The TM3270 Media-Processor

Jan-Willem van de Waerdt*+, Stamatis Vassiliadis+, Sanjeev Das*, Sebastian Mirolo*, Chris Yen*, Bill Zhong*, Carlos Basto*, Jean-Paul van Itegem*, Dinesh Amirtharaj*, Kulbhushan Kalra*, Pedro Rodriguez* and Hans van Antwerpen*

*Philips Semiconductors
San Jose, CA, USA

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

Abstract

We present the TM3270 media-processor, the latest TriMedia VLIW processor, tuned to address the performance demands of standard definition video processing, combined with embedded processor requirements for the consumer market. We discuss the architecture, implementation, and its first realization in a 90 nm process technology. The processor incorporates instruction set architectural (ISA) extensions and a load/store unit optimized for the video-processing domain. The ISA extensions improve the performance on video processing kernels. The data cache policies and prefetching techniques allow for efficient access to multimedia data. Finally, power consumption and performance data are presented.

1. Introduction

It is well established that high-frequency General-Purpose Processors (GPP) provide the computational performance to meet the requirements of the latest video standards, such as MPEG4, and H.264. They are easy to program, and appear to offer a steadily increasing performance level with each new processor generation. Furthermore, SIMD-style instruction set architecture (ISA) extensions have been added that target the multimedia domain, e.g. Intel's MMX extensions [2], and the AltiVec extensions to the PowerPC architecture [3]. Multiple approaches exist that suggest vector-processing enhancements to GPP architectures, see for example [4, 5, 6, 7, 8]. These enhancements try to exploit the assumed regular memory access pattern and streaming nature of multimedia applications. Typically, the enhancements are closely connected to the GPP, but have a dedicated access path to memory and their own register-file structure. The efficiency of these approaches relies on a certain regularity in memory accesses, and a stream-based processing of multimedia data. Whereas this may have been typical for older video codec standards, this assumption is less true for newer standards. As an example, consider the granularity at which video codecs transmit motion vector data. For MPEG2, a single motion vector is present for every 16x16 block of image pixels. For MPEG4, a motion vector may be present for every 8x8 block, and for H.264, a motion vector may be present for every 4x4 block. In general we can observe a decrease in block size and an increase in control overhead. Furthermore, the dependencies between blocks is increasing, which prohibits the parallel processing of multiple blocks, e.g. for H.264, processing a 4x4 block may require that the blocks to its left and above it have already been processed. It could be stated that video codecs are getting more control intensive. As a result, approaches that rely on stream-based processing on large vectors become less efficient.

It can be contented that in most cases silicon cost and power consumption of the GPP processor based approaches may prohibit successful application in the embedded processor market. A fixed function implementation in dedicated hardware is a different approach. It allows for a low cost implementation of a specific standard, but may be less efficient when a large variety of standards have to be supported. The increased multimedia workload, especially in the video processing domain, has given rise to its own class of processors: the media-processor. Unburdened by binary code compatibility issues, these new processors typically have a VLIW architecture to allow for a low cost silicon implementation. Examples are Texas Instruments' VelociTI architecture [16], Philips' TriMedia architecture [9], and Equator's MAP-CA [17]. These processors have been built from the ground up to address the requirements of video processing. Like GPPs, their ISA includes SIMD-style operations, but also their memory infrastructure has been optimized to address multimedia requirements, e.g. efficient support for nonaligned memory access, data prefetching and DMA-style memory transfers can be found in these processors. Whereas GPPs typically have a distinct register-file for SIMD-style operation, media-processors typically have a unified register-file

Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05)
0-7695-2440-0/05 $20.00 © 2005 IEEE

structure. Furthermore, media-processors have register-file sizes that exceed those of GPPs. As a result, a large data working set can be kept in registers, preventing the generation of load and store operations as a result of spilling due to register-pressure.

The programmability of media-processors gives a flexibility advantage over a dedicated hardware approach: it enables algorithmic changes after design, multiple applications can be mapped to the same platform, faster time-to-market, etc. Furthermore, their architecture allows for an efficient handling of both control- and media-processing tasks, as required by the latest video processing applications such as H.264.

This paper presents the TM3270, the latest Philips media-processor based on the TriMedia architecture. It addresses the requirement of multi-standard video processing (en-/de-coding) at standard resolution and the associated audio processing requirements. Additionally to adequate processing performance, area and power consumption were the main design constraints.

The remainder of this paper follows the separation of computer design into architecture, implementation, and realization, as introduced by Blaauw in [10]. Section 2 introduces the TM3270 architecture: the user view of the processor. This section includes some of the ISA enhancements. Sections 3 and 4 discuss implementation: the logical organization of the processor's inner structure. These sections include a description of the processor pipeline and load/store unit organization. Section 5 discusses realization: the physical mapping of the implementation on a certain process technology. This section includes a description of the processor floorplan, and presents area and power numbers. Section 6 presents some performance statistics. Finally, Section 7 presents the conclusions.

2. Architecture

The TM3270 is a VLIW-based media-processor, which is backward source code compatible with other processors in the TriMedia family [9]; i.e. C-code written for previous TriMedia processors can be re-compiled to run on the TM3270. Re-compilation is required, since binary compatibility is not guaranteed between all members of the family. Typically, the TM3270 is used as an embedded processor in a *System-on-a-Chip* (SoC). Table 1 gives an overview of the main architectural features.

Table 1. TM3270 Architecture

Architectural feature	Quantity
Architecture	5 issue slot VLIW guarded RISC-like operations
Pipeline depth	7-12 stages
Address width	32 bits
Data width	32 bits
Register-file	Unified, 128 32-bit registers
Functional units	31
IEEE-754 floating point	yes
SIMD capabilities	1 x 32-bit, 2 x 16-bit, 4 x 8-bit
Instruction cache	64 Kbyte, 128-byte lines, 8 way set-associative, LRU replacement policy
Data cache	128 Kbyte, 128-byte lines 4 way set-associative, LRU replacement policy, Allocate-on-write miss policy

2.1. Operation encoding

A VLIW instruction may contain up to five operations, which are template-based encoded in a compressed format to limit code size. Every VLIW instruction starts with a 10-bit template field, which specifies the compression of the operations in the *next* VLIW instruction. As a result, an instruction's compression template is available one cycle before the instruction's compressed encoding, which relaxes the timing requirements of the decoding process. Jump target VLIW instructions are not compressed and do not require an explicit template field in the preceding instruction. The 10-bit template field has five 2-bit compression sub-fields, which are related to the processor's issue slots 1 through 5. An issue slot's 2-bit compression field specifies the size of the operation encoding. Figure 1 gives an example of a VLIW instruction containing three operations in slots 2, 3, and 5. Issue slots 1 and 4 are not used, as specified by the "11" encoding of the related compression fields. Since issue slot 1 is not used, the first encoded operation is

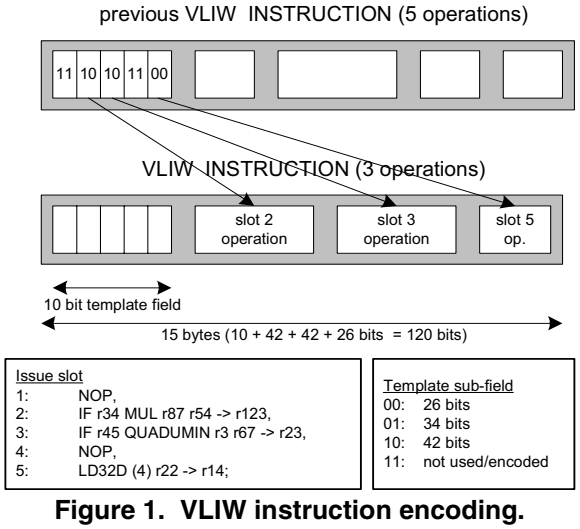

Figure 1. VLIW instruction encoding.

Table 2. TM3270 two-slot operations, collapsed load operation and CABAC operations.

Operation	Description	Issue slot(s) (latency)

SUPER_DUALIMIX
rsrc1 rsrc2 rsrc3 rscr4 -> rdest1 rdest2
	temp = rsrc1[31:16]*rsrc2[31:16] + rsrc3[31:16]*rsrc4[31:16];
	rdest1[31:0] = min (max (-2^31, temp), 2^31-1);
	temp = rsrc1[15:0]*rsrc2[15:0] + rsrc3[15:0]*rsrc4[15:0];
	rdest2[31:0] = min (max (-2^31, temp), 2^31-1);
		2 and 3
		(4)

SUPER_LD32R
rsrc3 rsrc4 -> rdest1 rdest2;
 	
Semantics: Two-slot load operation; load two 32-bit words, big endian.
	rdest1[31:24] = Mem[rsrc3 + rsrc4];
	rdest1[23:16] = Mem[rsrc3 + rsrc4 + 1];
	rdest1[15:8] = Mem[rsrc3 + rsrc4 + 2];
	rdest1[7:0] = Mem[rsrc3 + rsrc4 + 3];
	rdest2[31:24] = Mem[rsrc3 + rsrc4 + 4];
	rdest2[23:16] = Mem[rsrc3 + rsrc4 + 5];
	rdest2[15:8] = Mem[rsrc3 + rsrc4 + 6];
	rdest2[7:0] = Mem[rsrc3 + rsrc4 + 7];
		4 and 5
		(4)

LD_FRAC8
rsrc1 rsrc2 -> rdest1;
 	
Semantics: Collapsed load operation; load combined with two-taps filter function
	data0 = Mem[rsrc1]; data1 = Mem[rsrc1 + 1];
	data2 = Mem[rsrc1 + 2]; data3 = Mem[rsrc1 + 3];
	data4 = Mem[rsrc1 + 4];
	rdest1[31:24] = (data0*(16-rsrc2[3:0]) + data1*rsrc2[3:0] + 8) / 16;
	rdest1[23:16] = (data1*(16-rsrc2[3:0]) + data2*rsrc2[3:0] + 8) / 16;
	rdest1[15:8] = (data2*(16-rsrc2[3:0]) + data3*rsrc2[3:0] + 8) / 16;
	rdest1[7:0] = (data3*(16-rsrc2[3:0]) + data4*rsrc2[3:0] + 8) / 16;
		5
		(6)

SUPER_CABAC_STR rsrc1 rsrc2 rsrc4 -> rdest1 rdest2	rsrc1: DUAL16* (value, range) rsrc2: stream_bit_position rsrc4: DUAL16 (state, mps) < Functionality as described in Figure 2 > rdest1: stream_bit_position rdest2: bit	2 and 3 (4)
SUPER_CABAC_CTX rsrc1 rsrc2 rsrc3 rscr4 -> rdest1 rdest2	rsrc1: DUAL16 (value, range) rsrc2: stream_bit_position rsrc3: stream_data rsrc4: DUAL16 (state, mps) < Functionality as described in Figure 2 > rdest1: DUAL16 (value, range) rdest2: DUAL16 (state, mps)	2 and 3 (4)
*DUAL16 (a, b) = (a << 16) | (b & 0xffff)		

for issue slot 2. A VLIW instruction without any operations is efficiently encoded in 2 bytes, with a "11:11:11:11:11" template field. A VLIW with all operations of the maximum size of 42 bits is encoded in 28 bytes, with a "10:10:10:10:10" template field and 5 * 42 bits for the operation encoding. This compression scheme allows for an efficient encoding of code with a low amount of instruction level parallelism.

2.2. ISA enhancements

The TM3270 enhances the ISA of its predecessor, the TM3260, with roughly 40 new operations. The TM3260 finds commercial use in e.g. the PNX1500 IC [15]. In the following subsections we describe some of the new operations.

2.2.1. Two-slot operations

The concept of two-slot or super-operations was first introduced in [11], but only finds first application in the TM3270. Two-slot operations are executed by functional units that are situated in two neighboring issue slots. As a result, these operations have twice the register-file bandwidth, allowing for operations with up to four source operands, and up to two destination operands. We describe two of these operations: SUPER_DUALIMIX and SUPER_LD32R (Table 2). The SUPER_DUALIMIX operations perform a pairwise 2-taps filter on 16-bit signed values. Each of the two filter results is clipped to the signed 32-bit range, and returned in a separate destination register. The SUPER_LD32R operation retrieves two consecutive 32-bit values from memory. It has two source operands, which are encoded as part of the second operation in the operation pair, and two destination registers. The operation doubles the processor's bandwidth to data memory. The restriction to two consecutive address locations limits the applicability of this operation, when compared to two separate unrestricted 32-bit load operations. However, the SUPER_LD32R is easily supported by our cache implementation (as described in Section 4), whereas the support for two separate 32-bit load operations is more expensive [18]. Similar to our SUPER_LD32R operation, Texas Instruments' VelociTI architecture

```
LpsRangeTable[64][4]                                    /* range table for least probable symbol (LPS) */
MpsNextStateTable[64], LpsNextStateTable[64]/* MPS, LPS state transition tables */

biari_decode_symbol (      /* decodes a single binary value "bit" from the CABAC coded bitstream. */
  inout value,                                                          /* coding value, 10-bit value */
  inout range,                                                          /* coding range, 9-bit value */
  inout state,                                                          /* modeling context state, 6-bit */
  inout mps,                                                            /* modeling context MPS, 1-bit */
  in    stream_data,                                                    /* bitstream data */
  inout stream_bit_position,                                            /* bit position in "stream_data" */
  out   bit)                                                            /* decoded binary value */
{
  stream_data_aligned     = stream_data << stream_bit_position;
  range_lps               = LpsRangeTable[state][(range >> 6) & 3)];
  temp_range              = range - range_lps

  if (value < temp_range) {                                             /* MPS: most probable symbol */
    value  = value;
    range = temp_range;
    bit     = mps;
    mps   = mps;
    state  = MpsNextStateTable[state];
  } else {                                                              /* LPS: least probable symbol */
    value  = value - temp_range;
    range = range_lps;
    bit     = !mps;
    mps   = mps ^ (state != 0);
    state  = LpsNextStateTable[state];
  }

  while (range < 256) {      /* renormalization, at most 8 bits can be consumed */
    value                 =   (value << 1)
                            | ((stream_data_aligned >> 31) & 1);
    range                 <<= 1;
    stream_data_aligned <<= 1;
    stream_bit_position   += 1;
  }
}
```

Figure 2. Context-based adaptive binary arithmetic coding (CABAC), "biari_decode_symbol" function.

supports the LDDW operation to double the load bandwidth. However, the two destination registers of the LDDW operation are restricted to a pair of successive registers in the register-file, which puts additional constraints on the register allocator in the compiler/scheduler.

2.2.2. Collapsed load operations with interpolation

A significant part of the computational complexity of a video encoder is found in the motion estimation kernel. The TM3270 ISA includes collapsed load operations that involve memory collapsing rather than the ALU collapsing presented in [21]. The new operations perform a memory load with interpolation on the retrieved data, which allows for efficient calculation of pixels at fractional horizontal image positions. As a result, the computational complexity of motion estimation is significantly reduced [12]. The collapsed load operations combine the functionality of an ordinary load operation with a linear interpolation function, as defined by a fractional position. Table 2 gives the definition of the LD_FRAC8 operation. The operation retrieves *five* consecutive bytes from memory, performs a pair-wise interpolation based on a fractional position, and returns four interpolated byte values. A more traditional architecture would require at least two 32-bit loads to retrieve the five bytes from memory, and multiple arithmetic operations to perform the interpolation. Not only is the amount of required operations reduced, but also the register-file pressure is relaxed, which prevents spilling.

2.2.3. CABAC operations

A significant part of the computational complexity of the H.264/AVC video standard is found in the Context-Based Adaptive Binary Arithmetic Coding (CABAC) [18]. The intrinsic sequential behavior of the coding process prohibits an efficient implementation on a multi-issue SIMD processor. Performance evaluations of an optimized decoder indicate that a

Table 3. Performance measurements of the CABAC decoding process for I, P, and B-fields of a 4.5 Mbits/sec bitstream at standard resolution (60 720*240 NTSC fields/sec).

Field-type	Average bits/field	Non-optimized		Optimized (CABAC operations)		Speedup
		VLIW instr.	VLIW instr./bit	VLIW instr.	VLIW instr./bit	
I	215,408	4,535,945	21.1	2,686,787	12.5	1.7
P	103,544	2,901,381	28.0	1,799,559	17.4	1.6
B	153,035	1,439,821	33.8	951,545	22.3	1.5

significant part of the overall decoding time may be spent in the CABAC process. For I-fields, which require a relatively high amount of bits to encode a field, CABAC decoding may constitute up to 50% of the decoding time. To reduce these computational requirements, specific CABAC operations were introduced in the TM3270.

Figure 2 gives the biari_decode_symbol function, taken from the H.264 reference code. This function decodes a single binary value bit from a CABAC coded bitstream. Ideally, we would like to implement the above functionality with a single new operation. However, the amount of input and output function arguments exceeds the capability of our two-slot operations. Closer investigation of the function shows that by grouping of semantically related arguments as 16-bit sub-operands of a 32-bit operand, an implementation with two two-slot operations is possible. The value (10-bit value) and range (9-bit value) arguments are both related to a context, and are grouped in a two-way 16-bit representation. The state (6-bit value) and mps (1-bit value) arguments define the state of a probability model for a context, and are grouped in a two-way 16-bit representation. The other arguments are represented by dedicated operation operands. We introduce two new operations: SUPER_CABAC_CTX and SUPER_CABAC_STR (Table 2). The SUPER_CABAC_CTX operation calculates the new values of the context modeling: rdest1 contains (value, range) and rdest2 contains (state, mps). Note that for this calculation, all function input arguments are required: rsrc1 contains (value, range), rsrc2 contains stream_bit_position, rsrc3 contains stream_data and rsrc4 contains (state, mps). The SUPER_CABAC_STR operation calculates the new values related to the bitstream processing: rdest1 contains stream_bit_position and rdest2 contains bit. Note that for this calculation, only a subset of the function input arguments are required (stream_data is not required): rsrc1 contains (value, range), rsrc2 contains stream_bit_position, rsrc3 is not used and rsrc4 contains (state, mps).

We measured the performance of the complete CABAC decoding process (including decoder data structure maintenance and context computation) for a standard resolution 4.5 Mbits/sec bitstream with and without the use of the new CABAC operations. Table 3 gives the results for the different field types: the use of the new CABAC operations results in a speed up in the range of [1.5, 1.7].

2.3. Prefetching

Our prefetch approach is based on memory regions. It allows for a prefetching pattern that reflects the access pattern of a data structure mapped onto a certain address space. 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.

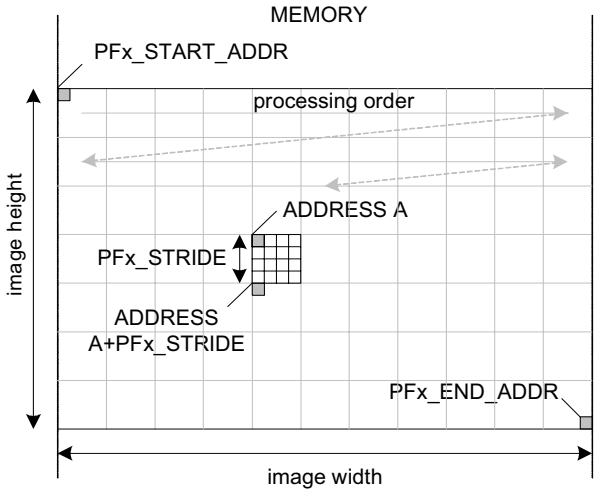

Figure 3. Memory region based prefetching.

Prefetched data is put directly into the data cache. The large data cache capacity of 128 Kbyte and its 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 [19], are required.

Traditional next-sequential cache line prefetching is realized by setting the prefetch pattern to the cache line size of 128 bytes. The effectiveness of the prefetch approach becomes apparent when we consider e.g. the block-based processing of an image (Figure 3). Assume an image of byte element in memory. The image is processed at 4x4 block-size granularity. Starting with the upper-left block, the blocks are processed in a left-to-right, and top-down order. The memory region (PFx_START_ADDR and PFx_END_ADDR) is set to include the image, and the associated prefetch pattern (PFx_STRIDE) is set to the

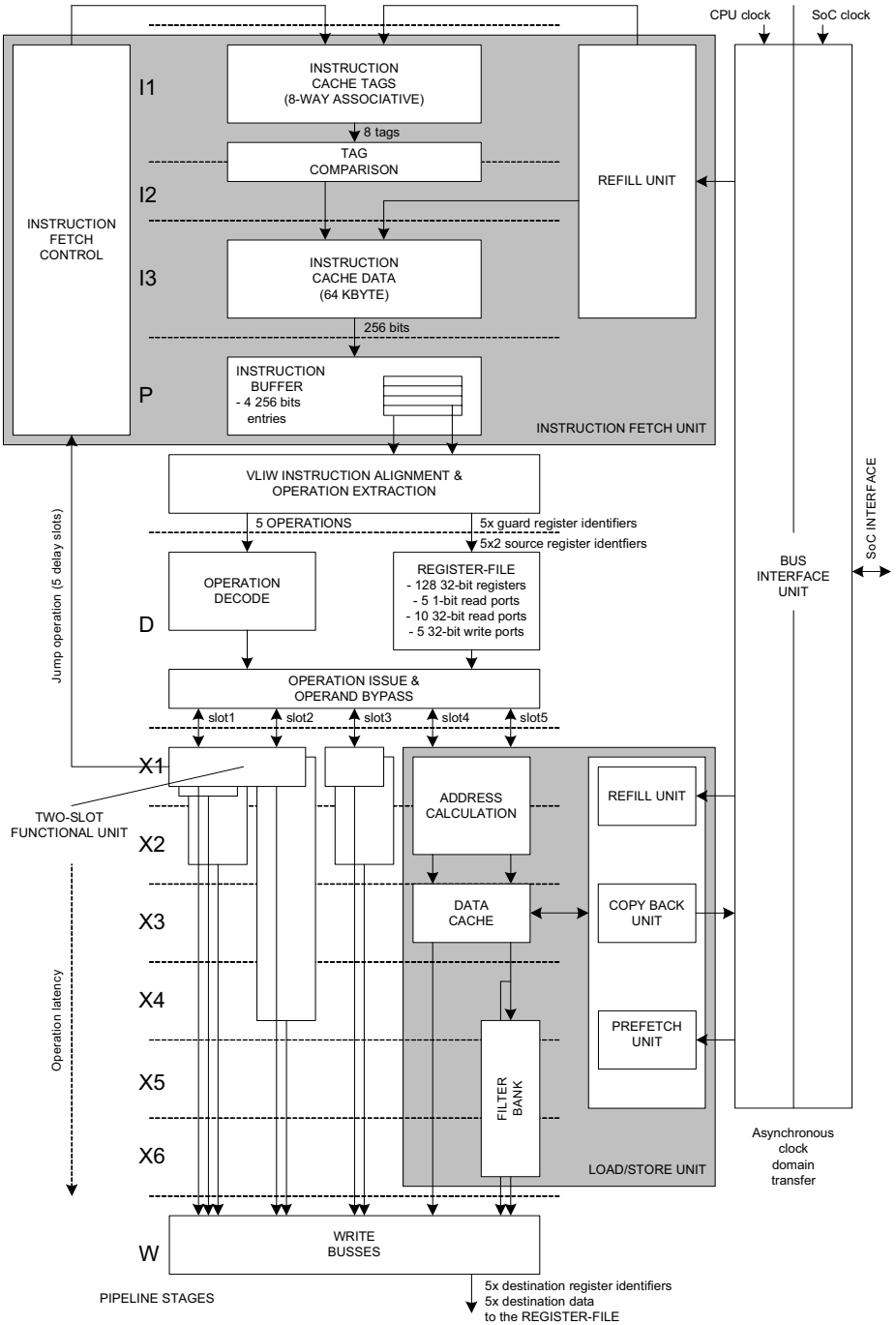

Figure 4. TM3270 media-processor pipeline partitioning.

image width times the block height of 4. While processing a certain row of blocks, the lower row of blocks is prefetched into the data cache. If the time to process a row of blocks exceeds the time to prefetch the lower row of block, the processor will not incur any stall cycles due to data cache misses.

3. TM3270 pipeline

This section gives an overview of the TM3270 pipeline (Figure 4). The pipeline has a depth of 7 stages for single cycle latency operations. Stages I1 through I3 hold the sequential instruction cache design: the access of cache tags (stage I1) and cache instruction information (stage I3) proceed in sequence to limit power consumption. Every cycle, a 32-byte aligned chunk of instruction information can be retrieved from the instruction memory. These chunks are stored in a 4-entry instruction buffer in stage P. The instruction buffer decouples the progress of the front-end of the pipeline (stages I1 through I3) from the back-end of the pipeline (stages D through W). In stage P a VLIW instruction is pre-decoded from the information in the instruction buffer. Stage D decodes the individual operations, and determines operations' operands through register-file access and operand bypassing. To support the maximum issue rate of five operations per VLIW instruction, the register-file has five 1-bit guard read ports, and ten 32-bit source read ports. Stages X1 through X6 are the execute stages. The amount of execute stages is dependent on an operation's latency. Single cycle operations have a single execute stage (X1); collapsed load operations with interpolation have six execute stages (X1 through X6). Two-slot functional units are depicted in two neighboring issue slots. Conditional and unconditional jump operations are executed in the X1 stage. Jump operations have five architectural delay slots, reflecting the pipeline distance from the first stage of instruction retrieval (stage I1) to the X1 stage. As a result, no stall cycles are observed during control flow changes, eliminating the need for branch prediction techniques. The scheduler tries to fill the 25 operations in the five jump delay instructions using aggressive predication, possibly based on program profile information. The load/store unit includes the data cache, and is located in issue slots 4 and 5. Section 4 discusses the load/store unit in greater detail. Separate cache line refill, copy back, and prefetch units connect this unit to the processor's bus interface unit (BIU). The BIU is the interface to the rest of the SoC. It includes an asynchronous clock domain transfer, which allows for flexibility when deciding upon processor and SoC operating frequency. Stage W gathers the operation results from the functional units, and allows for up to five simultaneous 32-bit updates to the register-file.

4. Load/store unit

This section describes the organization of the TM3270 load/store unit.

4.1. Cache parameters and policies

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 supports penalty-free (no stall cycles) non-aligned accesses. 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 (when compared to a fetch-on-write-miss policy), 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 a cache line. When an allocated cache line is victimized, only the validated bytes are copied back to the off-chip memory. The TM3270 and its SoC bus protocol support the transfer of cache lines with byte-validity indicators.

4.2. Load and store operations

Store operations can be issued in slots 4 or 5. Only a single load operation can be issued in slot 5. The two-slot SUPER_LD32R operation increases load bandwidth. It is issued in slots 4 and 5 (but the cache access path is restricted to slot 5, as illustrated by Figure 5), and retrieves two consecutive 32-bit words from memory into two destination registers. The collapsed LD_FRAC8 operation is issued in slot 5.

Figure 5 gives an overview of the data cache pipeline, partitioned into execute stages X1 through X6. Normal load operations have a 4-cycle latency and produce a result in stage X4. Collapsed load operations with interpolation have a 6-cycle latency and produce a result in stage X6. Stage X1 calculates the effective address for both the first and last byte referenced by an operation (Figure 5: addr_lo and addr_hi). Both addresses are required for non-aligned accesses. Stage X2 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 X3 into stage X2. Stage X3

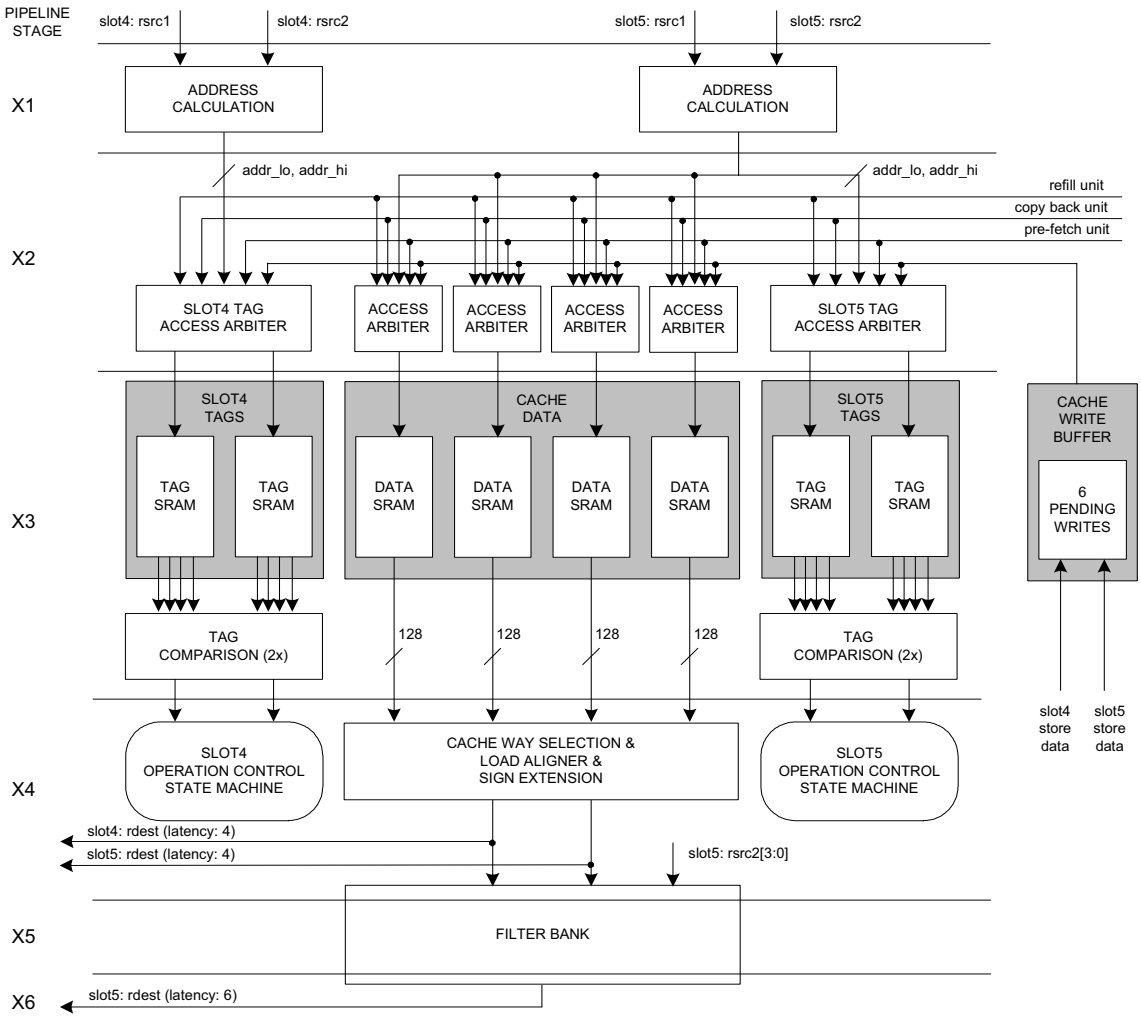

Figure 5. TM3270 load/store unit pipeline.

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 X3. Stage X4 contains the data cache way selection logic. Stages X5 and X6 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 writes 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 had a maximum data width restriction of 128 bits.

In stage X2, store operations request access to the tag memory structure, but not to the data memory structure, since stores do not produce a register 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 would require costly duplication of the data memory structure [20]. In stage X4, 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. Note that non-aligned accesses may result in two cache misses

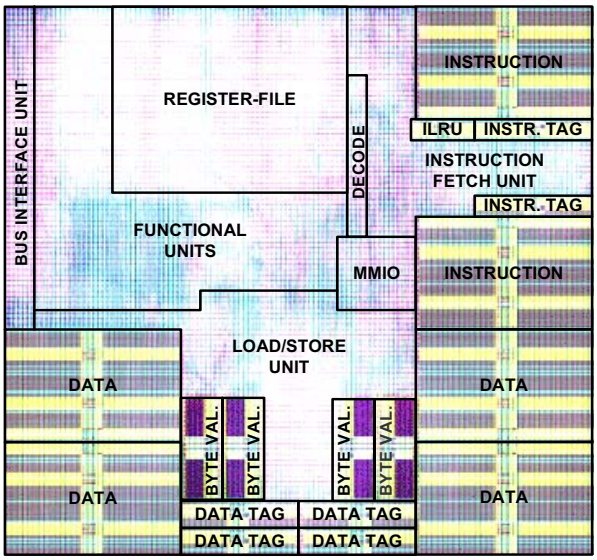

Figure 6. TM3270 floorplan.

when the data crosses a cache line boundary. For load operations, two cache lines are retrieved from memory, and for store operations, two cache lines are allocated.

5. Realization

The TM3270 is a fully synthesizable, static design. It uses off-the-shelf single ported SRAM memories. The first realization is in a low power 90 nm process technology. Under worst case operating conditions (125 C, worst-case voltage of 1.08 V, worst case process corner) the processor reaches a frequency of 350 MHz. Figure 6 gives the processor floorplan, and its partitioning into the major design modules. The SRAM memories were hand-placed. The placement of the standard cell logic was tool-driven; i.e. no labor-intensive hierarchical placement of the modules was required to come to a placed and routed design. The synthesizability and tool-driven place and route make it relatively easy to port the design to a different process technology.

5.1. Area

Table 4 gives an area breakdown of the major modules as found in the floorplan (Figure 6). The complete processor measures 8.08 mm^2. This number includes the SRAMs for the implementation of the 64 Kbyte instruction caches and 128 Kbyte data cache, which constitute roughly 50% of the overall area. The load/store unit is the largest module, when the data cache SRAMs are included. The data cache LRU is implemented in standard cell logic. The execute

Table 4. TM3270 area/power breakdown.

Module	Description	Area	MP3 decoder power (mW/MHz at 1.2 V)
IFU	Instruction fetch unit.	1.46	0.272
Decode	Decoding of operations.	0.05	0.022
Regfile	Register-file.	0.97	0.170
Execute	All functional units.	1.53	0.255
LS	Load/store unit.	3.60	0.266
BIU	Bus interface unit.	0.24	0.002
MMIO	Memory mapped IO peripherals	0.23	0.012
Total		8.08	0.935

module has the largest standard logic area. The 128-entry register-file is a relatively large module, due to the routing inefficiency of the 15 read and 5 write ports. To reduce area, a custom implementation of this module could be considered. However, such a custom implementation would increase the effort of porting the design to a different process technology.

5.2. Power consumption

One of the main TM3270 design constraints is power consumption. Low power consumption is important to allow for 1) application in battery-operated devices, 2) a low-cost package, and 3) a fan-less system solution. The initial realization uses a low power 90 nm process technology with a relatively high threshold voltage *Vt*. Although this limits the maximum operating frequency, it results in a low *static power* consumption since transistor leakage current is proportional to inverse exponential *Vt*. *Dynamic power* consumption is defined by CV^2f, C is the switched capacitance, V is the supply voltage, and f is the operating frequency. The capacitance C is determined by process technology and activity level. Activity is addressed during processor design. E.g. the *sequential* instruction cache design for a 8-way set associative cache greatly reduces the power consumption over a more traditional *parallel* cache design. Furthermore, the processor design has been heavily clock-gated; roughly 70 different functional clock domains exist. For example, all stages of all functional units are separately gated: when they are not used, they are not clocked. Typical supply voltage V for our process technology is 1.2 V, but functional operation at 0.8 V is guaranteed at a lower frequency. This allows for dynamic voltage scaling based on computational requirements. Since the processor has a fully static design and asynchronous bus interfaces to the rest of

the SoC, the operating frequency can be changed on the fly, independent of the rest of the SoC.

We used Synopsys' Power Compiler to measure gate level power consumption, including the power grid. Power consumption is derived for a MP3 decoder (384 Kbits stereo decoding at 44.1 KHz). Table 4 gives a power breakdown at an operating voltage of 1.2 V. The contribution of static power consumption is negligible. Reducing the voltage from 1.2 V to 0.8 V reduces the power consumption to 0.935 * (0.8^2 / 1.2^2) = 0.415 mW/MHz, as a result of the quadratic dependency on voltage. MP3 decoding is performed in approximately 8 MHz with a OPI around 4.5 and a CPI close to 1.0, thanks to the large caches and the high efficiency of data cache prefetching. As a result, MP3 decoding requires 8 MHz * 0.415 mW/MHz = 3.32 mW at 0.8 V. Measurements on other applications have shown that power consumption is more dependent on effective operations per VLIW instructions (OPI), and cycle per VLIW instruction (CPI), than on the specific application. Applications with similar OPI and CPI have a similar mW/MHz rating. As the amount of stall cycles increases (larger CPI), the mW/MHz number decreases as the processor performs clock gating when it stalls. Applications with a larger CPI use relatively more power in the bus interface unit (BIU).

6. Performance

For the evaluation of new processor designs, we use a suite of about 50 applications and kernels from the media-processing domain. We made a selection of the applications, focusing on video processing (Table 5), and compare TM3270 performance to that of it predecessor: the TM3260. The applications were optimized for the TM3260, and re-compiled for the TM3270 *without modifications*. The performance results do not include improvements that could be achieved by applying TM3270 specific features (non-aligned memory access, advanced data prefetching, new operations, etc.). As such, the results are a lower bound for achievable performance improvement.

Table 6 lists the main characteristics of the TM3260 and TM3270 that cause difference in performance. The most notable are the operating frequencies and data cache capacity. To evaluate the impact of these characteristics, we measured four processor configurations. Configuration A represents the TM3260. Configuration D represents the TM3270. Configuration B represents the TM3270, with TM3260 cache sizes and a TM3260 frequency of 240 MHz. Configuration C represents the TM3270, with TM3260 cache sizes and a frequency of 350 MHz. Note that to achieve the higher operating frequency of the TM3270 (350 vs. 240 MHz), the amount of jump delay slots and the load latency is increased. As a result, the TM3270 has a deeper pipeline than the TM3260, which has a negative impact on the CPI. However, this is more than compensated by the TM3270 improved data cache design and its higher operating frequency, as is illustrated by the performance numbers (Figure 7). The measurements were performed with a 32-bit off-chip DDR SDRAM memory operating at 200 MHz.

Typically, the TM3260 (configuration A) has the lowest performance. However, for the MPEG2 application, configuration A outperforms configurations B and C. This is explained as follows. MPEG2 decoding is heavily dependent on the ability of the data cache to capture the working set. Although all configurations A, B and C have the same data cache capacity, the line size is different. The TM3270 doubles the line size to 128 bytes (this decision was

Table 5. Performance evaluation kernels/applications.

Kernel/ application	Description
memset	Sets a 64 Kbyte region to a pre-defined value.
memcpy	Copies a 64 Kbyte region.
filter	Four kernels taken from the EEMBC consumer suite.
rgb2yuv	
rgb2cmyk	
rgb2yiq	
mpeg2_a	MPEG2 decoder application, run on different streams. "mpeg2_a" is characterized by a highly disruptive motion vector field.
mpeg2_b	
mpeg2_c	
filmdet	Film detection algorithm, as used in TV sets.
majority_sel	De-interlacer algorithm, as used in TV sets.

Table 6. TM3260 and TM3270 characteristics.

Feature	TM3260	TM3270
Operating frequency*	240 MHz	350 MHz
Instruction cache	64 Kbyte, 64-byte lines Parallel cache design 3 jump delay slots	64 Kbyte, 128-byte lines Sequential cache design 5 jump delay slots
Data Cache	16 Kbyte, 64-byte lines 8 way set-associative Fetch-on-write-miss 3-cycle load latency 2 loads / VLIW instr.	128 Kbyte, 128-byte lines 4 way set-associative Allocate-on-write-miss 4-cycle load latency 1 load / VLIW instr.

*under similar operating conditions: same process technology, 125 C, worst-case voltage 1.08 V, worst-case process corner.

based on a 128 Kbyte data cache), resulting in more capacity misses for MPEG2 decoding, increasing the amount of stall cycles. Furthermore, the ability to perform two loads / VLIW instruction and the 3-cycle load latency, give the TM3260 an advantage over the TM3270. As illustrated by configuration D, the larger TM3270 data cache capacity more than makes up for this TM3260 advantage.

The memcpy kernel shows the largest performance gain going from configuration A to B. The main reason is the TM3270's write miss policy. The kernel is memory bound for both configurations. Since the TM3270 generates less memory traffic, its performance is significantly higher (fetch-on-write-miss policy).

On average, the TM3270 gives a performance gain of 2.29 over the TM3260. This number is achieved through re-compilation of applications optimized for the TM3260. Not all applications benefit to the same extent from a larger data cache. Whereas the MPEG2 application shows a large performance gain, the EEMBC kernels and TV algorithms show a modest performance gain. These applications benefit most from a higher operating frequency.

TM3270 specific optimizations allow for larger performance gains. In [12] an optimized and non-optimized implementation of a motion estimation kernel on the TM3270 was evaluated. An additional performance gain of more than a factor two can be achieved, when taking advantage of non-aligned memory access, advanced data prefetching techniques, and new operations. In [13] a MPEG2 encoder application was evaluated. New operations improve the performance of a MPEG2 8x8 texture pipeline by 50%. In [14] a state-of-the-art temporal upconversion algorithm was evaluated. New operations improve performance by 40%, data prefetching improves performance by more than 20%.

7. Conclusions

We have presented an overview of the architecture, implementation, and first realization of the TM3270 media-processor. It is applied as an embedded processor in SoCs targeting the video and audio processing domains in the cost-driven connected and portable consumer markets. It provides a programmable platform on which a variety of video and audio processing applications are implemented.

With the introduction of new operations to the TriMedia ISA, and an increase in operating frequency when compared to its predecessor, it is able to either encode or decode H.264 video material at standard definition resolution. Especially the CABAC specific operations and collapsed load operations with interpolation contribute heavily to this ability. To our knowledge, the TM3270 is the first processor that supports two-slot operations. These operations open up the possibility to directly support functions that have up to four inputs and produce up to two outputs. Furthermore, they allow for the combining of multiple two input operations, possibly reducing overall function latency and register pressure.

We discussed an overview of the organization of the processor pipeline and load/store unit. The discussion illustrates the trade-offs made between architecture and

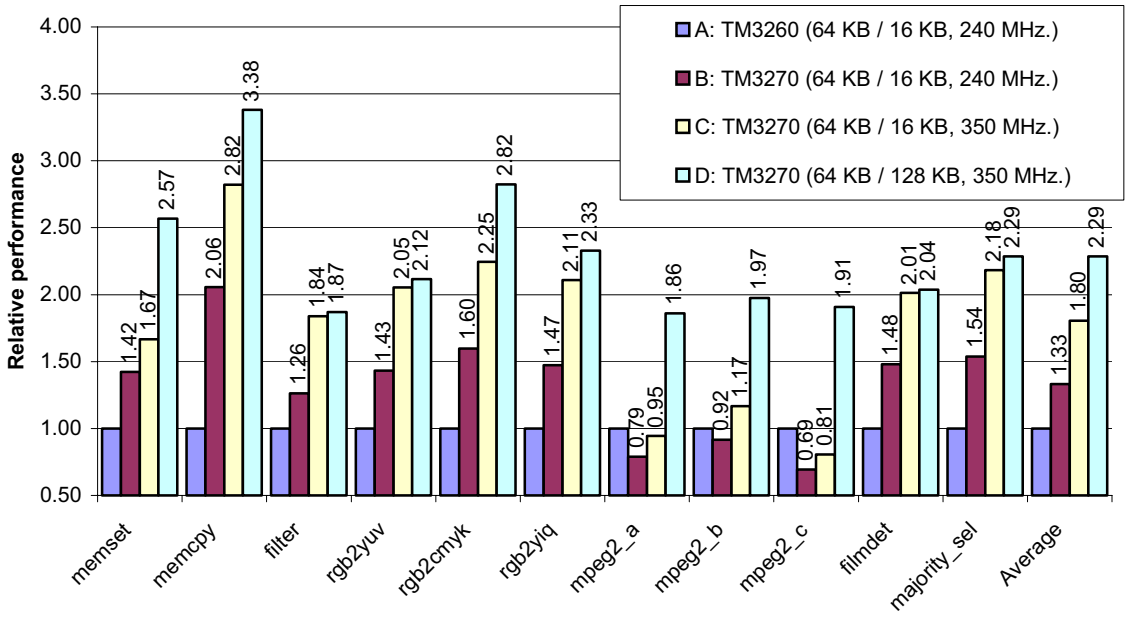

Figure 7. Relative performance numbers.

implementation of processor design, e.g. efficiency of implementation of a feature such as two stores per VLIW instruction results in a semi dual ported cache design. The processor achieves an average 2.29 performance gain over its predecessor, the TM3260, through re-compilation only. Further TM3270 specific optimizations allow for an additional performance gain of more than a factor two for certain applications.

8. Acknowledgements

We would like to acknowledge the significant contribution of Gerrit A. Slavenburg to the TriMedia architecture. Furthermore, we would like to thank those in the TriMedia application and compiler teams.

9. References

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

[2] Ravi Bhargava, Lizy John, Brian L. Evans, Ramesh Radhakrishnan, "Evaluating MMX technology using DSP and multimedia applications", Proc. of the 31st International Symposium on Microarchitecture, pp. 37-46, November 1998.

[3] K. Diefendorff, P.K. Dubey, R. Hochsprung, and H. Scales, "Altivec extensions to PowerPC accelerates media processing", IEEE Micro, pp. 85-95, March-April 2000.

[4] P. Ranganathan, S. Adve, and N.P. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions", Proc. of the 26th annual International Symposium on Computer Architecture, pp. 124-135, May 1999.

[5] S. Ciricescu, R. Essick, B. Lucas, P. May, K. Moat, J. Morris, M. Schuette, and A. Saidi, "The reconfigurable streaming vector processor (RSVP)", Proc. of the 36th International Symposium on Microarchitecture, pp. 141-150, December 2003.

[6] C. Kozyrakis, and D. Patterson, "Vector vs. superscalar and VLIW architectures for embedded multimedia benchmarks", Proc. of the 35th International Symposium on Microarchitecture, pp. 283-293, November 2002.

[7] D. Talla, and L.K. John, "A decoupled architecture for accelerating multimedia applications", Proc. of the International Conference on Parallel Architectures and Compilation Techniques, September 2001.

[8] B. Juurlink, D. Tcheressiz, S. Vassiliadis, and H.A.G. Wijshoff, "Implementation and evaluation of the complex streamed instruction set", Proc. of the International Conference on Parallel Architectures and Compilation Techniques, September 2001.

[9] S. Rathnam, and G. Slavenburg, "And architectural overview of the programmable multimedia processor, TM-1", Proc. of the 41st IEEE International Computer Conference, pp. 319-326, February 1996.

[10] G.A. Blaauw, and F.P. Brooks, "Computer architecture: concepts and evolution", Addison Wesley Longman Inc, 1997.

[11] J.T.J. van Eijndhoven, et. al., "TriMedia CPU64 architecture" , Proc. of the International Conference on Computer Design, pp. 593-599, October 1999.

[12] 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.

[13] J.W. van de Waerdt and S. Vassiliadis, "Instruction set architecture enhancements for video processing", Proc. of the 16th IEEE International Conference on Application-specific Systems, Architectures and Processors, pp. 146-153, July 2005.

[14] 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 IASTED International Conference on Signal and Image Processing, pp. 434-441, August 2005.

[15] http://www.semiconductors.philips.com/pip/PNX1502E_G.html

[16] N. Seshan, "High VelociTI processing", IEEE Signal Processing Magazine, vol. 15, issue 2, pp. 86-101, March 1998.

[17] C. Basoglu, W. Lee, and J. O'Donnell, "The Equator MAP-CA DSP: an end-to-end broadband signal processor VLIW", IEEE Trans. on Circuits and Systems for Video Technology, vol. 12, no. 8, pp. 646-659, August 2002.

[18] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard", IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 620-636, July 2003.

[19] N. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers", Proc. of the 17th International Symposium on Computer Architecture, pp. 364-373, June 1990.

[20] 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 International Symposium on Microarchitecture, pp. 46-56, December 1997.

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