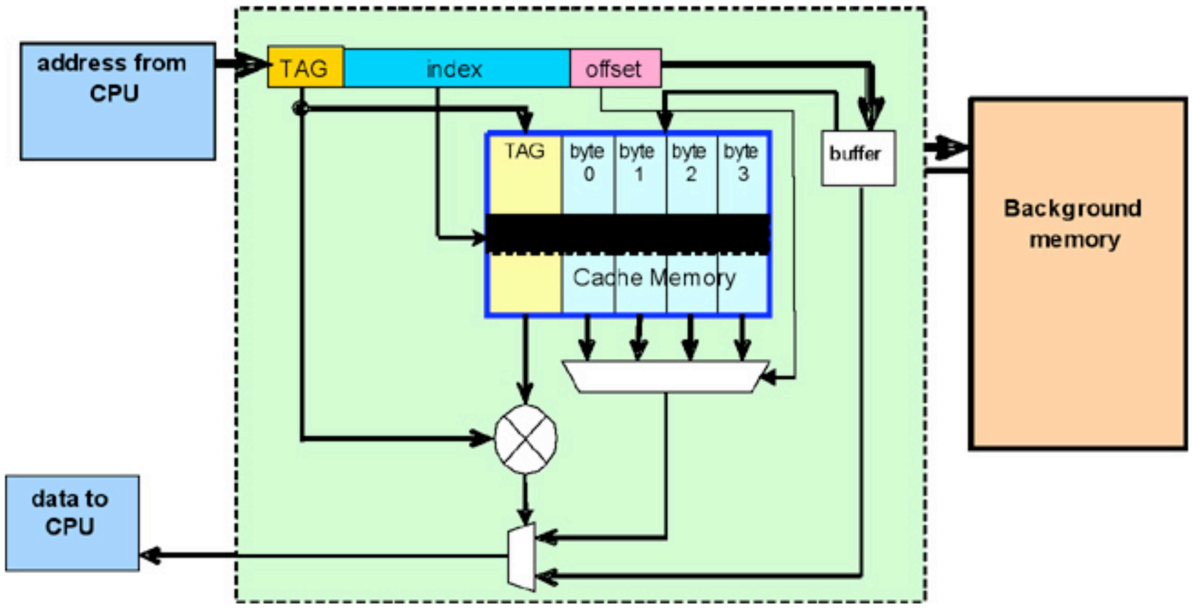

Figure 2: Structure of a cache controller

The “**associativity**” of a cache is related to the question: Which background memory locations can be cached by which cache locations?

The replacement policy decides where in the cache a copy of a particular block of background memory will go. If the replacement policy is free to choose any entry in the cache to hold the copy, the cache is called “fully associative“. On the opposite side, if each block of background memory has a single destination in the cache, the cache is called “direct mapped”. Many caches implement a compromise in which each block in background memory can be affected to one among N places in the cache: they are called “N-way set associative”. When accessing an N-way associative cache, N tags need to be read first, to check is the accessed data is available in cache memory. When increasing associativity, the accesses to cache memories (cache and tag) will then increase and as a consequence the power consumption of the cache. However, caches with more associativity suffer fewer misses and increasing associativity will reduce accesses to background memory. Selecting the right cache associativity will consist in finding the best trade-off between cache controller and background memory performances.

Solving cache configuration issues with I-Stratus-LP.

As a result, there are many parameters for configuring a cache (cache capacity, cache line size, associativity, replacement policy...) and each of them can have a significant impact on the power consumption of the overall system. For a given program, we can determine an optimum set of parameters that will minimize the power consumption of the system. Moreover, for each subset of this program, another optimal configuration may exist. As a consequence, if we consider the power consumption of the overall sub-system, the optimal configuration of the cache will vary all along the execution of the program. So, it is very difficult for a SoC integrator to find the optimal cache configuration for its specific application and the tuning of cache parameters appears to be too complex for non-experts.

The Dolphin I-Stratus-LP cache controller, in addition to offering an optimized architecture for reducing power consumption, is self-configurable. All along program execution, the sequence of memory accesses is analyzed by the controller and the lowest consuming configuration of the parameters is determined and applied dynamically.

This feature avoids the Soc Architect to be a ‘specialist’ of cache and to spend time to determine the “trade-off” configuration that will in average be the less consuming. Even if the application program is not known by the Soc Architect, the performances of the cache coupled to the background memory will always be the more efficient.

Conclusion:

Using a cache is the solution to combine the advantage of specific memories like Flash, OTP, EPROM, or EEPROM with the performances of fast and low consuming memories.

Thanks to I-Stratus-LP, Soc-Integrators can now consider the cache and its background memory as classical memory, and benefit for the best performances.

More details on I-Stratus-LP Cache Controller here.
http://www.dolphin.fr/flip/logic/peripherals/logic_stratus.html

See information on 16-bit MCU 80251 Hurricane, THE optimal solution for upgrading your current 8051-based solutions or targeting 16-bit low power and high density applications.
http://www.dolphin.fr/flip/logic/16bit/logic_80251_hurricane.html

About the Author
Olivier Montfort is the manager for the development of the microcontroller solutions at Dolphin Integration. He has over 7 years of experience in the design of embedded memories and microcontrollers.

Olivier holds a master's degree in electrical engineering from ENSEIRB in Bordeaux, France.