The TM3270 Media-Processor Data Cache

Jan-Willem van de Waerdt*+, Stamatis Vassiliadis+, Jean-Paul van Itegem*
and Hans van Antwerpen*

*Philips Semiconductors
San Jose, CA, USA

+Delft University of Technology Delft,
The Netherlands
Stamatis@dutepp0.et.tudelft.nl

Abstract

*This paper describes the (micro-) architecture of the TM3270 data cache. We present the cache parameters, such as cache size, associativity, line size and cache policies. We describe the data cache pipeline partitioning and the cache memory structure organization. We introduce "collapsed" and "two-slot" load operations. Furthermore, we introduce a combined software/hardware based technique for prefetching of data into the cache. We use an MPEG2 encoder application for a quantative evaluation of architectural aspects such as data prefetching and show that MPEG2 encoding at 352*288 resolution (CIF) at 25 frames per second can be performed in 33.3 MHz.*

1. Introduction

The TM3270 media-processor has been designed having in mind potentially diametrically opposite constraints, such as *programmability*, *adequate performance*, and embedded processor requirements for the cost driven consumer market: *small size* and *low power*. The processor provides the performance for multi-standard video and audio (en/de)-coding (MPEG2, MPEG4, DivX, H.264/AVC [1]) at standard resolution. Codecs rely on standardization to ensure inter-operability, and therefore, the definition of these codecs will generally not change over time. However, algorithmic innovations in the area of e.g. motion estimation may be applied to allow for a more efficient video encoder implementation. Programmability provides flexibility, which can be exploited in different ways. It enables algorithmic changes after design, multiple algorithms can be mapped to the same architecture, faster time-to-market, etc.. Designing processors for the cost-driven consumer market poses the constraint of getting sufficient performance out of a minimal silicon area possibly with low power consumption. As all processor resources, the data cache is subject to this constraint. Typically, media-processors have an abundance of computational resources; in today's processing technologies adders and multipliers are cheap. We found, however, that media-processor performance is heavily dependent on data cache performance. The important question is no longer *"how many operations can be performed per processor cycle"*, but is *"how do we deliver data to the operators in a timely fashion"*. In other words, like general-purpose processors, media-processors suffer from the memory wall problem [2].

In this paper we discuss (micro-) architecture aspects of the TM3270 data cache. We use an MPEG2 encoder for a quantative evaluation of some of these aspects. Earlier evaluations, using somewhat simpler

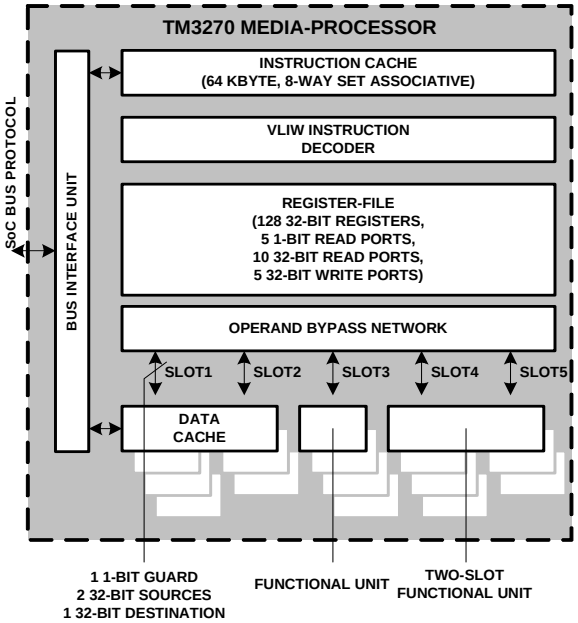

Figure 1. TM3270 media-processor.

applications, can be found in [3, 4]. A more elaborate study into the use of new multimedia operations to optimize MPEG2 encoder performance can be found in [5]. The remainder of this paper is organized as follows. In Section 2, we give an overview of the TM3270 media-processor architecture. In Section 3, we describe the data cache (micro-) architecture. In Section 4, we describe our performance evaluation environment, and the results of performance measurements. Finally, in Section 5, we present our conclusions.

2. TM3270 media-processor overview

The TM3270 media-processor (Figure 1) is backward source code compatible with the TriMedia architecture. An overview of the TriMedia architecture can be found in [6, 7]. The processor is used as an embedded processor in a *System-on-a-Chip* (SoC). The processor is implemented as a fully synthesizable design using a standard-cell logic library and single-ported SRAMs, allowing for fast process technology mapping. The processor achieves a frequency of 450 MHz in a 90 nm process technology optimized for low power consumption, and measures 8.1 mm^2. The TM3270 has a 32-bit VLIW architecture. A VLIW instruction may contain up to five operations. Each of these operations may be guarded; i.e. their execution can be made conditional on the value of a guard register. SIMD arithmetic and shuffle operations allow for efficient manipulation and re-organization of 8-, and 16-bit data types. Floating-point operations comply with the IEEE-754 standard. Operations are grouped into functional units, and most functional units have multiple instantiations. Most functional units are fully pipelined, allowing for back-to-back issue of operations. The simple arithmetic functional unit has five instantiations, so up to five simple arithmetic operations, e.g. additions, can be issued every cycle. The TM3270 supports *two-slot* operations, which are executed by functional units that are located in two neighboring issue slots. These functional units provide twice the register-file bandwidth, when compared to ordinary functional units. As a result, two-slot operations may have up to four 32-bit sources, and may produce up to two 32-bit destinations.

3. Data cache (micro-) architecture

This section describes some (micro-) architecture aspects of the TM3270 data cache. The TM3270 has a 128 Kbyte data cache, which is organized as a 4-way set associative cache with 128 byte cache lines. The cache has a least-recently-used (LRU) replacement policy, a copy-back write policy, and an allocate-on-write-miss policy. The allocate-on-write-miss policy reduces the write miss penalty, and results in less bandwidth to off-chip memory. A cache byte-validity structure is used to keep track of the validity of the individual bytes in the cache. When an allocated cache line is victimized, only the validated bytes are copied back to the off-chip memory. The TM3270 and the SoC memory bus system (on-chip infrastructure and external SDRAM interface) support the transfer of cache lines with byte-validity indicators.

3.1. Load and store operations

Table 1. Some of the TM3270 ISA's load and store operations (big endian description).

Operation	Description	Issue slot(s)	Latency
ST32D (displacement) src1 src2; Semantics: Store 32-bit word.	Mem[src2 + displacement] = src1[31:24]; Mem[src2 + displacement + 1] = src1[23:16]; Mem[src2 + displacement + 2] = src1[15:8]; Mem[src2 + displacement + 3] = src1[7:0];	4 or 5	not applicable

LD32R
src1 src2 -> dest1;
 	
Semantics: Load 32-bit word.
	dest1[31:24] = Mem[src1 + src2];
	dest1[23:16] = Mem[src1 + src2 + 1];
	dest1[15:8] = Mem[src1 + src2 + 2];
	dest1[7:0] = Mem[src1 + src2 + 3];
		4	4

TWOSLOT_LD32R
src3 src4 -> dest1 dest2;
 	
Semantics: Two-slot load operation; load two 32-bit words.
	dest1[31:24] = Mem[src3 + src4];
	dest1[23:16] = Mem[src3 + src4 + 1];
	dest1[15:8] = Mem[src3 + src4 + 2];
	dest1[7:0] = Mem[src3 + src4 + 3];
	dest2[31:24] = Mem[src3 + src4 + 4];
	dest2[23:16] = Mem[src3 + src4 + 5];
	dest2[15:8] = Mem[src3 + src4 + 6];
	dest2[7:0] = Mem[src3 + src4 + 7];
		4 and 5	4

LD_PACKFRAC8
src1 src2 -> dest1;
 	
Semantics: Collapsed load operation; load combined with two-taps filter function.
	data0 = Mem[src1]; data1 = Mem[src1 + 1]; data2 = Mem[src1 + 2];
	data3 = Mem[src1 + 3]; data4 = Mem[src1 + 4]; data5 = Mem[src1 + 5];
	data6 = Mem[src1 + 6]; data7 = Mem[src1 + 7];
	dest1[31:24] = (data0*(16-src2[3:0]) + data1*src2[3:0] + 8) / 16;
	dest1[23:16] = (data2*(16-src2[3:0]) + data3*src2[3:0] + 8) / 16;
	dest1[15:8] = (data4*(16-src2[3:0]) + data5*src2[3:0] + 8) / 16;
	dest1[7:0] = (data6*(16-src2[3:0]) + data7*src2[3:0] + 8) / 16;
		5	6

Table 1 defines some of the TM3270 load and store operations. The cache supports penalty-free (no stall cycles) non-aligned accesses. A store can be issued in both slot 4 or 5. A load can only be issued in slot 5. The *two-slot* TWOSLOT_LD32R load operation increases load bandwidth. It is issued in slots 4 and 5 (but the cache access path is restricted to slot 5), and retrieves two sequential 32-bit words from memory into two destination registers. The *collapsed* LD_PACKFRAC8 load operation is issued in slot 5. It is a new operation that involves memory collapsing rather than the ALU collapsing presented in [8]. This operation retrieves 8 bytes from consecutive memory addresses, and performs a filter function. It can be used to calculate image pixels at horizontal fractional positions. It finds use in e.g. the motion estimation kernel, which constitutes a significant computational part of video encoder applications [3, 5].

Figure 2 gives an overview of the data cache pipeline, partitioned into stages A through F. Normal load operations have a 4-cycle latency and produce a result in stage D. Collapsed load operations have a 6-cycle latency and produce a result in stage F. Stage A calculates the effective address for both the first and last byte referenced by an operation (Figure 2: *addr_lo* and *addr_hi*). Stage B performs access arbitration to the cache tag and data memory structures. Although the functionality provided by this stage is limited, the delay in this stage is significant. This is because a large amount of relatively wide address and data busses need to be multiplexed and routed to the different SRAMs. Furthermore, the large SRAM setup time extends their presence from stage C into stage B. Stage C contains the cache tag and data memory structures (LRU and byte validity structures are not depicted). The data memory structures have a maximum clock-frequency that is close to the processor clock-frequency. The tag memory structures are somewhat faster, and allow for the inclusion of tag comparison logic in stage C. Stage D contains the data cache way selection logic, and includes two operation control state machines for issue slots 4 and 5. Stages E and F contain the filter bank for collapsed load operations. On the right side of the pipeline we find the *cache write buffer* (CWB), which is used to keep pending stores to the data cache.

The cache memory structures use single ported SRAMs with bit write functionality, to allow for a selective update of memory bits as identified by a bit mask. The available SRAMs have a maximum data width restriction of 128 bits.

In stage B, store operations request access to the tag memory structure, but not to the data memory structure, since stores do not produce a register operand result. Load operations request access to both the tag and data memory structures. Since load operations are only supported in slot 5, the data memory structure access path is restricted to slot 5. To support two simultaneous stores, both slot 4 and 5 have a dedicated copy of the tag memory structure. Note that to support two simultaneous loads (without stall cycles), duplication of the memory structure would be required [9]. In stage D, the control state machines act upon the retrieved cache control information, such as the cache hit signal. For loads, the validity of the requested bytes needs to be checked, somewhat complicating the generation of the hit signal (when compared to a cache without byte-validity). In case of a load hit, way selection is performed on the retrieved cache data. In case of a load miss, a cache line is retrieved from off-chip memory by the refill unit. In case of a store hit, data is sent to the CWB. In case of a store miss, a cache line is allocated.

3.2. SRAM organization

Given the architectural cache parameters (128 Kbyte, 4-way set associativity, and 128 byte line size), there are multiple possibilities of organizing the cache. Figure 3 shows the organization of tag information, cache line byte elements, and byte validity bits.

Issue slots 4 and 5 have a dedicated tag memory structure, consisting of two exact copies of the cache tags. The SRAMs are indexed with the set address, and each entry contains the tag information of the four ways within the selected set. The tag information consists of a cache line valid bit, the cache line tag, and a cache line prefetch bit.

The data memory structure is partitioned into four separate SRAMs of 128 bits wide. The SRAMs are indexed with bits 14 downto 4 of a 32-bit address A, and each entry contains cache line data of the four ways. The SRAM partitioning is based on bits 3 and 2 of the address; all byte elements that share address bits 3 and 2 reside in the same SRAM. As a result, the 128 bytes of a cache line are located in 8 entries of each SRAM, and cache line bytes are SRAM interleaved at a granularity of 4 byte elements.

The byte-validity memory structure has a similar organization as the data memory structure. It is partitioned into four separate SRAMs of 64 bits wide. The 128 byte-validity bits of a cache line are located in 2 entries of each SRAM. Note that a cache line byte and its corresponding valid bit are located in corresponding SRAM memories, since both structures are partitioned based on bits 3 and 2 of the address. As a result, the access arbitration for the two structures can be shared.

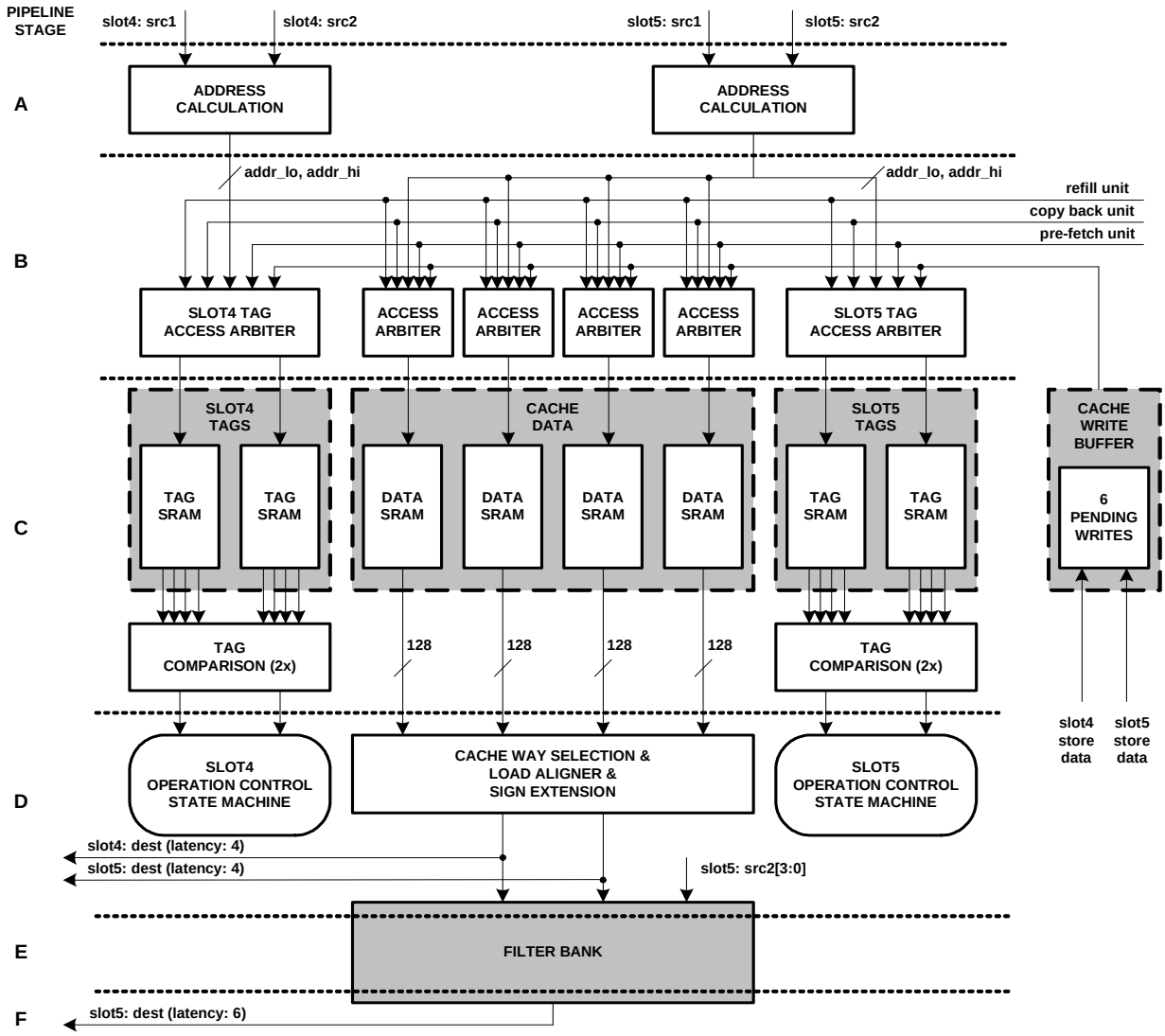

Figure 2. TM3270 data cache pipeline.

We illustrate the SRAM organization by means of an example. Consider a non-aligned 32-bit load operation from address A=0x000000ff. The set address is 0x1 for *addr_lo*, and 0x2 for *addr_hi*. The set addresses are used to index the tag SRAMs. A tag memory structure has two exact copies of the cache tags to cope with references that cross a cache line boundary. The tag is 0 for both *addr_lo* and *addr_hi*. Address bits A[3:2] determine which of the data and validity SRAMs need to be accessed. Data SRAM "11" is indexed with 0x00f, and data SRAM "00" is indexed with 0x010. Validity SRAM "11" is indexed with 0x003, and validity SRAM "00" is indexed with 0x004.

Note that the data SRAM organization provides the simultaneous (single cycle) access of up to 16 sequential bytes of a cache line. This allows for an 8-cycle cache line update by refill and prefetch units, and for an 8-cycle cache line extraction by the copy back unit. By limiting the amount of cycles these units need for cache line access, the interference with load and store operations is reduced, preventing data cache stall cycles. The byte validity organization provides the simultaneous access of up to 64 byte valid bits of a cache line (half of the total cache line byte validity bits), allowing for a 2-cycle cache line allocation.

Our data memory structure is partitioned into four separate SRAMs. Increasing the amount of SRAMs can increase the bandwidth to this structure, which would allow for more efficient cache line update and extraction. However, layout and routing is complicated as the amount of SRAMs increases. Furthermore, doubling the amount of SRAMs (each with half the capacity) more than doubles the silicon area, due to the

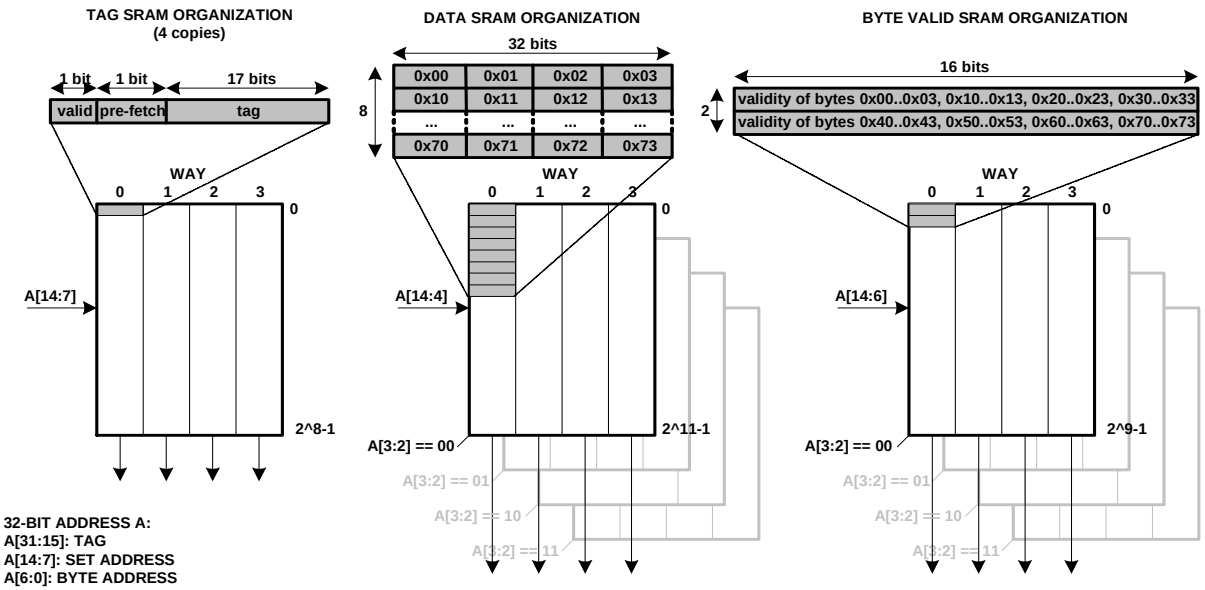

Figure 3. Data cache SRAM organization.

overhead in terms of address decoders and sense amplifiers in SRAM design.

3.3. Memory access arbitration

Figure 2 shows that separate memory arbiters control the access to cache tag and data memory structures. Furthermore, each of the four SRAMs in the data memory structure has its dedicated arbiter. This allows for low granularity access, resulting in high cache efficiency. For example, a 32-bit store in slot 4, a 32-bit load from address 0x0000009c in slot 5, and two 32-bit CWB updates to addresses 0x00000004 and 0x00000008, can all be granted access to the cache in a single cycle.

Besides the low granularity access control, cache efficiency is affected by the arbiter priority setting. We distinguish five separate entities that may request cache access: the issued operations, the CWB, the refill unit, the copy back unit and the prefetch unit. All may request access to the memory structures at the same point in time, and only one can be granted access (apart from simultaneous operation and CWB access to mutually exclusive data SRAMs).

We decided upon the following priority setting for the data memory structure arbitration in *normal operation mode* (listed in decreasing priority):

- Copy back unit
- Refill unit
- Issued operations
- CWB
- Prefetch unit

The rationale is as follows. A copy back operation has the highest priority, because a victimized cache line frees up a cache line location for a cache line that missed in the cache. A refill operation has the second highest priority. This operation includes both the allocation and retrieving of a cache line. Both will stall the processor till completion. Next in line are the issued operations. They get the highest priority as long as no copy back or refill operation is required. Since stores are kept pending in the CWB, their priority is lower than that of the issued operations. Lowest priority is given to the prefetch unit. Prefetches retrieve cache lines based on anticipated *future* use, and are typically not stalling the processor. When the issued operations are granted access to the data memory structure, the low granularity access control grants the CWB access to those SRAMs that are not required by the issued operations.

The priority setting may be changed in the following exceptional situations:

- The CWB is full, and required by a store.
- A load has an address conflict with a pending store.
- The prefetch unit raises its priority.

The first and second situations cause the CWB to have the highest priority. When the first situation occurs, a CWB entry needs to be freed up for a new store operation, so at least one of the pending store data elements needs to be put in the data memory structure. Likewise, the second situation can only be resolved by putting the data of the conflicting pending store in the data memory structure (data forwarding from the CWB is not supported). The details of the third situation and

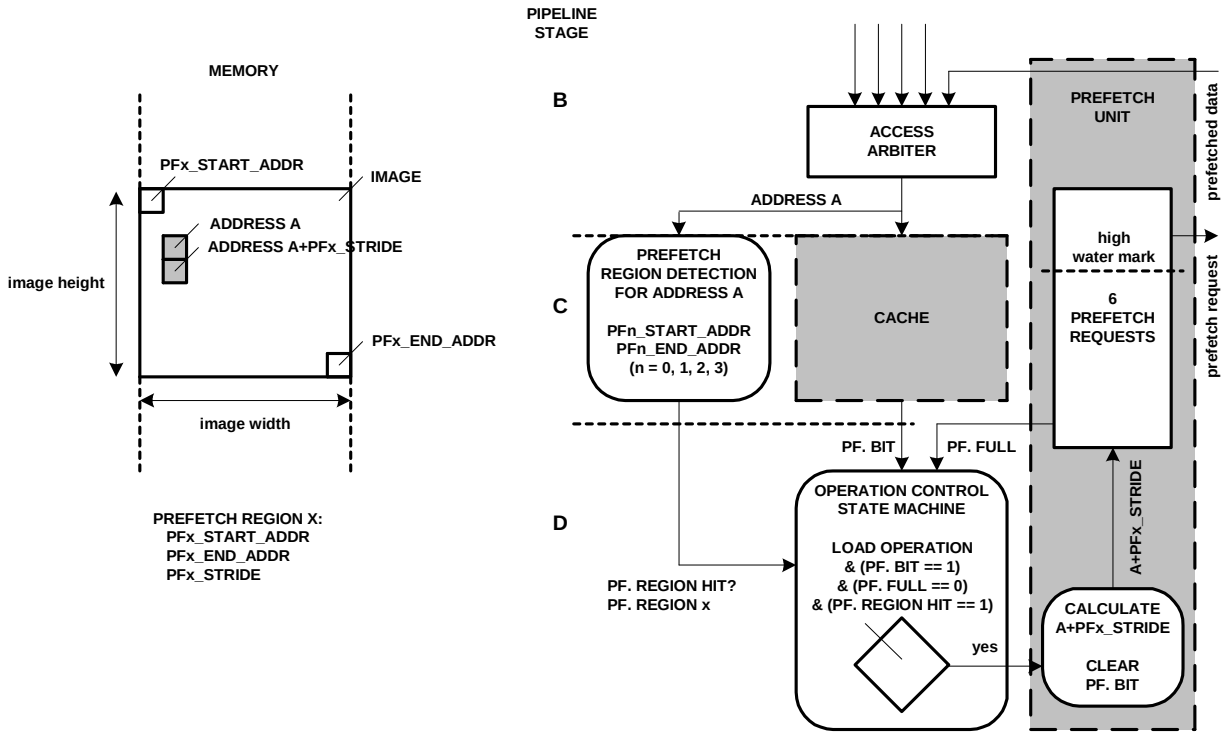

Figure 4. Region based prefetching. Left: an architectural perspective, right: a design perspective.

the effect on the priority setting are described in the next section.

3.4. Prefetching

Our prefetch approach is based on memory regions, and allows for a prefetching pattern that reflects the access pattern of a data structure mapped onto a certain address space (Figure 4). The TM3270 supports four separate memory regions. The identification of these memory regions and the required prefetch pattern is under *software control*, and defined by the following parameters (n = 0, 1, 2, 3):

- *PFn_START_ADDR*
- *PFn_END_ADDR*
- *PFn_STRIDE*

The first two parameters, *PFn_START_ADDR* and *PFn_END_ADDR*, are used to identify a memory region. The third parameter, *PFn_STRIDE*, is used to specify the prefetch pattern for the associated region. When the processor *hardware* detects a load from an address *A* within a prefetch region *x*, a prefetch request for address *A+PFx_STRIDE* is sent to the prefetch unit, if the prefetch address is not yet present in the cache. Note that by setting the prefetch pattern to the cache line size of 128 bytes, traditional next-sequential cache line prefetching is implemented. Prefetched data is put into the cache. The large data cache capacity of 128 Kbyte and the 4 way set associativity make it unlikely that useful data is victimized. Furthermore, no dedicated prefetch storage structures, such as stream buffers or stream caches, are required [10, 11].

The prefetch unit has a buffer for up to six outstanding prefetch requests. When the prefetch request buffer is almost continuously at the maximum of its capacity, the effectiveness of prefetching may decrease. Therefore, whenever a prefetch request turns into a demand miss, or when the requests buffer exceeds a *high water mark* (more than four entries occupied) the arbitration priority of the prefetch unit is raised, resulting in a *prefetch operation mode* for the data memory structure:

- Copy back unit
- Refill unit
- Prefetch unit
- Issued operations
- CWB

The relative priority of issued operations and the prefetch unit has changed, and prefetching is accelerated.

In our 5-issue slot processor, it is not uncommon for almost every VLIW instruction to contain a load operation. Every load operation that accesses a prefetch memory region may potentially trigger a prefetch request. As a result, for every load from address A, a prefetch address A+PFx_STRIDE may have to be

calculated, and the presence of this prefetch address in the cache needs to be checked. A traditional approach would require a dedicated tag SRAM to provide the required bandwidth to check the presence of prefetch addresses in the cache. We decided upon a different approach that does not require a dedicated tag SRAM, and is therefore cheaper to implement. Whenever a cache line is created in the data cache, either by a refill or prefetch request, a cache line prefetch bit is set to '1'. This introduces an overhead of a single bit to every cache line (Figure 3). A prefetch will only be considered for those load operations that access a prefetch region and have their prefetch bit equal to '1' (for the load address A). Furthermore, a prefetch request will only be sent to the prefetch unit, when its prefetch request buffer is not full; i.e. its six entries are not all occupied. Only when these conditions are met, will the prefetch bit of the cache line be set to '0'. Our approach has the following advantage. Every cache line will give rise to at most one prefetch request, which prevents the possibility of multiple duplicate prefetch requests for the same cache line. Furthermore, the amount of checks for the presence of prefetch addresses in the cache is reduced. As a result, the existing tag SRAMs can be used to check the presence of prefetch addresses in the cache, without any significant performance penalty.

Since prefetches are only requested when the prefetch request buffer is not full, prefetch requests will not get lost. When they do not make it into the prefetch unit, the prefetch bit will remain '1', and a future load to the same cache line will initiate the same prefetch request if the buffer has freed up one of its entries. This try-and-retry mechanism is especially useful for memory access patterns that initiate a burst of prefetch requests in a relatively short period of time that would overflow the prefetch request buffer.

4. Performance evaluation

An MPEG2 encoder is used to evaluate data cache performance. We started with a plain-vanilla C-implementation, and invested 6 man weeks to optimize the implementation for the TM3270. We have not undertaken any optimizations that would compromise standard compliancy. Most of the optimizations involve the selection of custom operations to reduce computational complexity. We encoded the "Foreman" sequence at a 352*288 resolution (CIF) and a 4:2:0 format at 25 frames per second. The target bitrate was set at 500,000 bits per second. The bitrate is controlled through the macroblock quantization factor. An MPEG2 "group of pictures" (GOP) includes 12 frames, and the frame types in display order are given by the pattern: I-B-B-P-B-B-P-B-B-P-B-B. The motion estimation for P-frames evaluates 17 motion vector candidates. For B-frames, motion estimation is performed with two reference images. To balance the computational complexity of B-frames with that of P-frames, B-frames evaluate 10 motion vector candidates per reference frame.

Table 2.
VLIW and cycle counts for I-, P-, and B-frames, 0 additional memory delay cycles.

	I-frame	P-frame	B-frame	Average
Scenario A: 1 store per VLIW instruction, prefetching on				
VLIW instr.	1203043	1277629	1092035	1147684
Stall cycles	102488	187011	224582	205015
Cycles	1305531	1464640	1316617	1352699
Scenario B: 2 stores per VLIW instruction, prefetching on				
VLIW instr.	1187447	1255290	1065326	1122994
Stall cycles	91528	198640	229574	210336
Cycles	1278975	1453930	1294900	1333330
Scenario C: 2 stores per VLIW instruction, prefetching off				
VLIW instr.	1187437	1255280	1065315	1122983
Stall cycles	130120	219185	260061	239014
Cycles	1317557	1474465	1325376	1361997

For performance evaluation, a cycle-accurate C-model was automatically generated from the processor's Verilog HDL description. The 450 MHz processor (Figure 1) is attached through its bus interface unit to a 200 MHz 32-bit DDR SDRAM off-chip memory. We measured the impact of prefetching and the support of two stores per VLIW instruction on processor performance. Table 2 gives the measurement results for I-, P-, and B-frames. The average column is calculated based on the frame type frequencies as defined by the GOP pattern. The support of two, rather than one, store operations per VLIW instruction improves performance by 1.43% (based on the average cycle counts for scenarios A and B). The additional area for duplicating the tag memory structures is 0.142 mm^2 and additional power consumption is negligible. With support for two store operations and prefetching turned on, the average frame cycle budget of 1,333,330 cycles translates into a 33.3 MHz load for encoding a CIF sequence at 25 frames per second. At a processor frequency of 450 MHz, this leaves plenty of available processor cycles for other functionality.

The performance impact of prefetching is dependent on off-chip memory latency (Figure 5). Off-chip memory latency was artificially increased, by adding delay cycles to the memory path between the processor's bus interface unit and the off-chip memory controller. Note that the additional delay cycles are in the off-chip memory clock domain (200 MHz). As the

latency increases the absolute performance difference between scenarios B and C grows, which illustrates the ability of prefetching to hide memory latency. For 100 additional delay cycles, prefetching improves performance by 20%. The steepness of the performance curves reflects the dependency on memory latency.

5. Conclusions

We have presented the (micro-) architecture of the TM3270 data cache. The discussion of the memory structures and pipeline shows that apparent "design details" such as SRAM organization and cache arbitration impact cache architecture and performance. E.g. SRAM organization influences the ability to simultaneously support a high cache associativity (4-way), fast cache line update (8 cycles for a 128-byte line), and penalty-free non-aligned access. Furthermore, counter-intuitive architectural aspects (from a performance perspective), such as the support for two stores, but only a single load per VLIW instruction, are explained by a design aspect such as cost.

The support of two store operations and data prefetching was evaluated using an MPEG2 encoder application. The support of two, rather than one, store operations improves performance by 1.43%. For 100 additional delay cycles, prefetching improves performance by 20%. Furthermore, the benefit of prefetching for increased off-chip memory latency was evaluated.

6. References

[1] I.E.G. Richardson, "H.264 and MPEG-4 video compression, video coding for next-generation multimedia", *Wiley*, 2003.

[2] W. Wulf, and S. McKee, "Hitting the memory wall: implications of the obvious", *ACM SIGARCH Computer Architecture News*, vol. 23, issue 1, pp. 20-24, 1995.

[3] J.W. van de Waerdt, J.P. van Itegem, G. Slavenburg, and S. Vassiliadis, "Motion estimation performance of the TM3270", *ACM Symp. on Applied Computing*, pp. 850-856, March 2005.

[4] J.W. van de Waerdt, S. Vassiliadis, and E.W. Bellers, "Temporal video up-conversion on a next-generation media-processor", *Proc. of the 7th Int. Conf. on Signal and Image Processing*, August 2005.

[5] J.W. van de Waerdt, and S. Vassiliadis, "Instruction set architecture enhancements for video processing", *Proc. of the 16th Int. Conf. on Application-specific Systems, Architectures and Processors*, July 2005.

[6] S. Rathnam, and G. Slavenburg, "An architectural overview of the programmable multimedia processor, tm-1", *Proc. of the COMPCON '96*, pp. 319-326, February 1996.

[7] T. Halfhill, "Philips powers up for video", *Microprocessor Report*, http://www.mpronline.com/, November 2003.

[8] S. Vassiliadis, J. Phillips, and B. Blaner, "Interlock collapsing ALU's", *IEEE Trans. on Computers*, vol. 42, issue 7, pp. 825-839, July 1993.

[9] J.A. Rivers, G.S. Tyson, E.S. Davidson, and T.M. Austin, "On high-bandwidth data cache design for multi-issue processors", *Proc. of the 30th Int. Symp. on Microarchitecture*, pp. 46-56, December 1997.

[10] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers", *Proc. of the 17th Int. Symp. on Computer Architecture*, pp. 364-373, May 1990.

[11] R.J. Eickemeyer, and S. Vassiliadis, "A load instruction unit for pipelined processors", *IBM Journal of Research and Development*, vol. 37, no. 4, pp. 547-563, July 1993.

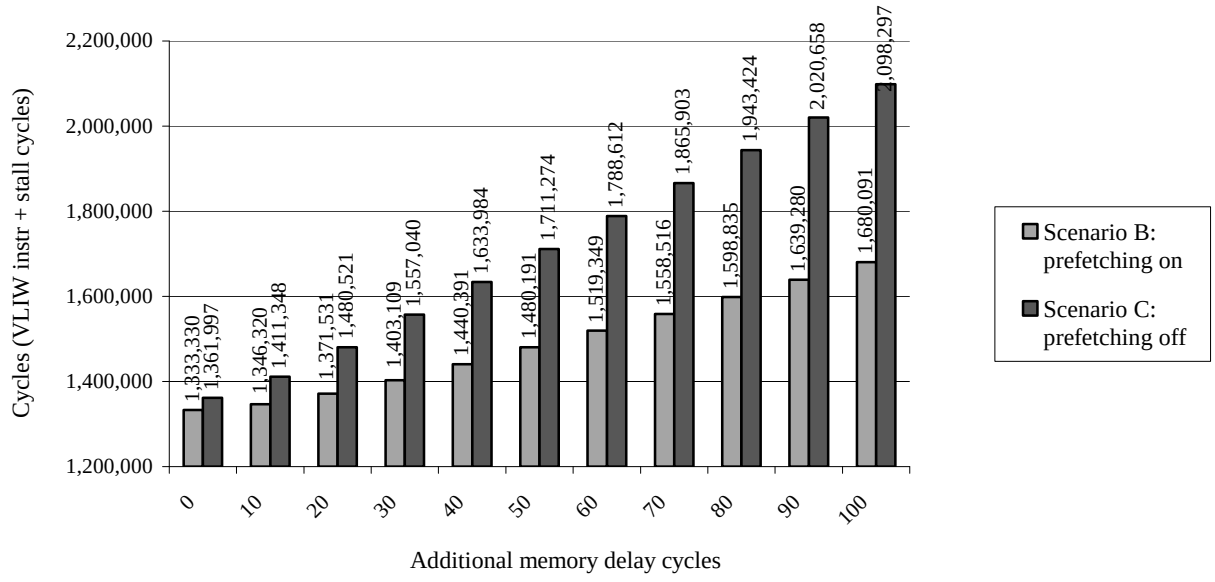

Figure 5. Off-chip memory latency. Delay cycles are in the 200 MHz off-chip memory clock domain.