# DESIGN SPACE EXPLORATION FOR FUTURE TRIMEDIA CPUs

F.Sijstermans, E.J.Pol, B.Riemens, K.Vissers
Philips Research, Prof. Holstlaan 4,
5656AA Eindhoven, The Netherlands

email: sijs@natlab.research.philips.com

S.Rathnam, G. Slavenburg
Philips Semiconductors, TriMedia Product Group,
811 E. Arques Ave.,
Sunnyvale CA94088, U.S.A.

## Abstract

*It is widely recognized that fine-grain parallelism can greatly enhance a processor's performance for signal processing applications. For this reason, future generation TriMedias will combine VLIW and subword parallelism in a single CPU. In this article, we present a snapshot of the new CPU's design process: the outlines are clear but fine tuning is still ongoing. We present the design flow and 'workbench' that the designers use for further tuning.*

## Introduction

The TriMedia TM1000 media processor [1][2] is the first product in a family of processors that Philips Semiconductors Sunnyvale plans to release. It can be used in stand-alone configurations or as a multimedia accelerator in a PC environment. Application areas are video conferencing, MPEG-related applications, 3-D graphics and digital audio. In this paper we describe the possible CPU architecture of successors of the TM-1000.

VLIW architectures have been prominent in specialized multimedia processors [3][4]. These architectures exploit instruction level parallelism without complex instruction-issue hardware. As such, VLIW offers a cost-effective alternative to superscalar processing. In the same time that the VLIW concept matured, standard microprocessor instruction sets have been extended with instructions that operate on several data elements simultaneously, thus exploiting subword parallelism [5][6]. The TM1000 is a VLIW architecture with limited provisions for subword parallelism. To enhance the performance of the TriMedia CPU core, the instruction set will be extended with more subword parallelism.

This paper is organized as follows. We start with a brief overview of the TM1000 architecture. Next, we explain how VLIW and subword parallelism can be combined. This combination leads to a class of CPUs that we are interested in. To find the optimal instance of this class, we have to compare the various options with respect to e.g. performance and area. A formal description of instances is indispensable for such a quantitative comparison. We present the format of these so-called machine descriptions. The machine descriptions are used as parameters in the design flow, so that most experiments with CPU variants only require adaptation of the machine description. The last section describes the tool chain in which this is realized.

## The TriMedia TM1000 processor

The TM1000 consists of the following components:

- a high-performance bus and memory system that provides communication between the processing units
- A powerful VLIW DSP-CPU that runs the multimedia application and operating system software.
- multimedia input and output units
- Two multimedia co-processors that implement MPEG variable length decoding (VLD) and image processing functions like spatial scaling, alpha-blending, and YUV-RGB conversion.

The most innovative part of the chip is the VLIW DSP-CPU. Each clock cycle, an instruction is fetched from the instruction cache. Such an instruction consists of up to five

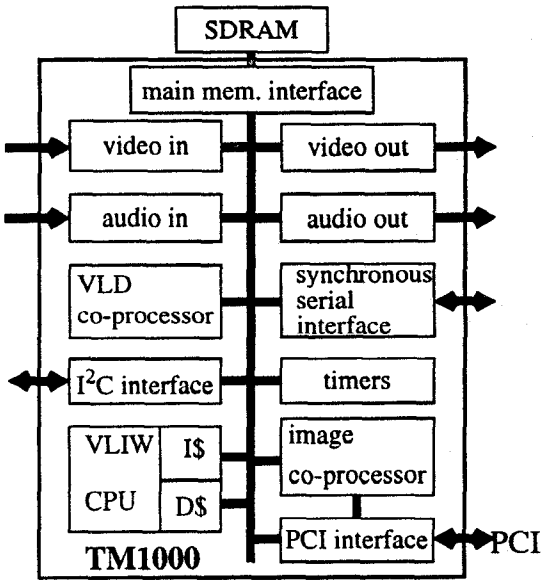


Fig. 1: TM1000 block diagram



0-7803-4428-6/98 $10.00 © 1998 IEEE

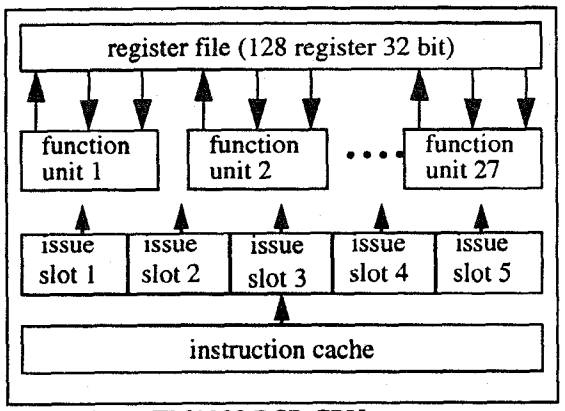

Fig. 2: TM1000 DSP-CPU

independent operations. Each operation is issued for execution on one of the 27 functional units. Examples of functional units are ALUs, memory access units, and branch units. It is the compiler's task to analyse the availability of functional units, registers, and register file ports. Therefore, all five operations packed in one instruction can be issued directly without further run-time control. The only reason for stalling the pipeline is a cache miss; if these occur the complete pipeline will be stalled.

The operations in the TM1000 are RISC like in the sense that they operate on up to two register arguments and put their result back into a register (the obvious exception of course being explicit load and store operations). There is a single register file consisting of 128 registers of 32-bits. The register file has ten read ports and five write ports. Combined with a powerful forwarding network, this enables the parallel execution of five two-argument and one-result operations per cycle.

## VLIW and subword parallelism

The TriMedia concept applies a highly modular design: many parts can be changed independently of each other. E.g. at the system level, I/O units or co-processors can be added or deleted without consequences for other units. For the DSP-CPU, the number of registers, the set of functional units, and the number of operations per instruction are typical parameters. In this section, we discuss possible instruction set extensions that allow for further exploitation of subword parallelism.

To clarify the discussion on the concepts of VLIW and subword parallelism, we consider a trivial example: the addition of two byte arrays. In C, the loop that implements this function looks like:

```
char a[n], b[n], c[n];
int i;
for (i=0; i<n; i++) c[i] = a[i]+b[i];
```

Below, we show the pseudo-assembly code of the loop (without initialization) for a 'normal' RISC-like processor. Most operations are obvious. Operations 7 and 8 are jump operations. We are assuming that there are three branch delay slots, i.e. if the jump in operation 8 is taken, operation 9, 10 and 11 are still executed before control jumps back to operation 1. Operation 7 and 8 are a jump on false and on true, respectively, where the first argument is the condition and the second argument is the operation number to jump to.

```
1  x = *a          7  jmpf g 12
2  y = *b          8  jmpt g 1
3  i += 1          9  a += 1
4  g = i<n         10 b += 1
5  z = x+y         11 c += 1
6  *c = z          12 ....
```

In this implementation, each iteration takes eleven operations and hence also eleven clock cycles.

The TM1000 compiler tries to pack as many operations as possible in each instruction with a maximum of five. It takes data dependencies and latencies of the operations into account. Jumps and loads have a latency of three cycles, all other operations in this example have a latency of one. Another restriction is imposed by the number of functional units of a kind, but this does not play a role for our example. If no useful operation can be scheduled, the compiler inserts a 'nop' operation. The pseudo-assembly code for the TM1000 code could be:

```
1  x=*a        y=*b        i+=1  a+=1  b+=1
2  g=i<n       nop         nop   nop   nop
3  jmpf g 7    jmpt g 1    nop   nop   nop
4  z=x+y       nop         nop   nop   nop
5  *c=z        c+=1        nop   nop   nop
6  nop         nop         nop   nop   nop
```

This code contains the same eleven operations as the sequential code, but these operations are shuffled by the compiler to fit in six instructions, a reduction of almost a factor two. This example still exploits limited VLIW parallelism. How to exploit, with programming in C, the available parallelism in this architecture is subject of [7] and [8]

For the next generation of CPUs, we are considering to extend the use of subword parallelism: a number of separate elements are packed together in a single register and part of the instructions operate on these 'vectors of elements' rather than treating the registers as scalar data. Since the benefits grow with the vector length, we introduce wider registers in the CPU, typically 64 or 128 bit wide. Unfortunately, current compilers cannot detect the opportunities for the use of subword parallelism. Therefore, the application programmer will have to use 'vector libraries' that are mapped to subword parallel instructions. The most convenient programming style is offered by C++ in which we can define vector classes and overload operators. In our example, we use the class vec64sb of vectors consisting of 8 signed bytes. The addition of elements of this class is defined to be a vector addition. The 'vectorized' version of our example

becomes:

```
vec64sb a[n/8], b[n/8], c[n/8];
int i;
for (i=0; i<n/8; i++) c[i] = a[i]+b[i];
```

Notice that a[i], b[i], and c[i] are vectors of eight bytes now and the + denotes a vector addition. Thus, every iteration now handles eight bytes.

The compiler will map this C++ fragment to the assembly code below. We use capitals for the variables in 64-bit registers. At assembly level, the type information has been translated into special vector operations.

```
1  X=ld64b a      Y=ld64b b  i+=1  a+=1  b+=1
2  g=i<n          nop        nop   nop   nop
3  jmpf g 7       jmpt g 1   nop   nop   nop
4  Z=add64sb X Y  nop        nop   nop   nop
5  st64b c Z      c+=1       nop   nop   nop
6  nop            nop        nop   nop   nop
```

The structure of this code is the same as that of the non-vectorized code. Instead of one byte, the six instructions now handle eight bytes, thus achieving a speed up factor of eight.

This small example demonstrates that VLIW instruction level parallelism and subword data level parallelism can well be used in combination. Furthermore, although the example is oversimplified, people involved in signal processing may recognize some of the 'typical' signal processing characteristics: many samples of a finite resolution are all treated the same; control processing is relatively simple.

## Machine Description

In this section, we explain the formalism that we use to describe CPUs that combine VLIW and subword parallelism. We call this our 'machine description' format. A machine description file describes all parameterizable features of the processor. We present a simplified version of the machine description part that we use for the CPU. The machine description plays a central role in the CPU's design process, which is described in the next section.

One section in the machine description describes the register files.Scalar and vector registers need not have the same length. Therefore, more than one register file may be defined. In the example below, we define two register files: r for scalar data (the TM-1000 register file) and v for vector data. The definition of a register file consists of the number and size of its registers and the number of read and write ports.

```
REGISTERS
    r       SIZE 32 NUMBER 128;
    v       SIZE 64 NUMBER 32
READ BUSES
    REGISTERS r    NUMBER 10;
    REGISTERS v    NUMBER 7;
WRITE BUSES
    REGISTERS r    NUMBER 5;
    REGISTERS v    NUMBER 3;
```

A second part describes the arguments and results of operations, together called the signature. The semantics of an operation are not a part of the machine description; they are defined in a separate C-file. Below we show an example of part of the operations section. The first signature defines the vector byte load operations, which take a scalar register with an address as input and put their result in a vector register. The second signature defines signed and unsigned integer additions and subtractions with an immediate argument of which the range is defined.

```
OPERATIONS
   SIGNATURE (r->v)
      ld64sb, ld64ub;
   SIGNATURE (r, PAR(-256 TO 255) -> r)
      iaddi, isubi, uaddi, usubi;
```

There is no one-to-one mapping between C-level software operations and operations in hardware. Several software operations may be mapped to the same hardware operation; other software operations are mapped to multiple hardware operations. Such mappings are defined in the so-called pseudo-operation list. In the example below, we define that the 'less' operation is implemented by a 'greater' operation with swapped arguments and that the 'add-immediate' operation is implemented by an 'immediate' operation followed by an 'add' operation. In this list, arguments and results are prefixed by a $-sign. The result is mentioned before the operation name, arguments after it.

```
PSEUDO OPERATIONS
   $3 less  $1  $2  = $3 gtr $2 $1;
   $3 iaddi $1 ($p) = $4 iimm ($p),
                      $3 iadd $1 $4;
```

Yet another part of a machine description file describes the functional units, i.e. operations and latency, and their number and slots in the CPU. We have seen that the TM1000 has five slots in which operations can be issued. An operation can only be scheduled in a slot if a functional unit that executes this operation is connected to this slot. Below, we show an example with five alus and two load-store units. Load-store operations can only be issued in the first two issue slots.

```
ISSUESLOTS 5
FUNCTIONAL UNITS
   alu
      SLOT  1 2 3 4 5
      LATENCY 1
      OPERATIONS    iadd,isub,uadd,usub
   memory
      SLOT  1 2
      LATENCY 3
      OPERATIONS    ld64b,st64b
```

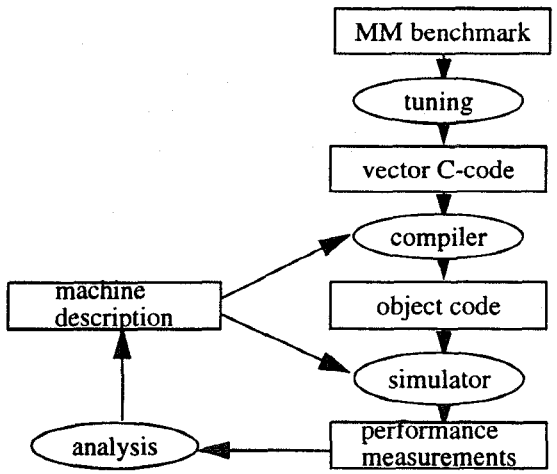


Fig. 3: CPU design flow

## Design Flow

The machine description files describe a class of CPUs from which we have to choose the optimal instance. The main optimization criteria are performance and chip area. Here, we focus on performance measurements. In particular, we explain how our advanced tool set reduces the time to do design space exploration. Chip area can be estimated quite accurately by extrapolation of TM1000 results.

Performance measurements are done for a set of relevant multimedia benchmark functions with high processing demands, in particular in video (de)compression, video quality improvement, and 3D graphics. These benchmarks are manually 'massaged' into tuned C-programs, taking into account the target architecture, especially possibilities for VLIW parallelism and the vector instruction set. The compiler maps the resulting C-code to object code for the target architecture, which is then executed by a simulator. The simulator generates the relevant performance figures like number of instruction cycles and achieved parallelism. Analysis of the performance measurements may lead to new ideas on the architecture. To assess the value of these new architectures, the performance measurements have to be repeated. Thus, the design loop is closed.

The central idea in the design flow is to minimize the effort for testing a new idea or tuning a design-parameter. To achieve this, we have developed a retargetable compiler and simulator that are parameterized with a machine description file. Performance measurement of another instance of the CPU, thus, boils down to changing the machine description and running the compiler and simulator again. The manual tuning from benchmark to vector C-code usually needs to be done just once. One reason is that our optimizing compiler extracts the VLIW parallelism itself. A second reason is that the library of C-level vector operations is kept constant even if the vector hardware changes. The pseudo-operation section of the machine description file is used by the compiler to map software operations to hardware operations. Only for largely different architectures, the manual tuning may have to be redone.

## Concluding remarks

We have shown how the performance of the DSP-CPU of the TriMedia can be boosted by a combination of VLIW and subword parallelism. This mixture appears to be very suitable for signal processing applications. A unique feature of our solution is the powerful compiler support that allows for fast application development. We have made the compiler and simulator retargetable to different CPU instances, in order to exploit this strength for processor design. The design space is so huge that we can cover a sufficiently large part only if we let tools do the bulk of the work.

Similar design trajectories exist or are in development for caches, buses, and co-processors. We use a modular simulator framework so that design optimization can be done independently, but a complete system can be simulated by linking the modules together.

## Acknowledgements

Our thanks go to all members of the TriMedia and Prommpt teams and to all other colleagues that contributed to the reported work.

## References

[1] Gerrit A. Slavenburg, Selliah Rathnam, Henk Dijkstra, "The TriMedia TM-1 PCI VLIW Media Processor", *Proc. Hot Chips 8,* August 1996

[2]S. Rathnam and G. Slavenburg, "An architectural overview of the programmable multimedia processor TM1", *Proc. Compcon,* IEEE CS press, 1996

[3] S. Purcell, "Mpact 2 media processor, balanced 2X performance", *Proc. Multimedia Hardware Architectures 1997,* Proc. SPIE 3021, 1997

[4] "Digital signal processing solutions products", http://www.ti.com/sc/docs/dsps/products/c6x/index.htm

[5] Alex Peleg, Uri Weiser, "MMX technology extension to the Intel architecture", IEEE Micro, 16(4), August 1996

[6] Ruby B. Lee, "subword parallelism with MAX-2", IEEE Micro, 16(4), August 1996

[7]M. Beemster, A. van Inge, F. Sijstermans, "Polyphase filtering on the TriMedia core", *Proc. Multimedia Hardware Architectures 1997,* Proc. SPIE 3021, 1997

[8]A.K. Riemens, R.J. Schutten, K.A. Vissers, "High speed video de-interlacing with a programmable TriMedia VLIW core", Proc. ICASSP , 1997, San Diego, CA.