How to manage Cortex-M7 Cache Coherence on the Atmel SAM S70 / E70
Prerequisites
Atmel Technical Presentations

• Atmel SMART Cortex-M7 Introduction

• Using the SAM S70/E70 Cortex-M7 Memory Protection Unit and Memory Barriers for Memory Accesses Ordering

• Atmel SMART SAM S70 / E70 Product Highlights
Outline

• Cortex-M7 Caches Introduction
  • Cache Terminology
  • Reading / Writing Policies
  • SAM S70/E70 Caches Implementation

• Cache Coherence Concerns
  • DMA Writes / Reads Operations
  • Non-Optimal / Recommended Solutions

• Cache Coherence Management
  • Using Cortex-M7 Cache Maintenance Operations

• Cache Coherence Management Limits
  • Non Cacheable Memory Regions Solution
  • Data TCM (DTCM) Solution

• Appendix
  • Atmel Software Package Use Cases Implementations
  • References
Cortex-M7 Caches Introduction
Cortex-M7 Caches Introduction

Cache Terminology: Basics

- **Cache Way**
  - Refers to a particular cache ‘page’

- **Cache Line**
  - Smallest loadable unit of a cache
  - Always a block of contiguous words in memory

- **Cache Set**
  - The same line from each cache way grouped together forms a set

- **Cache Hit**
  - Access to an information already in the cache

- **Cache Miss**
  - Access to an information not in the cache
Cortex-M7 Caches Introduction

Cache Terminology: Cache Entry Structure

• Cache entries usually have the following structure:

  • Cache Line (Data Block)
    • Contains the actual data fetched from main memory

  • Tag
    • A part of the address of the data fetched from main memory, used to identify the cache line

  • Flag Bits
    • Valid bit: indicates whether or not a cache line has been loaded with valid data
    • Dirty bit (Data Cache only): indicates the cache line has been updated by the CPU since it was read from main memory
Cortex-M7 Caches Introduction

Cache Terminology: Addressing

- Cache Addressing is based on the memory location physical address:
  
  ![Memory Location Physical Address Diagram]

  - Tag: used to find the right cache line by comparing all tag fields of the selected set (part of the cache entry structure)
  
  - Set: describes which cache set the data has been put in/found
  
  - Offset: specifies the desired data within the cache line
Cortex-M7 Caches Introduction

Cache Terminology: Associativity

• Direct Mapped Cache
  • A memory location in the main memory can occupy only one memory location in cache

• Fully Associative Cache
  • Any memory location in the main memory can occupy any location in the cache

• Set Associative Cache
  • Trade-off between a fully associative cache and a direct mapped cache

<table>
<thead>
<tr>
<th>Cache</th>
<th>Direct Mapped</th>
<th>Fully Associative</th>
<th>N-Way Set Associative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>Simple addressing system (Best Search Speed)</td>
<td>Flexible (Best Cache Hit ratio)</td>
<td>Best Compromise:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Fewer conflicts than Direct Mapped Cache</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Better search speed than Fully Associative</td>
</tr>
<tr>
<td>Drawbacks</td>
<td>High probability of conflicts resulting in performance drop</td>
<td>Complex Hardware to manage addressing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requires a big cache to reduce conflicts</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cortex-M7 Caches Introduction

SAM S70 / E70 Caches Implementation

• 16kB of Instruction and Data Caches with line-length of 32-bytes
  • I-Cache: two-way set-associative
  • D-Cache: four-way set-associative

• I-Cache: 8kB per way ⇔ 256 lines per way ⇔ 256 sets
• D-Cache: 4kB per way ⇔ 128 lines per way ⇔ 128 sets
Cortex-M7 Caches Introduction
SAM S70 / E70 Caches Interconnect

- Cached I/D are fetched from memory using the AXIM interface
Cortex-M7 Caches Introduction

Reading Policies (Cache Miss Case)

- All Cacheable locations are Read-Allocate
  - I- and D-Cache will always allocate on a read when a cache miss occurs
- When a cache miss occurs...
  - … a cache linefill is generated
  - Instructions/Data can be then fetched/read directly out of cache

- Cache Line Eviction
  - If all cache lines in a set are valid, to allocate a different address to the cache, the cache controller must evict a line from the cache

(*) Tag is only a part of the address of the data from main memory
Cortex-M7 Caches Introduction

Writing Policies: Write-Through (Cache Hit Case)

• Write is done synchronously both to Data Cache and Memory

• Example:
  • Word 0-7 are present in D-Cache
  • Word 2 value is updated (Cache Hit)

(*) Tag is only a part of the address of the data from main memory
Cortex-M7 Caches Introduction

Writing Policies: Write-Back (Cache Hit Case)

- Write is only made on Data Cache
  - Dirty bit for the line to indicate the main memory has not been updated

  1. Write Access
     - SRAM
     - D-Cache
     - TAG
     - FLAGS

     ![Diagram showing write access and cache hit]

  2. Write-Back
     - Cache Line is marked as ‘Dirty’

     ![Diagram showing cache hit]

  3. Line Eviction Request
     - SRAM
     - D-Cache
     - TAG
     - FLAGS

     ![Diagram showing cache line eviction]

  4. Write-Back
     - Cache Line is evicted

     ![Diagram showing cache line eviction]

- Write to the main memory is postponed until the cache line is evicted
## Cortex-M7 Caches Introduction

Write-Through vs. Write-Back

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Write-Through</th>
<th>Write-Back</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Solves the coherence problem between cache and main memory</td>
<td>Performant as not all write operations need to access main memory</td>
</tr>
<tr>
<td></td>
<td>Simple to implement</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drawbacks</th>
<th>Write-Through</th>
<th>Write-Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not performant as every writes will be done on the main memory</td>
<td></td>
<td>Cache coherency management is required when several masters try to access the same memory location (CPU and DMA as example)</td>
</tr>
<tr>
<td>Negates the main advantage of having a cache</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cortex-M7 Caches Introduction

Writing Policies (Cache Miss Case)

• Two policies in case of a Cache Miss:
  • Write Allocate:
    • Both the cache and the memory location are written
    • Good for performance as data is then available in the D-Cache for any future cache hits
  • No-write Allocate
    • Only the memory location is written / Cache is not affected
    • In this approach, only the reads are being cached
    • Good when data is written but not immediately used again

```c
for (int i = 0; i < SIZE; i++)
    a[i] = i;
```
Cortex-M7 Caches Introduction

SAM S70 / E70 Cortex-M7 Cacheability Attributes

• Cacheability attributes only relates to Normal memory
  • Applies to most memory used in a system (Flash, ROM, SRAM, DRAM)

• Supported Configurations
  • Write-back Cacheable, Read-Allocate, Write-Allocate
  • Write-back Cacheable, Read-Allocate only
  • Write-through Cacheable, Read-Allocate only
  • Non-Cacheable

• Write-back with Write-Allocate is usually chosen as this is the most performant policy
# Cortex-M7 Caches Introduction

## SAM S70 / E70 Default Address Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Device Type</th>
<th>Cacheability</th>
<th>SAM S70/E70 Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000-0x1FFFFFFF</td>
<td>Code</td>
<td></td>
<td>Write-Through</td>
<td>Flash, ROM</td>
</tr>
<tr>
<td>0x20000000-0x3FFFFFFF</td>
<td>SRAM</td>
<td></td>
<td>Write-Back with Write-Allocate</td>
<td>SRAM</td>
</tr>
<tr>
<td>0x60000000-0x7FFFFFFF</td>
<td>RAM</td>
<td>Normal</td>
<td>Write-Back with Write-Allocate</td>
<td>External Memories (EBI, SDRAM)</td>
</tr>
<tr>
<td>0x80000000-0x9FFFFFFF</td>
<td>RAM</td>
<td></td>
<td>Write-Through</td>
<td>Quad SPI Memory</td>
</tr>
</tbody>
</table>
Cache Coherence Concerns
Cache Coherence Concerns

About Cache Coherence

• Caches are usually transparent to the application programmer during normal operations

• BUT, a cache can become incoherent as soon as:
  • CPU and AHB masters share a cacheable memory region
  • A memory location in that region is cached and modified by at least one of them

• Typical cases:
  • DMA Writes Operations
  • DMA Reads Operations
Cache Coherence Concerns

DMA Writes Operations

• Conditions:
  • CPU initially accessed the SRAM and cached a region in Data Cache

1. A Peripheral writes data to the same region through a DMA channel

2. If CPU accesses the Data Cache at the same address, it will read the old data, not the new value made by the DMA access

[Diagram showing DMA write and CPU access scenarios with cache coherence issues]
Cache Coherence Concerns

DMA Reads Operations

- Conditions:
  - SRAM is cached with Write-Back cacheability attribute (best performance)

1. CPU is updating regularly data in cache
   - Only Data Cache is updated, not the main memory

2. Data read from the SRAM by the DMA will be accessing the old data, not the new as in the cache
Cache Coherence Concerns

About Cache Coherence Solutions

• Care must be taken to maintain coherence between the data cache and any data in memory accessed by any AHB masters

• Unfortunately, cache coherency is not handled by hardware at DMA/peripherals side on the Cortex-M7

• Various software solutions can be considered
Cache Coherence Concerns

Non-Optimal Solutions

• Disabling Data Cache
  • This would solve DMA Reads/Writes problems
  • But this will bring huge read latencies when accessing the sram and especially the flash (considering its wait states)

• Changing Cache Mode to Write-Through
  • Much better solution than disabling Data Cache as Data can be read at processor clock
  • But incomplete solution for cache coherence management as it only fixes DMA Reads concerns
  • It is also generally much less performant than Write-Back mode
Cache Coherence Management
Cache Coherence Management
Using Cortex-M7 Cache Maintenance Operations

• This solution simply requires the application to manage at run-time the caches using Cortex-M7 Cache maintenance operations
  • Controlled by dedicated Cortex-M7 registers

• Advantages:
  • Simple to implement
  • Portable over MCUs which don’t support TCM

• Drawbacks
  • Cache maintenance operations must operate on units of 32bytes
    • Buffers must be aligned to a cache line size boundary
  • Operations on whole cache (large buffers) could have an impact on overall performance

• Use Case: Handling of DMA Buffers
Cache Coherence Management

Cache Maintenance Operations

• Invalidate Cache operation
  • Allows to invalidate cache lines by clearing their ‘Valid’ flag bit

• Clean Cache (or Flush) operation
  • Allows to update the memory location with the current cache content

• I-Cache supported operations
  • Invalidate all / Invalidate by address

• D-Cache supported operations
  • Invalidate by address / Invalidate by Set/Way combination
  • Clean by address / Clean by Set - Way combination
  • Clean and Invalidate by address / Clean and Invalidate by Set - Way combination

• Functions which implement the use of cache maintenance operations are available in CMSIS (CMSIS-Core)
**Cache Coherence Management**

Using Cache Invalidate Operation to Handle DMA Writes

- **Conditions:**
  - CPU initially accessed the SRAM and cached a region in Data Cache

1. A Peripheral writes data to the same region through a DMA channel

2. A Cache Invalidate operation is performed
Cache Coherence Management

Using Cache Invalidate Operation to Handle DMA Writes (cont.)

• When CPU will access the data:
  • Data will be cached (Read-Allocate)
  • CPU will then read the up to date value

Data Coherency is guaranteed!
Cache Coherence Management
Using Cache Clean Operation to Handle DMA Reads

- Conditions:
  - SRAM is cached with Write-Back cacheability attribute (best performance)
  - Only Data Cache is updated, not the main memory

1. CPU is updating regularly data in cache
2. A Cache Clean operation is performed
3. Data read from SRAM by a DMA peripheral will be now coherent

Data Coherency is guaranteed!
Cache Coherence Management Limits
Cache Coherence Management Limits
GMAC Buffer Descriptors Handling

• Each GMAC TX/Rx Buffer Descriptor is composed of two words
  • Up to four descriptors can be cached in a single line

• GMAC Receive Buffer Descriptor Entry
  • Word 0 – Bit 0 is accessed by both the CPU and the GMAC AHB Master
    • CPU clears this bit when he has read the Rx buffer
    • GMAC sets this bit when it has received a frame and filled the Rx buffer
Cache Coherence Management Limits

GMAC Buffer Descriptors Handling (cont.)

• GMAC received a frame (bit 0 of right descriptor is then set by GMAC)

• A cache invalidate is performed to ensure data coherence when CPU will access that descriptor
Cache Coherence Management Limits
GMAC Buffer Descriptors Handling (cont.)

- Next CPU access to read any descriptor will trigger a cache line allocation.

Data Coherency is guaranteed!
Cache Coherence Management Limits

GMAC Buffer Descriptors Handling (cont.)

- CPU clears Word0-bit0 so that buffer can be used again
  - Main memory is not updated (Write-Back)

- A cache clean operation is performed to ensure data coherence

Data Coherency is guaranteed!
Cache Coherence Management Limits

GMAC Buffer Descriptors Handling (cont.)

• What does happen if before the Cache Clean operation, another buffer is received?

• Rx Buffer from descriptor 1 will be skipped!

• Use of cache maintenance operations does not guarantee here data coherence
  • A solution will be to use non-cacheable memory regions
**Cache Coherence Management Limits**

**Non-Cacheable Memory Regions Solution**

- **Use Case:**
  - Handling of Buffers with size smaller than cache line size such as GMAC descriptors

- **Advantages:**
  - Transparent for the application \( \Leftrightarrow \) no caches maintenance required

- **Drawbacks**
  - Impact on performance as data will be processed @150 MHz
  - Requires a more « complex » linker file

- **By default, only Normal, Non-shareable memory regions can be cached**
  - Possibility to configure using the MPU the memory region with the attribute ‘shareable’ to get the same effect

*Please refer to the Appendix to get a GMAC buffers’ descriptors implementation*
Cache Coherence Management Limits
Operations on Large Buffers

• What does happen if the buffers’ sizes are large and even exceed the cache size?

• Cache will have to perform frequent cache clean operations on the whole cache which will impact the application performance

• In such applications, use of cache maintenance operations may not guaranty good overall performance
Cache Coherence Management Limits

Data TCM (DTCM) Solution

• Use Cases:
  • Buffers used by deterministic code and only accessed by the CPU
  • Buffers with size larger than Cache size

• Advantages:
  • No impact on performance (CPU clock)
  • Transparent for the application ⇔ no caches maintenance required
  • Solves the strong non-determinism of the Cache
  • More performant than Cache if dual issues accesses occur on both DTCM0/1 (only for non contiguous data)

• Drawbacks
  • More complex to implement in both the linker file as the application (TCM configuration, activation, copy Flash to TCM)
  • TCM management must be supported in case of RTOS use

Please refer to « How to configure the SAM S70 / E70 Cortex-M7 Tightly Coupled Memories » presentation for more information
Conclusion
Conclusion

About Data Coherence

• As soon as cache use is required, data coherence must be guaranteed using cache maintenance operations

• There are still cases where data cache use may not be the right solution
  • GMAC descriptors, large buffers handling...

• Solutions still exist by:
  • Using Non-Cacheable Memory Regions
  • Using Data TCM

• Their choice will strongly depend on the application context, the peripherals used, the buffers size...
  • Usually a mix of them
Appendix
Appendix

Atmel Software Package Use Cases Implementations

• SPI Example: examples\spi\build\studio

• In SPI DMA driver (spi_dma.c):
  • DMA Writes to SRAM (SPI Receive Callback) are handled by a Cache Invalidate Operation

```c
static void SPI_Rx_Cb(uint32_t channel, Spid *pArg)
{
    /* Release the DMA channels */
    XDMAD_FreeChannel(pArg->pXdmad, spiDmaRxChannel);
    XDMAD_FreeChannel(pArg->pXdmad, spiDmaTxChannel);
    SCB_InvalidateDCache_by.Addr((uint32_t *)pArg->pCurrentCommand->pRxBuff,
                              pArg->pCurrentCommand->RxSize);
}
```

• DMA Reads to SRAM (DMA Start Transfer) are handled by a Cache Clean Operation

```c
uint32_t SPI_SendCommand(Spid *pSpid, SpidCmd *pCommand)
{
    /* Enables the SPI to transfer and receive data. */
    SPI_Enable (pSpidHw);
    SCB_CleanDCache_by.Addr((uint32_t *)pCommand->pTxBuff, pCommand->TxSize);

    /* Start DMA 0(RX) & 1(TX) */
    if (XDMAD_StartTransfer(pSpid->pXdmad, spiDmaRxChannel))
        return SPID_ERROR_LOCK;
}
```
Appendix

Atmel Software Package Use Cases Implementations

• Ethernet GMAC uip Hello World Example:
  examples_ethernet\gmac_uip_helloworld\build\studio

• In GMAC driver (gmac_tapdev.c):
  • GMAC Tx/RX descriptors are declared in a RAM section which is not cached:

```c
/** TX descriptors list */
COMPILER_SECTION(".ram_nocache")
COMPILER_ALIGNED(32) static sGmacTxDescriptor gTxDs[TX_BUFFERS], gDummyTxDs[DUMMY_SIZE];

/** RX descriptors list */
COMPILER_SECTION(".ram_nocache")
COMPILER_ALIGNED(32) static sGmacRxDescriptor gRxDs[RX_BUFFERS], gDummyRxDs[DUMMY_SIZE];
```

• Dedicated ram_nocache region defined in same70q21flash.ld script
  (MPU configuration in board_lowlevel.c):

```c
/* Memory Spaces Definitions */
MEMORY
{
  rom (rx) : ORIGIN = 0x00400000, LENGTH = 0x00200000
  ram (rwx) : ORIGIN = 0x20400000, LENGTH = 0x00050000

  ram_nocache (rwx) : ORIGIN = 0x2045f000, LENGTH = 0x00001000
}
```

```c
dwRegionBaseAddr =
  SRAM_NOCACHE_START_ADDRESS |
  MPU_REGION_VALID |
  MPU_NOCACHE_SRAM_REGION;   //11

dwRegionAttr =
  MPU_AP_FULL_ACCESS |
  INNER_OUTER_NORMAL_NOCACHE_TYPE(SHAREABLE) |
  MPU_CalMPURegionSize(NOCACHE_SRAM_REGION_SIZE) |
  MPU_REGION_ENABLE;
```
Appendix

References

• ARM®v7-M Architecture Reference Manual (ARM DDI 0403E.b)

• ARM® Cortex®-M7 Processor Revision r0p2 Technical Reference Manual (ARM DDI 0489B)

• SAM E70 Atmel | SMART ARM-based Flash MCU DATASHEET (Atmel-11296C-ATARM-SAM E70-Datasheet_19-Jun-15)

• SAME70-XPLD Atmel Studio Software Package 1.5: http://www.atmel.com/tools/ATSAME70-XPLD.aspx