Vrije Universiteit Brussel

Literature study: the TriMedia, C6000, SODA and EVP processor architectures

Advanced Computer Architecture

Federico Dominguez
Ward Van Heddeghem

December 19, 2008

1 Introduction

This report describes and compares the micro-architectural aspects and ISA rationale of 4 processors:

- the **TriMedia** mediaprocessor (NXP, formerly Philips Semiconductor),
- the **C6000** platform (Texas Instruments),
- the **SODA** architecture (University of Michigan)
- and the **EVP** (NXP).

These 4 processors can all be broadly categorized as application specific digital signal processors (DSPs). The first two processors (Trimedia and TI's C6000) are targeted as **media processors**, the latter two (SODA and EVP) are processors developed almost exclusively for **Software Defined Radio** (SDR).

Media processors are typically found on DVD players, digital TV, IP TV, video security systems, video editing systems, etc. SDR processors try to replace the RF processing hardware by software, and are aimed mainly at consumer handheld devices such as PDAs, Smart Phones, Mobile Phones etc.

2 General comparison of media and SDR processors

2.1 Multimedia Processing

Multimedia processing is the handling of video and audio data in electronic devices. This is sometimes accomplished by specific purpose integrated circuits, but the lack of flexibility and high cost of this approach has led to the development of multimedia processors, programmable processors specially designed to efficiently execute all tasks related to multimedia processing. Multimedia processors are typically found on DVD players, digital TV, IP TV, video security systems, video editing systems, etc. [5]

The most common tasks for a multimedia processor are[10]:

- Image, video and audio decoding and encoding
- Image, video and audio compression
- Image, video and audio transmission
- Image, video and audio enhancement, filtering
- Pattern recognition
- Motion detection

Since multimedia processors are most often found on battery operated embedded devices and since the nature of multimedia processing requires real time deadlines, the design choices in this family of microprocessors aim to achieve high performance processing of multimedia data streams with minimum power consumption and unit cost.[5]

Multimedia processors achieve these goals by optimizing the execution of the multimedia tasks mentioned above. This implies:

- *High level of parallelism:* VLIW, SIMD to boost performance and to allow for a low cost silicon implementation.
- *Large cache memories:* To accommodate typically large images and video frames close to the processor saving costly memory access and bandwidth.

- *Penalty free un-aligned memory access:* Processing algorithms typically access image blocks in an un-aligned manner.
- *Data pre-fetching:* Multimedia processing algorithms access memory locations in predictable strides and blocks.
- *Large register files:* Large data working sets can be kept in registers, preventing costly load and store operations.
- *DMA-style memory transfers:* Increases overall system performance.
- *Application specific instructions:* Typical multimedia operations can be done in one or a few clock cycles, saving energy and gaining performance.
- *Small data words:* 8 and 16-bit data words are typically sufficient for most multimedia processing task. Limiting the size of data words allows for higher memory density and parallelism.

TriMedia and TI C6000 are two competing families of multimedia microprocessors that in one way or another implement the architectural design choices listed above.

2.2 Software-defined radio (SDR)

The physical layer of most wireless protocols is traditionally implemented in custom hardware to satisfy the heavy computational requirements while keeping power consumption to a minimum. These implementations are time consuming to design and difficult to verify. A programmable hardware platform capable of supporting software implementations of the physical layer, or **software-defined radio** (SDR), has a number of advantages: support for multiple protocols (i.e. multimode operation), faster time-to-market, higher chip volumes, and support for late implementation changes. SDR can be considered as a high-end digital signal processing (DSP) application. [13]

The digital baseband processing for SDR can be divided into three stages: filter stage, modem stage, and codec stage.

- The *filter stage* requires a high computational load and its implementation is uniform between different standards and algorithms, a programmable solution would not be the most efficient one.
- The *modem stage* is diverse among different standards, in this stage there is plenty of space for different vendors to differentiate their products by applying different algorithms and standards. This stage is ideal for a fully programmable solution.
- The *codec stage* is implemented by similar algorithms across several standards making a programmable solution not desirable. [17]

The modem stage of digital baseband processing is an ideal application for a programmable SDR processor. **SODA** and **EVP** are two such processors.

The main design goals of these SDR processors are typically *high performance* (imposed by the high throughput requirements of current wireless protocols), *energy efficiency* (battery operated devices) and *programmability* (multi-protocol support, higher chip volumes).

2.3 Common characteristics

As is clear from the discussion above, both media and SDR processors are characterized by a number of common properties: they should achieve **high performance** processing of data streams, must meet **real-time deadlines**, should be energy efficient since they are typically used on mobile (battery operated) devices, and they should be highly **programmable** as to provide support for different and new media and radio protocols. The data streams they operate on are relatively small (8 or 16 bit).

Figure 1 shows the throughput and power requirements of a number of wireless protocols, as well as the performance of two processors discussed in this paper: the TI C6000 and the SODA processor. This figure gives a good notion on the power efficiency these processors target to achieve.

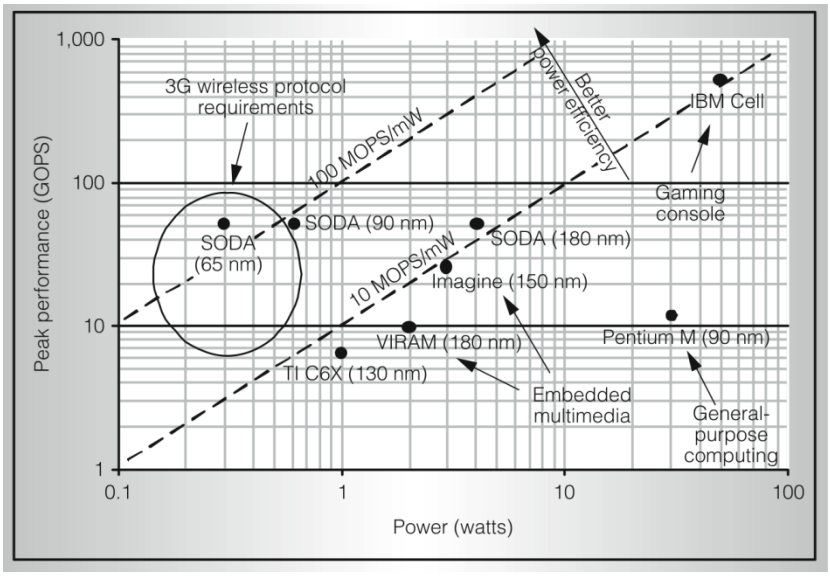

Figure 1 – Throughput and power requirements of typical 3G wireless protocols [13]

2.4 Differences between media and SDR processors

As much as they have in common, these two groups of processors also are distinctly characterized by their intended application: media and SDR. In the next paragraphs we will highlight some of the major design distinctions between these two processor groups.

Differences between the two media processors, and differences between the two SDR processors will be discussed in § 3.3 and § 4.3 respectively. Table 1 gives the specification of each of the four processors.

SIMD width — Since media processors typically operate on small macroblocks (a macroblock is a used in video compressing and represents a block of 16 by 16 pixels[1]), smaller vectors suffice. This translates in narrow SIMD designs for media processors.
For SDR, on the other hand, most of the computationally intensive algorithms have abundant data level parallelism, much more than instruction level parallelism, so wide SIMD is very efficient.[13]
It should also be noted that SIMD in general is power efficient compared to other architectures, since the 'overhead' of address calculation, address decoding, instruction fetching etc. is shared per p operations. [16]

Predication — Predication is available on both media processors. This seems logical since in video compressing, a common thing to do is compare data with previous data (e.g. in motion estimation). Predication could then be a power efficient way to make decisions, as opposed to costly branching. We have not found any hard references to back up this assumption. However, one document explicitly mentions predication as technique for increasing performance in media processors.[2]
For SDR processors we didn't find any indications that predication is used.

Data width — Both media processors have a data width of 32 bits, in contrast with the SDR processors which have only half of that.
This provides media processors with a much larger addressable memory range, not in vain given new video standards like HDTV, which currently reaches up to resolutions of 1920x1080 pixels (i.e. a total of 2.073.600 pixels, far beyond what is addressable with 16 bits). [1]
Algorithms running on SDR processors typically operate on variables with small values. Analysis of two typical wireless protocols shows that there should be strong support for 8 and 16-bit operations.[13] Also, wireless packets handled by these processors don't potentially consume as much memory space as the media processors.

Unaligned loads — Media processing algorithms typically access image blocks in an un-aligned manner. As already mentioned, when dealing with video compression, media processors handle macroblocks. The location of these macroblocks is not fixed, neither on screen, nor in memory. It seems logical that providing for unaligned loads allows these blocks to be fetched faster. We

don't have any references to back up this assumption.
For SDR processors we didn't find any indications that unaligned loads are supported.

Cache — On media processors, caches are implemented (or optional), to accommodate for typically large images and video frames close to the processor, saving costly memory access and bandwidth.
SDR processors don't implement caches since they operate on streaming data (caching is of use then), and they require real-time behavior.[13]

Delay slots — Delay slots are available on media processors to improve performance. On SDR processors, delay slots are not used since branch prediction isn't use either.

Table 1 – Processor specifications

Feature	Media processors		SDR processors	
	TriMedia	**C6000**	**SODA**	**EVP**
Architecture	Uni-processor 5 issue slot VLIW guarded RISC-like operations	Uni-processor	1 ARM processor + 4 PEs (processing elements) static scheduled	Uni-processor SIMD through vector processing capabilities. 13 issue slot VLIW scalar and vector operations
VLIW	Yes	Yes	Yes	Yes
Pipeline depth	7 – 12 stages	11 stages	5 stages	--
Data width	32 bits	32 bits	16 bits	16 bits
Register file	Unified, 128 32-bit registers	Clustered, 16 32-bit registers	Per PE: 32x16 SIMD 16-bit registers, 16 scalar 16-bit registers	16 vector registers (each register contains 16 words). 32 scalar registers.
Functional units	31	8	Per PE: 32+1	6 vector, 3 scalar
SIMD capabilities	1 x 32-bit, 2 x 16-bit, 4 x 8-bit	C62/67: no C64: 4 x 8-bit, 2 x 16-bit	Per PE: 32 x 16-bit	16 x 16-bits
Memory structure	Cache	Optional cache	Clustered, Scratchpad Global: 64 KB Per PE: 4 KB scalar, 8 KB SIMD	Scratchpad
• **Instruction cache**	64 Kbyte, 128-byte lines, 8 way set-associative, LRU replacement policy	(model dependent)	No	No
• **Data cache**	128 Kbyte, 128 byte lines 4 way set-associative, LRU replacement policy, Allocate-on-write miss policy	(model dependent)	No	No
Frequency	240 MHz (300 MHz)	300 MHz (1 GHz)	380 MHz	300 MHz
Delay slots	5 for branch instructions	1 for multiply instruction, 4 for a load instruction and 5 for a branch instruction	No	No
Compression	Yes (10 bit template)	Yes (using stop bits)	*NDA*	*NDA*
Predication	Yes	Yes	*NDA*	*NDA*
Forwarding	Yes	Yes	No	*NDA*
Unaligned loads	Yes	Yes	*DNA*	*DNA*
Compiling / programming	Hard. C, C++ with special extensions.	Hard. C, C++, assembly.	Hard. SPIR on top of high level language	Hard. EVP-C
***NDA*: no data available**				

3 Multimedia processors: TriMedia and TIC6000

3.1 TriMedia

***Remark**: unless otherwise specified, the information in this section is taken from the main TriMedia paper by Van de Waerdt, et al.* [3]

3.1.1 Introduction

TriMedia is a family of microprocessors developed by NXP. The main application of the TriMedia microprocessors is multimedia data processing in embedded systems. This particular application domain shapes the TriMedia microprocessor design, diverging drastically from general purpose processors.

TriMedia processors are deployed on consumer devices as a System On Chip solution. TriMedia is programmable, so that it can be adapted to many different applications and so that products can be changed through software to meet evolving standards requirements[5].

TriMedia processors employ VLIW architecture. The latest model's CPU core (TM3270) has 31 functional units with five issue slots. This family of processors also supports SIMD operations. VLIW and SIMD provide a high degree of instruction parallelism, optimizing the overall performance of the system.

3.1.2 Architecture overview

TriMedia instruction and data level parallelism

TriMedia implements instruction and data level parallelism through a 5-issue slot **VLIW** and through **SIMD**, respectively. VLIW supports up to 5 operations per instruction word. SIMD instructions can work on two 16-bit data words or four 8-bit data words.

Figure 2 shows the **functional units** of the TM1000, the first member of the TriMedia family. The TM1000 has 28 functional units divided among five issue slots. The TM3270 has added three new functional units (totaling 31 units) but maintains the same basic design structure as the TM1000. As can be seen in the figure, each issue slot has multiple types of functional units, allowing the compiler more freedom when scheduling the VLIW instructions.

The TM3270 implements **super-operations** or two-slot operations. These operations are executed by two functional units that occupy two neighboring issue slots. These operations can take up to four operands and up two destination operands. Two-slot operations can improve overall system performance since the equivalent separate operations require more cycles to complete.

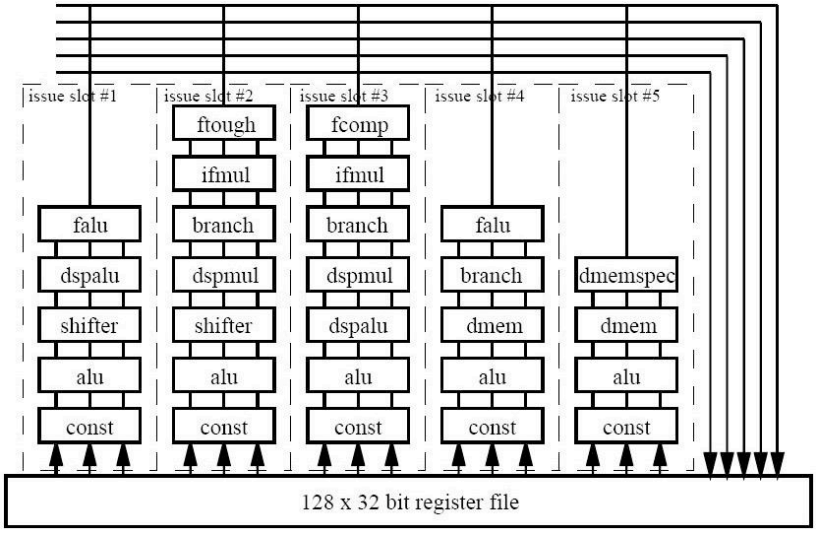

Figure 2 - TriMedia TM1000 Functional Units [5]

Cache policies and memory prefetching

The TM3270 has a 128 Kbyte data **cache** with 128 cache lines. The cache supports penalty-free unaligned accesses; image processing typically requires fetching blocks of unaligned image data from memory. The cache has a write-back policy and an allocate-on-write-miss policy. The combination of these two policies reduces the required bandwidth to off-chip memory.

The TM3270 supports **prefetching** and is based on memory regions. The prefetching pattern and region can be specified by the programmer and is designed to match the memory access pattern of typical multimedia codec algorithms (typically unaligned and non-sequential). The TM3270 supports four separate memory regions. The pattern is defined by a start and end address and a stride. The region specified by the start and end address is usually a complete image or frame, and the stride corresponds to the size of the blocks within the image that are being processed. The main goal of prefetching is to reduce cache misses and therefore avoid costly stall cycles and improve system performance.

Pipeline

Figure 3 shows a basic block diagram of the TM3270 pipeline. The pipeline has a minimum depth of 7 stages for single cycle latency operations and a maximum of 12 stages for more complex instructions. The front end of the pipeline consists of instruction fetch (I1 – I3) and instruction pre-decode (P). The instruction fetch control unit at the front end of the pipeline supports 5 **delay slots** for the jump operation, the number of delay slots corresponds to the distance between the first stage and the functional units. The back end of the pipeline consists of operation decode (D), execution (X1-X6) and write (W).

Data forwarding from the Cache Write Buffer is currently not implemented in the TriMedia pipeline because its inclusion causes path delays beyond the desired design parameters. **Predication** is supported by using guard registers; each operation execution is predicated with the least significant bit of its guard register. [6]

3.1.3 Compiler support

The TriMedia processor family comes with compiler support from the manufacturer. Binary compatibility is not guaranteed between different models within the family requiring a different compiler for each model. The supported programming language is C/C++.

Programs written in C/C++ are compatible between models provided they are recompiled for each model. Some performance is lost when porting programs written for one model to a newer model without including model specific optimizations. For example when porting a program from the TM3260 to the TM3270, a performance gain of 20% can be obtained when applying new data prefetching techniques.

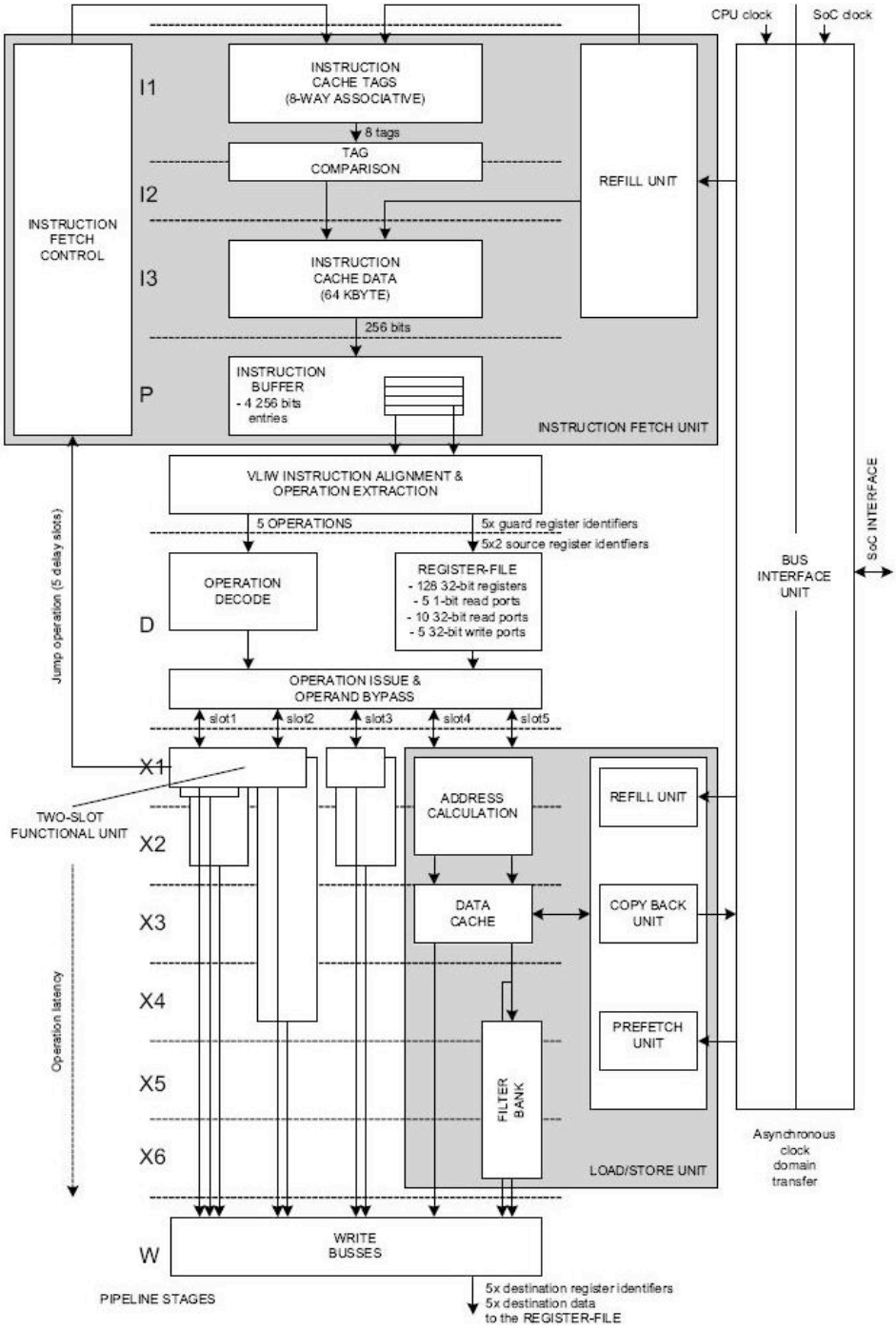

Figure 3 - TriMedia TM3270 Pipeline [3]

3.2 TI C6000

***Remark**: unless otherwise specified, the information in this section is taken from a number of Texas Instruments reference documents.*[7][8][9][10]

3.2.1 Introduction

The TMS320C6000 digital signal processor (DSP) platform is part of the TMS320 DSP *family* developed by Texas Instruments.

The **TMS320 family** consists of 16-bit and 32-bit fixed- and floating-point DSPs. There are three main *platforms*, including the TMS320C2000 (control applications), the TMS320C5000 (power-efficient applications), and the **TMS320C6000** (high-performances applications). These processors are used in cell phones, digital cameras, modems etc.

The C6000 platform is a **very long instruction word (VLIW) architecture** targeted at high-performance DSP tasks. It comprises the following three main *generations* (i.e. versions):

- TMS320C62x ('C62x) offering **fixed-point** arithmetic up to 300 MHz,
- TMS320C64x ('C64x) offering **fixed-point** arithmetic up to 1 GHz,
- TMS32067x ('C67x) offering **floating-point** arithmetic up to 300 MHz.

3.2.2 Architecture overview

Overview

The 'C6000 processor consists of three main parts: a **CPU**, **peripherals** (not discussed further) and **memory**. The processors operate at various frequencies, ranging from 150 MHz up to 1 GHz.

The processor executes up to eight 32-bit instructions every cycle. The core CPU, as shown in Figure 4, consists of eight functional units, two register files and two data paths.

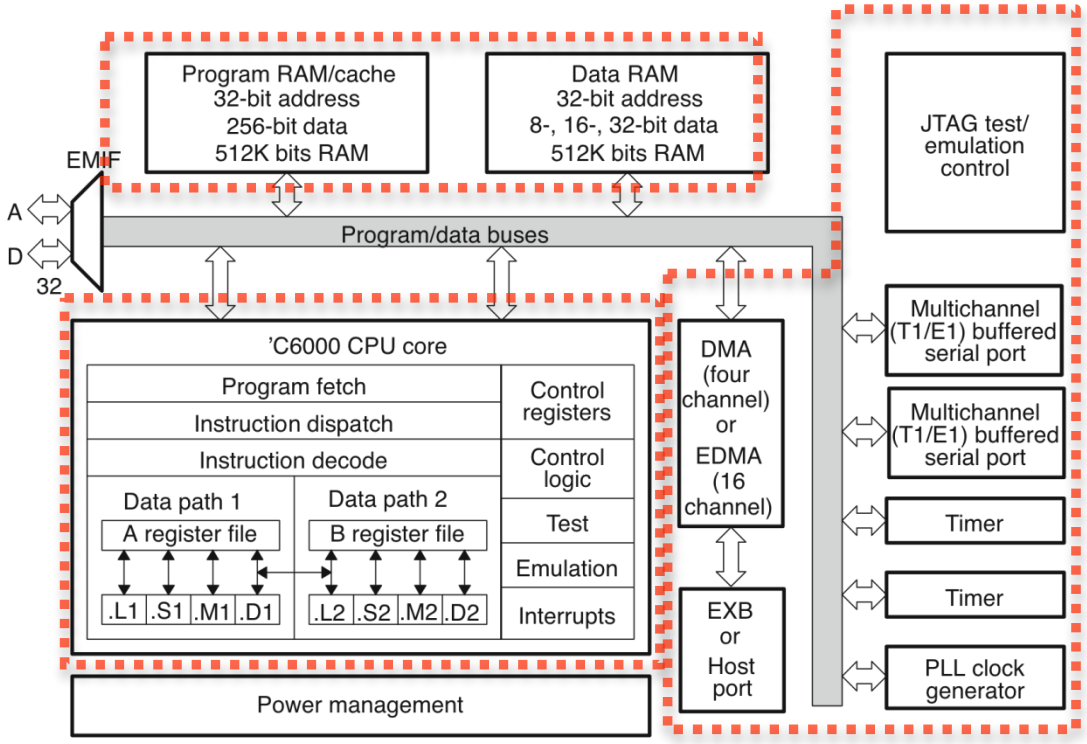

Figure 4 – The 'C6000 block diagram

CPU

The CPU has two **clusters** (labeled **data path** 1 and 2 in Figure 4) in which processing occurs. Each data path has four functional units (.L, .S, .M, and .D) and a **register file** containing 16 32-bit registers (32 for the 'C64x generation).

The **functional units** execute logic, shifting, multiply, and data address operations. All instructions except loads and stores operate on the registers. The two data-addressing units (.D1 and .D2) are exclusively responsible for all data transfers between the register files and memory.

The **cross paths** bus connecting both data paths, supports one read and write operation per cycle.

VLIW processing flow begins by fetching a 256-bit wide packet (eight 32-bit words) from program memory, which is then dynamically dispatched over the different functional units (FUs). Dynamic dispatching refers to the fact that the order of instructions in a VLIW should not match the order of the functional units; instead instructions in the VLIW are dispatched to the appropriate FU.

The **pipeline** stages are grouped into 3 phases: instruction fetch (4 stages), instruction decode (2 stages) and instruction execute (maximum 5 stages). The pipeline also supports delay slots for multiply (1 delay slot), load (4 delay slots), and branch (5 delay slots) operations.

Memory

Data and program memory — The C6000 DSP has a 32-bit, byte-addressable address space. Internal (on-chip) memory is organized in separate data and program spaces, which may be configured as cache on some devices. When off-chip memory is used, these spaces are unified on most devices to a single memory space via the external memory interface (EMIF).

Memory ports — The C6000 DSP has two 32-bit internal ports to access internal data memory. The C6000 DSP has a single internal port to access internal program memory, with an instruction-fetch width of 256 bits.

Memory alignment — The C6400, unlike the C6200 and C6700, supports unaligned memory access for load and store operations. Word and double-word data does not always need alignment to 32-bit or 64-bit boundaries as in the other models.

'C64x extensions

The key extensions between the 'C62x/'C67x architecture and the 'C64x architecture that allow more work each clock cycle include (see also Figure 5):

- wider data paths (dual 64-bit load/store paths instead of 32-bit),
- a larger register file (32 registers instead of 16),
- more orthogonality, i.e. more generality in the architecture (for example, the .D FU can perform 32-bit logical instructions in addition to the .S and .L units).
- unaligned memory access for load and store operations.
- More SIMD support. For example, the functional units on the C64x can now execute two 32-bit multiplies and four 16-bit multiplies. On Figure 5, the yellow boxes in each functional unit on the C64x represent the increase in SIMD support. This increase in SIMD parallelism is possible due to the increased register file size and data path width.

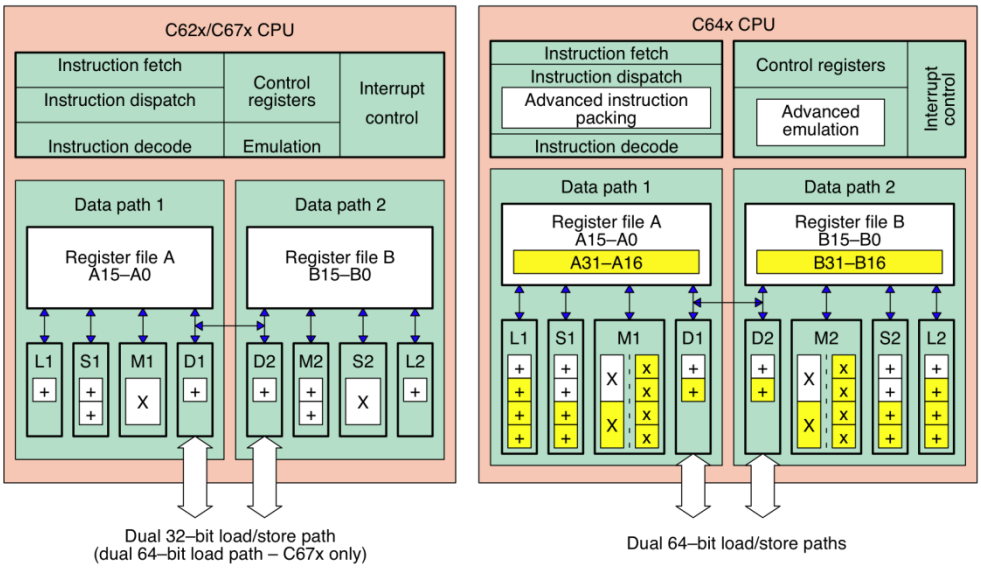

Figure 5 – Differences 'C62/67x and 'C64x generation

3.2.3 Compiler support

The C6000 platform can be programmed using C, C++ or assembly language.

Texas Instruments provides a proprietary toolchain and IDE, called *Code Composer Studio*, to do so. No other compilers are available for the C6000 platform; however, a department at the Chemnitz University of Technology developed preliminary support for the TMS320C6x series in the GNU Compiler Collection (GCC).[12]

The fact that only one proper toolchain is available for the C6000 platform, and that TI provides hand-optimized libraries for various algorithms (for example, FFT), seems to indicate that it can be a complex task for the programmer to manually develop software that efficiently uses the architectural resources and capabilities offered.

This assumption is enforced by an Internet forum post citing a TI C compiler developer: "*His answer [to the question why there are no other compilers supporting C6000] was that when TI developed the C6000 core, it was a side-by-side development effort by both the hardware and software developers, and that the C compiler by necessity would be rather complex and hence very hard for a third party to develop.*" [11]

3.3 Comparing TriMedia and TI C6000

***Remark**: unless otherwise specified, the information in this section is taken from a number of Texas Instruments reference documents* [7][8][9][10] *and the main TriMedia paper by Van de Waerdt, et al.* [3]

TriMedia and TI C6000 achieve high performance, low power consumption and low unit cost by implementing the architectural features described in § 2.1. However, the details of how they accomplish each feature may differ between them. Each architectural feature implies design compromises that were sometimes solved differently in each processor.

Most of the architectural design choices can be divided by the goal they try to reach: high performance or low power consumption. Although these two goals certainly overlap at times, for clarity the main comparison between the two processor families is divided as such.

3.3.1 High performance

Parallelism on TriMedia — The algorithms required to implement video processing codecs are suitable for parallelization using VLIW and SIMD. They typically require similar independent operations on multiple small data words. TriMedia processors provide instructions and super instructions that accomplish typical tasks in these codecs in few cycles and with minimum memory bandwidth consumption. The VLIW in TriMedia is implemented with 5 issue slots and compressed with a 10-bit template field.

Parallelism on TI C6000 — VLIW and SIMD provide instruction and data level parallelism in the TI C6000. In addition, **compression** through stop bits is supported: parallelism of instructions in

the fetched 256-bit VLIW can be controlled by using a stop bit, when setting the least significant bit (LSB) of each of the eight contained individual instructions to either 0 or 1. When set to 1, the individual instruction will be executed in parallel with the subsequent individual instruction. This allows eight instructions to be executed fully serial, partially serial or fully parallel.

Large register file on Trimedia — Video processing requires working with a large data set; a large register file prevents overuse of expensive load and store instructions. TriMedia processor TM3270 has a unified register file with 128 32-bit registers.

Specific domain ISA instructions — Not all tasks found in video processing codecs are parallelizable, for example in H.264 Context-Based Adaptive Binary Arithmetic Coding (CABAC) intrinsic sequential behavior cannot be properly optimized with SIMD. The TriMedia ISA is equipped with several native instructions to simplify CABAC programming to compensate for this disadvantage and minimize sequential execution. TI C6000 provides also specific domain instructions designed to optimize the execution of image processing kernels.

Fixed-point arithmetic — On the TI C6000 processors fixed-point arithmetic is less computationally demanding than floating-point arithmetic, and thus a suitable design choice to increase arithmetic processing speed.

Predication — Both families of processors allow instructions to be executed conditionally (predication), thus reducing costly branching.

Delay slots — On the TI C6000, for fixed-point instructions, a number of delay slots are available. The number of delay slots is equivalent to the number of additional cycles required after the source operands are read for the result to be available for reading: for a multiply instruction this is 1, for a load this is 4, and for a branch this is 5.
On TriMedia, five delay slots are provided only for branch instructions.

3.3.2 Low power consumption

Clock gating and frequency scaling on TriMedia — TriMedia applies two main power saving techniques: clock gating and voltage-frequency scaling. The latest Trimedia implements 70 clock domains, for example all stages of all functional units are gated. The normal supply voltage is 1.2V but the TriMedia guarantees normal operations at 0.8V at a lower frequency. The maximum operating frequency for the TM3270 is 350Mhz, ample room for scaling down its frequency when processing typical tasks such as MP3 decoding (only 8 Mhz needed).

Clustering on TI C6000 — On the other hand, the TI C600 family, to avoid a slow and power-hungry register file (read / write ports from each register to each of the 8 FUs) and a large forwarding network, two data paths (i.e. clusters) are available. This allows higher frequencies and lower power consumption.
TriMedia's register file is twice the size of the TI C6000 and is unified. In this way TriMedia designers choose to save memory bandwidth with a large register file and TI C6000 designers choose to save on power consumption and to reach higher frequencies with a smaller divided register file.

Memory Access — On TI C6000, 4 interleaved single-ported memory banks assure lower power consumption. This however can lead to reduced performance in a VLIW architecture: only one access to each bank is allowed per cycle; if two parallel load instructions are both trying to access the same bank, one load must wait, resulting in a memory stall.
On TriMedia, low power consumption is also accomplished with a 4-way set associative cache.

4 SDR processors: SODA and EVP

4.1 SODA

***Remark**: unless otherwise specified, the information in this section is taken from the main SODA paper by Yuan Linet al.*[13]

4.1.1 Introduction

SODA (Signal-processing On-Demand Architecture) is a (proposed) SDR processor architecture developed at the University of Michigan. Its main design goals are: high performance, energy efficiency, and programmability through a combination of features that include single-instruction multiple-data (SIMD) parallelism, and hardware optimized for 16-bit computations.

The proposed architecture has been implemented on 180 nm process technologies, and it is projected to meet the throughput and power requirements of current wireless protocols (see Figure 1) when implemented on 65 nm.

4.1.2 Architecture overview

Overview

The SODA system is composed of (see Figure 6): four **processing elements** (PEs), a scalar ARM **control processor** and a global **scratchpad memory**, all connected through a shared bus.

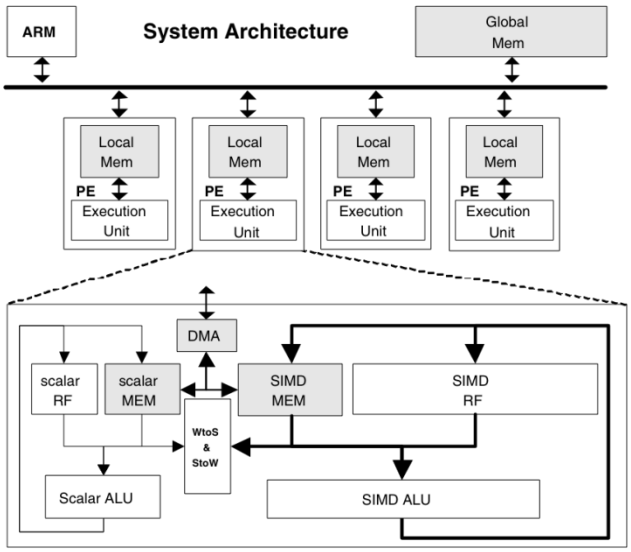

Figure 6 – Overview of SODA multi-core DSP architecture

Processing Element

Each PE consists of 3 pipelines that run in lockstep: a wide SIMD pipeline, a scalar pipeline and an AGU (address generation unit) pipeline.

- The wide **SIMD unit** runs most of the compute intensive algorithms (FFT, Viterbi, Turbo decoder, etc.). The SIMD pipeline consists of 32 clusters with a 16-bit datapath (see Figure 7). Each cluster contains 1 ALU and a simple register file with 2 read ports and 1 write port. A shuffle exchange network interconnects the clusters.
- The **scalar unit** is used to support many of the DSP algorithms that are scalar in nature and cannot be parallelized. The scalar pipeline consists of one 16-bit datapath.
- An **AGU** (address generation unit) **pipeline** handles DMA transfers and memory address calculations for both the scalar and SIMD pipelines.

The SIMD Shuffle Network (**SSN**) support intra-processor data movements. It contains a shuffle exchange network and an inverse shuffle exchange network, which allows to efficiently perform permutations on vectors, a common task in wireless communication algorithms.

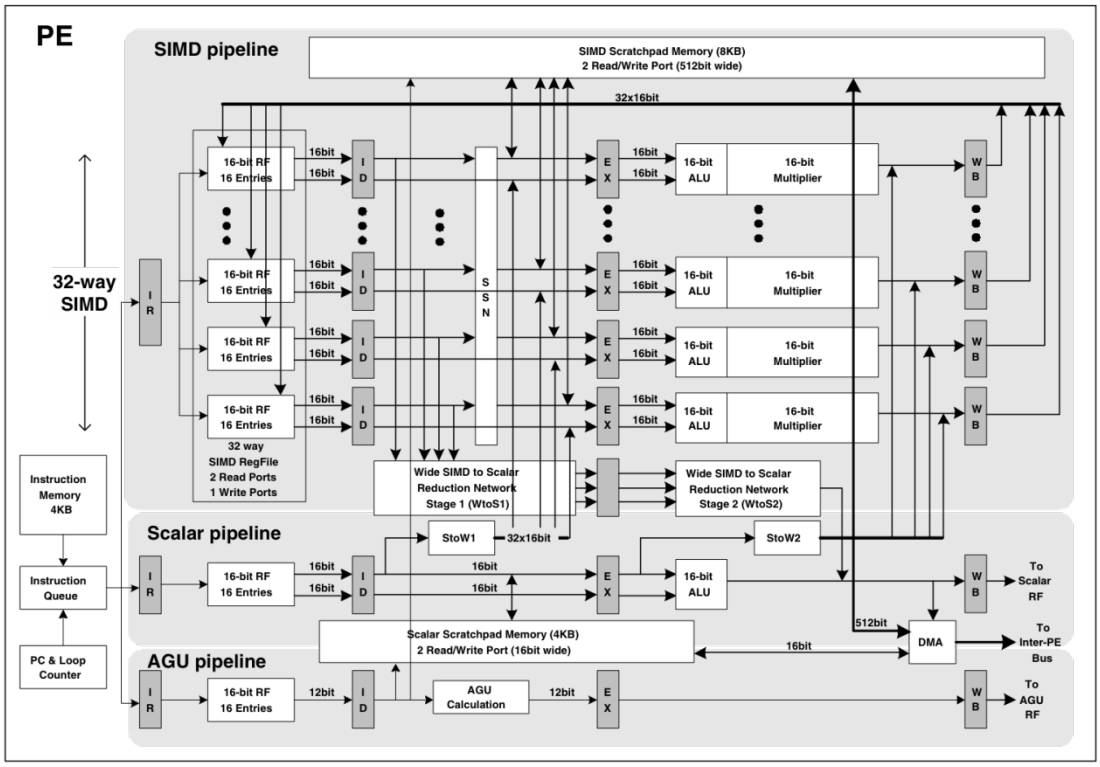

Figure 7 – Processing Element architecture

Control processor

The ARM processor is mainly used as a system controller to handle protocols' control code and state transitions.

Memory

There are no cache structures: both the **local PE memories** (4 KB scalar, 8 KB SIMD) and the **global scratchpad memory** (64 KB) are managed by software through each PE's DMA controller to which all local and global memories are visible.

The SIMD memory contains two 512-bit ports (one read, one write). The scalar memory contains two 16-bit ports (one read, one write).

Communications between PEs are via scalar streams, but intra-PE computations are vector operations. Therefore, support for a scalar-vector interface between the scalar and SIMD pipeline is necessary. This is shown in Figure 6 as **WtoS and StoW**.

4.1.3 Compiler support

It is clear that heterogeneous multiprocessor chips such as SODA provide difficult challenges for programmers and compilers to efficiently map applications onto the hardware. Especially since many of the projected high-performance features rely on static scheduling.

A new dataflow programming model, called SPIR (Signal Processing Intermediate Representation), designed for modeling SDR applications has been researched by Lin et al[14]. It acts as an intermediate representation that can be automatically generated from existing high-level languages (such as C) through a compiler front-end.
The general idea is that SPIR represents a task graph consisting of a set of nodes (tasks, for example a descrambler) interconnected with arrows (dataflow). Each arrow specifies the input and output stream rates for the source and destination nodes. This property allows a compiler to generate static execution schedules.

4.2 EVP

***Remark**: unless otherwise specified, the information in this section is taken from the main EVP paper by* Van Berkel et al.[16]

4.2.1 Introduction

Embedded Vector Processor (EVP) is an application specific processor developed by NXP. The EVP was developed almost exclusively for SDR applications and is aimed mainly at consumer handheld devices such as PDA, Smart Phones, Mobile Phones, etc.

EVP has been designed to execute the typical algorithms found in the modem stage (see § 2.2) in real time and in a power efficient manner. EVP accomplishes this with data and instruction level parallelism (vector processing and VLIW) and by providing application specific instructions.

4.2.2 Architecture Overview

Figure 8 shows a block diagram of the EVP architecture. The processor consists of a general scratchpad program memory, a VLIW controller, an Address Calculation Unit (ACU), a vector register file, a scalar register file, a vector memory, a set of vector functional units (FUs), and a set of scalar functional units. The main word is 16 bits.

The VLIW controller dispatches a **VLIW** instruction over the various vector and scalar functional units. The maximum VLIW-parallelism available equals five vector operations plus four scalar operations plus three address updates plus loop-control.

The vector FUs support **SIMD**, allowing for data parallelism. The SIMD width of the EVP is 16 (256 bits). This implies that the EVP can execute the same instruction on 16 words of 16 bits, 32 words of 8 bits or 8 words of 32 bits.

Some of the vector FUs are very specific: the MAC unit provides for Multiply-Accumulates, the shuffle unit can rearrange the elements of single vector, the intravector unit provides vector specific operations such as determining the maximum element of a vector, the code generation unit provides support or CDMA specific functionality (i.e. so called "code chip" generation).

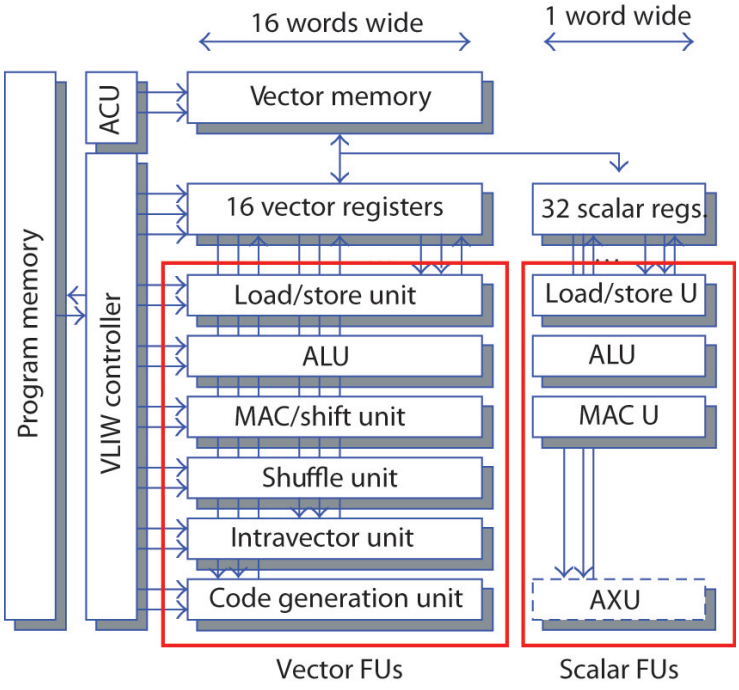

Figure 8 - EVP Architecture

4.2.3 Compiler Support

The EVP comes with its own language and own compiler. EVP programs are written in EVP-C a superset of ANSI-C. C programs are mapped exclusively to the scalar units of the EVP. To exploit the VLIW, SIMD, and vector parallelism it is mandatory to use the EVP language extensions and some libraries. It is still a task of the programmer to manually map the DSP baseband algorithms to their "vectorized" version in EVP-C.

Furthermore, manual optimization is required to achieve *efficient* vectorized code. As an example, the code produced by a prototype EVP-C compiler for a specific FFT implementation required 25% more cycles than hand-scheduled assembly code. [16]

4.3 Comparing SODA and EVP

This section will discuss why certain design decisions were chosen for both SODA and EVP.

4.3.1 Design goals

As stated in § 2.1, the main design goals for an SDR processor are high performance, energy efficiency, and programmability. The table below summarizes the techniques used to achieve this. More detail on the different techniques is provided in the subsequent paragraphs.

Unless otherwise mentioned, all techniques and discussions apply to both SODA and EVP.

High performance	**Energy efficiency**	**Programmability**

• Multiple parallelism
  • VLIW
  • SIMD
  • Multiple cores (SODA only)
	• No branch prediction	• VLIW / SIMD

• Hardware optimized for 16-bit fixed-point operations	• Fixed-point operations	
• Scratchpad memory	• No cache	
• Special DSP instructions	• Special DSP instructions • Separated SIMD memory	

4.3.2 Wireless protocol characteristics

Wireless protocols are characterized by a number of specific properties, which have an important impact on the design of a DSP system.

- **High Data-Level Parallelism** — Most of the computationally intensive DSP algorithms have abundant data level parallelism (for example the "searcher" in a W-CDMA protocol, can be represented by 320-wide vectors), much more than instruction level paralellism. [13] Parallelism is elaborated on in § 4.3.3.
- **8 to 16-bit data width** — Most algorithms operate on variables with small values. Analysis of two typical wireless protocols shows that there should be strong support for 8 and 16-bit fixed–point operations. 32-bit fixed-point operations and floating-point support is not necessary.[13]
- **Real-time requirements** — Strict real-time requirement in wireless protocols requires deterministic architectural behaviour. Therefore features such as **caching**, **multi-threading** and **prediction** are not well suited. [13]
- **Vector operations** — Intravector operations (vector reductions) and shuffling of data within a vector is key to a number of common algorithms (e.g. FFT). Therefore specific, power and performance optimal, support for these operations should be provided. [16] This is elaborated on in § 4.3.4.

4.3.3 Parallelism

The key to achieve the high performance required for SDR is to maximally exploit the parallelism available.

SODA provides three levels of parallelism: multiple cores (PEs), VLIW and SIMD. [13]

- **Multiple cores** — Periodic real-time deadlines require algorithms to compute with different data rates and different latencies. Realizing this for complex protocols using a single threaded system is too expensive (for example, complex context switching software).

However, it is necessary to support efficient concurrent DSP algorithm execution. Thereto multiple cores, implemented as PEs, are used. (Each protocol pipeline is broken up into kernels, and each kernel is assigned to a PE.)
Since the task in the wireless protocols analysed can be partitioned into 4 major task groups, 4 PEs has been chosen.

- **VLIW** — The three pipelines – scalar, SIMD and AGU – can be considered as an asymmetric VLIW pipeline (asymmetric, since e.g. scalar instructions can not be scheduled on the SIMD pipeline and vice versa). The scalar pipeline is necessary because there are many small, important scalar algorithms. Wide SIMD units would be too inefficient in supporting these scalar and narrow SIMD operations.
- **SIMD** — The computation intensive DSP algorithms in wireless protocols usually contain operations on very wide vectors. In addition, vector widths and strides are known at compile time. Although this makes a good fit for vector architectures, the extra hardware support for dynamic vector management is unnecessary, as all data operations can be *statically* scheduled. This favors a wide SIMD-style execution.
  The width of the SIMD unit is not chosen to be 32 clusters by accident. It is the optimal power consumption configuration, given the estimated computation requirement of 10 GOPS per PE. Figure 9 maps the power consumption to SIMD width. The required 10 GOPS can be achieved through a number of SIMD/frequency combinations, for example a 2-wide SIMD running at 5 GHz. High frequencies require unrealistic deep pipelines (indicated by 'I'), and suffer from high power consumption.

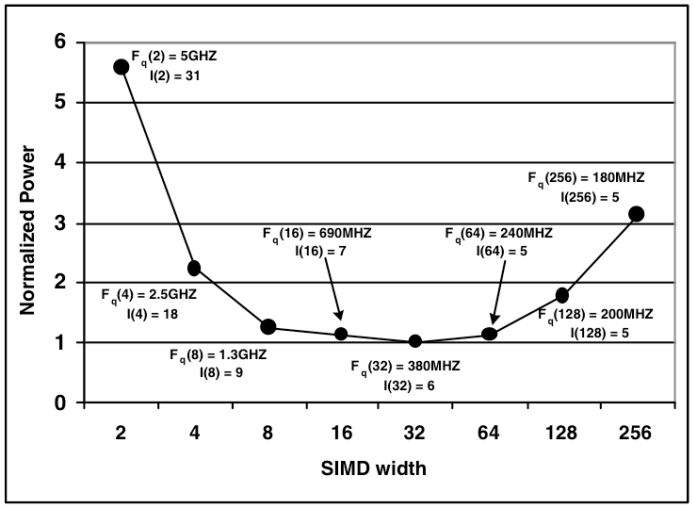

Figure 9 – Average normalized power requirements of a PE (180 nm)

EVP only provides two levels of parallelism: **VLIW** and **SIMD**. However, EVP differs in its VLIW organization: it can execute multiple types of wide SIMD operations in parallel. While this provides support for a higher degree of ILP, it also requires a more complex register file. Because there is very limited ILP among vector computations (i.e. data level parallelism is much more prevalent than instruction level parallelism), this extra level of parallelism does not add much to performance. [13]

The EVP SIMD width is scalable, but has been set to 16 for the first EVP prototype, labeled EVP_{16}.[16] The exact reason of choosing 16 is not specific as was the case for SODA. Of course, higher SIMD width will provide for higher data parallelism.

4.3.4 Special DSP instructions

SODA contains special DSP instructions, both to optimize performance and power consumption. Provided are special vector operations supported by the SIMD shuffle network (for example, the butterfly operation to enhance FFT performance) and vector to scalar operations (for example, determining the maximum value in a vector).
Multiply-Accumulate (MAC) instructions are *not* provided but instead handled by vector logic operations (predicated negation) since many of these multiplication operations are with 1 bit values, consuming significantly less power. [13]

EVP provides a number of specialized FUs (see also Figure 8). The *shuffle unit* provides functionality similar to SODA's SSN (SIMD shuffle network). The *intravector unit* provides functionality similar to SODA's special instructions for vector to scalar operations (for example it can also determine the maximum value in a vector). The *code generation unit* supports CDMA 'code chip' generation (a code chip is a fundamental unit of transmission in CDMA, a wireless channel access method). In a single clock cycle 16 successive complex code chips are generated. The unit is configurable for different standard (UMTS, GPS, etc.).
Contrary to SODA, the *MAC unit* provides support for MAC instructions. [16]

4.3.5 Other

No branch prediction — Branch prediction is not implemented, because for most DSP algorithms the core kernels consist of shallow nested loops. Instead, for SODA, a loop counter-based branch instruction is available. [13]

Scratchpad memory / no cache — To meet the need for high memory bandwidth and low power consumption, an on-chip scratchpad memory is used which is small and hence fast. With no data temporal locality (streaming data), cache structures provide little additional benefits, in terms of power and performance, over software controlled scratchpad memories. [13]

Separated SIMD memory — Both SODA and EVP have a dedicated SIMD memory. In general two memories consume lower power than one unified memory. In addition, this separation allows optimization of the read/write ports for its intended use (in the case of SODA: 512-bit for the SIMD memory and 16-bit for the scalar memory). [13]

Power consumption — At 180 nm, SODA's power consumption is 3 W. This is too high for embedded mobile devices (200 mW is a typical cellular phone's power budget for the physical layer). Scaling to 65 nm predicts the power consumption to around 250 mW. [13]
No practical comparable data is given for the EVP's power consumption, although EVP literature somewhat unclearly mentions that running certain UMTS protocols on the EVP results in a power consumption close to 1 W (at 90 nm); other protocols perform better. [16]

Both EVP and SODA literature does not mention support **unaligned loads**, **forwarding** and **compression**, which seems to indicate that it is not supported.

4.4 Final thoughts on SODA and EVP

It should be clear from the previous overview that key to achieving the performance required for SDR lies in exploiting as much as possible the available parallelism. How, then, can both SODA and EVP reach this performance, given that SODA is much better equipped for exploiting parallelism (running at approximately the same frequency)? SODA in addition not only has four cores, its SIMD pipeline is 2 times wider.

A response from both a SODA and EVP developer to this same question clarifies a number of things:

- Both cite the difference in VLIW organization — the EVP can execute multiple types of wide SIMD operations in parallel. [15][18]
- The EVP 'outsources' a number of algorithms to other processors (such as the Viterbi decoding), whereas SODA performs these functionalities on chip (i.e. on designated PEs). [15]
- For EVP to catch up with new and more demanding protocols, multiple cores are indeed necessary. [18]
- Finally, SDR is a relatively new research topic. It is lacking standardized benchmarks, and each wireless protocol has countless number of different implementations and configurations. These different configurations and implementations can affect performance/power significantly. Hence, it is very hard to compare two separated developed and tested processors. [15]

5 Literature

5.1 General

[1] Wikipedia

[2] Jason Fritts, Wayne Wolf and Bedde Liu, *Understanding multimedia application characteristics for designing programmable media processors*, 1999

5.2 TriMedia

The TriMedia section was based mainly on Van De Waerdt's paper. He designed this processor for his thesis therefore he was able to explain thoroughly many technical details with enough clarity and to the point. The remaining sources were used mainly to see the "big picture" of the TriMedia, they are less technical and more concerned with consumer applications and business aspects.

[3] Van de Waerdt, et al. *The TM3270 Media-Processor*. Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05). 2005

[4] Hoogerbrugge Jan et al. *Instruction Scheduling for TriMedia*. Philips Research Laboratories.

[5] Bores Signal Processing. *TriMedia Overview.* http://www.bores.com/courses/tm_overview/index.htm. Last updated: March 2007.

[6] Van de Waerdt, *The TM3270 Media-processor*, October 2006, TU Delft, ISBN 90-9021060-1, PhD Thesis

5.3 TI C6000

Texas Instruments provides extensive documentation on the C6000 platform and the differences between specific generations. These documents are in general very complete, however, information on why certain features are implemented are not included.

[7] Texas Instruments, *TMS320C6000 CPU and Instruction Set Reference Guide*, SPRU189d, July 2006

[8] Texas Instruments, *TMS320C6000Technical Brief*, SPRU197d, February 1999

[9] Texas Instruments, *TMS320C64x Technical Overview*, SPRU395b, January 2001

[10] Texas Instruments, *TMS320C62x DSP CPU and Instruction Set Reference Guide*, SPRU731, July 2006

[11] Internet forum post "*C compilers for TI TMS320 DSPs*", 2005- 3-02, http://www.dsprelated.com/showmessage/30734/1.php

[12] Jan Parthey and Robert Baumgartl, *Porting GCC to the TMS320-C6000 DSP Architecture*, Appeared in the Proceedings of GSPx'04, Santa Clara, September 2004

5.4 SODA

The paper "*SODA, A Low-power Architecture For Software Radio"* seems very complete and thorough, discussing the SODA architecture as well as the rationale behind the different design decisions, while also setting them of against the alternatives. It was lacking however in discussing how such a complex architecture can be reasonably programmed.

[13] Yuan Lin et al. *SODA, A Low-power Architecture For Software Radio*. In Proceedings of the 33rd Annual International Symposium on Computer Architecture, 2006.

[14] Yuan Lin et al, *Hierarchical Coarse-grained Stream Compilation for Software Defined Radio*, In CASES'07, 2007

[15] Personal email communication with Kees Moerman, 2008-12-16.

5.5 EVP

There was a very limited amount of information available in the Web on the EVP. This section was based mainly on Van Berkel's paper. This paper does not focus on the EVP but on SDR applications on vector processors and offers the EVP as an example; it does not explore in depth its architectural features. Moerman's article offers a brief business oriented view of the EVP capabilities and applications.

[16] Van Berkel et al. *Vector Processing as an Enabler for Software-Defined Radio in Handheld Devices*. EURASIP Journal on Applied Signal Processing 2005.

[17] Moerman, Kees. *Embedded vector processor is one way to tune software-defined radios*. Wireless Net Design Line. http://www.wirelessnetdesignline.com/202403292;jsessionid=5UPHUZ4YXPVRYQSNDLRSKHSCJUNN2JVN?pgno=1. Last updated: October 2007.

[18] Personal email communication with Kees Moerman, 2008-12-16.