УДК 004.273

A. V. Chapyzhenka, D.V. Ragozin, A.L. Umnov

LOW-POWER ARCHITECTURE FOR CIL-CODE HARDWARE PROCESSOR

In the article the authors present the architecture of a hardware CIL processor, which is capable to execute CIL instructions as native code. The CIL hardware engine is implemented on the top of the low-power DSP architecture, and the CIL processor has two execution cores: DSP and CIL. Such solution allows to execute both CIL and DSP instruction sets as native instructions sets and gain performance in common multimedia tasks. Therefore, the DSP-based CIL processor may be targeted for multimedia digital home and even embedded applications. The research was sponsored by RFP 2 Microsoft Corp. grant.

Introduction

The use of the stack-based hardware microprocessor, which represents stack-based programming model like Java virtual machine (JVM) or CIL virtual machine (CIL-VM), has both advantages and disadvantages. Although the stack-based programming model has advantages in uniformity: instructions have no address field, stack effects are predefined, the instruction set is homogeneous – the stack is a serial device, where the maximum execution speed can not exceed one instruction per clock, because any two consequential instructions always have dependence on the top-of-stack cell. But despite some computational inefficiency, the stack processor model is well suitable for virtual machine implementations. In internal representation the program is usually represented as a sequence of addresses of procedures and instructions. If the stack machine instruction can be represented by a standardized number, the software may be executed on any processor which has a table, representing the relation between the standard instruction set and local instructions. This stack machine principle is the basis of Java and CIL machine concepts, and allows us to have a semi-machine-independent way of software execution. Also, the code generation for the stack-based execution model is quite simple, and some earlier internal compiler program representations were developed with the use of stack-based code.

From the point of view of hardware, the stack machine is the simplest way to execute machine codes. There is no address information, instruction decoding is very simple, and instruction execution cycle is very short, because there are no additional multiplex switches for read and write ports of the register file. The stack machine register file consists of the top-of-stack register with a read and write port and the under-the-top-of-stack register with a read port (fig. 1). Such a structure has limited ability for parallel execution, but has small complexity and low power consumption, because there are no large multiplex switches for read and write ports of the non-existent register file.

The hardware execution of code, based on stack utilization paradigm, has nearly forty year long history. The basic concepts of hardware stack-based processors were developed in early 60s-70s [1], but the software development languages resembled the Forth [2] language – a kind of high level macroassembler for basic stack machine instruction set. The best known stack processor is Patriot Scientific PSC-1000A [3], which is intended for Java program

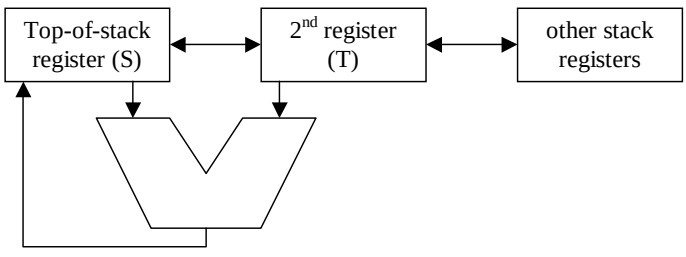

Fig. 1. Structural scheme of the stack processor

© Chapyzhenka A., Ragozin D., Umnov A., 2005

execution. PSC 1000A is a typical stack processor and has typical stack machine advantages and disadvantages

So, there are good technical solutions for the basic CIL-code hardware processor implementation. At this development stage meta information utilization is not considered, because in extreme case all meta information checking instructions may be implemented as a low level microcode. A full featured CIL-code processor may be implemented as software on top of a very simple (basic) stack processor.

Here we have a very important problem – what the trade-off between the simplicity of hardware implementation and execution efficiency is. This problem escalates other important problem – what kinds of tasks must be executed efficiently on the target hardware processor? Each hardware must have the target market, and concrete hardware implementation must be driven by target applications and the target market.

The paper consists of 4 parts. Part 1 describes the target market and possible applications for the CIL-code processor. In Part 1 common trends in processor architectures are discussed. Part 2 describes a case for the CIL-code processor core implementation. Also the kernel implementation of the CIL processor is discussed, jointly with additional features, such as caches for meta information, which are intended to speed up the CIL-code processor. In Part 3 software decisions for supporting the CIL processor are considered. In Part 4 possible system software implementations are discussed. In conclusion we outline our plans for the future.

1. Application target for the CIL-code processor

At present the .NET technology has such a direct competitor as Java technology, which is used primarily in such mobile devices as smart cards and mobile phones. Different PDA devices, Internet terminals, home desktops use mainstream processors such as Intel® x86 family, ARM-based chips and other devices. To start targeting the CIL processor we will discuss both markets: the “computer” market and the mobile devices market.

The market of PDAs, laptops, desktop computers requires processors with usually high computational power, despite the consumed electrical energy. E.g. the latest 3.4-3.6 GHz processors with 85-90W power consumption are available on the market now, and the upper limit for frequency for the nearest years is near 10 GHz. The number of distinct architectures has become very small: Intel® x86, IBM Power, Sun UltraSPARC, Intel® Itanium families, most of them are based on the superscalar architectures with specialized components for improving available scalar parallelism such as multithreading. On the other hand some architectures are specialized for high performance computations such as Itanium with explicitly defined parallelism. Some hardware expert say that there is no room for new competitors at the desktop processor market.

The market of mobile devices is different from the “computer” market in its requirements to the computational units: it requires high computational power with very low energy consumption. The maximum consumed energy is computed from the capacity of particular batteries and the time of device uninterruptible work. So, the current consumption of the processor must be between several milliamperes and tens of microamperes. The number of possible architectures exceeds thousands because the market accepts any developed technical solution, even if the new processor architecture has unique peculiarities that involve full reprogramming of the application. Good examples are VLIW-chips, the chips with specialized integrated devices, DSP chips.

To concern possible hardware platform, some considerations about the possible target market for the CIL processor based devices must be done.

1.1. Target market for the hardware CIL processor. We will start the discussion by some real life example. Different handheld devices are very popular in Japan. Many standards, like 3G for mobile phones, and the phones with ability of video data transmission were originally started and originated in Japan, as well as different Internet-related services such as chats and news over mobile phones (before wide spreading of Wi-Fi hot-spots). On the other hand, the handheld devices and PDAs aren’t so widely spread in other countries, in comparison to Japan. Some sociologists say, that Japanese people spend too much time

during their morning trips on the electric trains to the office and evening trips home. The latest electronic devices are used for entertaining people during their trips (up to 2 hours) to and from work.

So, some type of the devices may be intended to serve specific needs of a particular groups of people. The targets of the .NET technology are similar to those of Java technology. The .NET software is intended for the execution model "once written – executed on any platform" for the software transmission over the Internet and safe execution on different hardware platforms. But primarily .NET technology is a newer programming model, intended for different Web-oriented services, distributed business databases, online transactions, CRM system support, logistic support, etc. Also .NET is a good technology for different computational devices' convergence - .NET software must be enabled for mobile phones, PDAs, different handhelds, notebooks, desktop computers; it must be delivered over all possible network connections: cable and wireless. Here the .NET technology is the basis technology for connecting different devices, that is why any application consists of several thousands of classes combined into assemblies, and can be loaded up to some mobile device on demand. Every mobile device can have only basic set of integrated .NET classes, other necessary classes may be downloaded from the application servers on demand. Depending on available resources of the mobile device, some assemblies can be cashed (if the device has enough memory). A detailed technical description of application transmission via Internet is beyond the scope of our paper.

Due to the fast growth of wireless technologies, all notebooks, most of desktop chipsets, PDA devices will be have some kind of wireless adapter in a year or two. The necessity of different devices integration into uniformly managed network with automatically organized intranet, personal data sharing and future mobile agent software model leads to future convergence of technologies and involves the use of an integrating software paradigm like Java or .NET technologies. In case of mobile devices, Java is supported in Java oriented coprosessors, such as [3] or technology like Jazelle [4], where two additional pipeline stages are integrated in ARM processor. These two pipeline stages decode Java instructions so that their direct execution over the ARM RISC-type architecture is implemented. The hardware CIL processor allows direct support of .NET software inside the different mobile devices.

There are several basic assumptions that must be considered first. The hardware CIL processor can not be a competitor of desktop processors, mobile processors and PDA ones, but the end-user specialized devices, like Web-terminals, Web-browsers, interactive television, house control systems and other hardware are the proper target. The desktop computer processors and the notebook processors have too much computational power, on the other hand mobile phones aren't ready to handle big applications. So, the hardware CIL-processor must target an intermediate position between the mobile phone processor and powerful processors for PDAs, like XScale (600 Mhz) or TI OMAP (ARM core + DSP C55 core). For example now video playback is not common for most mobile phones (even high resolution color displays aren't used in nowadays mobile devices), and the screen size is not enough to handle most database-oriented applications. So, the proper suggestion can be made that the target for the hardware CIL processor should be mobile phones and PDA devices, oriented for distributed Intetner-related content processing.

There are important additional requirements to the CIL processor. First, the CIL processor must be capable of operating with different data, because the modern Internet content includes many video and audio files, animation. Without such files the Internet content is not very impressive for a typical user. So, the CIL processor must handle such tasks as MPEG1/2/3 files playback, MPEG4 and DVD video playback, Flash playing, JPEG encoding and decoding, utilizing 96kHz 6-channel audio and 1280x1024 screen resolution, 32-bit color. Second, the energy consumption of the CIL processor must be very low. It is a very important marketing issue, and there is a good modern example. Before the appearance of Pentium® M processors (Centrino) most notebooks could operate using battery only for 2-2.5 hours, so that a mobile user had to have an additional battery feeling uncomfortable because of the short work time of the notebook.

The Pentium® M energy saving technology brings the user up to 5 hours of time (that allows taking the notebook from office to home without power supply in the evening, reading and answering e-mail stored into Outlook, editing some documents, and finally seeing new thriller on DVD before going to bed). The power consumption technology improvement appeared even more important than integrated Wi-Fi network adapter.

Due to the nature of the CIL processor based on the stack virtual machine, most of the DSP-related operations (required for multimedia processing) will be implemented with high penalties in execution in comparison with traditionally used RISC and superscalar processors. So, the challenging task is the implementation of the CIL supporting processor with low power consumption, which also has to handle DSP tasks effectively.

After positioning the hardware CIL processor in the market, the next thing to discuss is existing (embedded) processor architectures and the basic ideas that can be used for the CIL processor implementation.

1.2. Discussion of available processor and cores at market. The stack paradigm used the in .NET specification [5] suggest the idea to try a stack processor as the prototype for the CIL processor implementation.

The stack processor has a significant limitation on the available parallelism because arithmetical expressions can be computed only sequentially (but may be computed in parallel on several stacks inside one processor) at each stack implemented in the processor. Also, the formal stack paradigm does not operate with address registers (there is at least one address register in minimal stack processor), but most of DSPs and general purpose processors have 4-8 address registers with different modes of address arithmetic. The arithmetical stack is usually overloaded with addresses update operations (an example is given below) if the address registers are not implemented.

The limited ability for parallelism is an inherent peculiarity of the stack expression evaluator – by definition. That is why a routing scheme (multiplexer) for transferring data from stack top registers to ALU and back to the top-of-stack register is not used, buses are used instead. The memory is connected to these buses by 2-to-1 multiplexers (routing 2 data path into one). So, the ALU operation execution scheme is very low power consuming, because the energy is consumed only while data change occurs between stack registers and ALU functional units. Usually the functional units are scattered over the die and there are no local sources for overheat. That is why in case of the stack processor there is limited parallelism but low energy consumption.

Well-known RISC processors and cores, like ARM, SPARC, MIPS and well-known extended DSP processors (AD SHARCs, TI TMS) have quite a large register file. But, RISC instructions usually have a regular structure, and every register may be commutated to any input of any functional unit. Also, the outputs of any functional unit can be commutated as the source to any register. The commutation system (the multiplexer) for the register file is very complex for any RISC–paradigm register file. The complexity of the multiplexer structure grows as quadratic function of the number of registers and read/write ports (e.g. AD SHARC-21k processor has 10 read ports and 6 write ports and 16 fourty-bit general purpose integer/floating point registers), and the overall length of wires grows as cubic function of the number of registers in the register file. The consumed energy is formed from two sources: the energy consumed by switching transistors (quadratic function) and the energy consumed by leakage in wires – a wire has internal capacitance and a leakage of current because the surrounding dielectric does not have ideal characteristics. Thus the main source of the high energy consumption is the large number of transistors in the multiplexer and the big length of wires. (Precise formulas for energy consumption calculation can be found in [6]. Due to the nature of the RISC processor, the solution of the problem of the high energy consumption is not trivial because the multiplexer is very large, the number of register and read/write ports is big and depends on available parallelism inside the chip. Any attempt to decrease the number of ports leads to decrease of the potential parallelism of the RISC processor (fig. 2).

For example, to count resources needed for a typical DSP procedure, the resources necessary for convolution computation will be

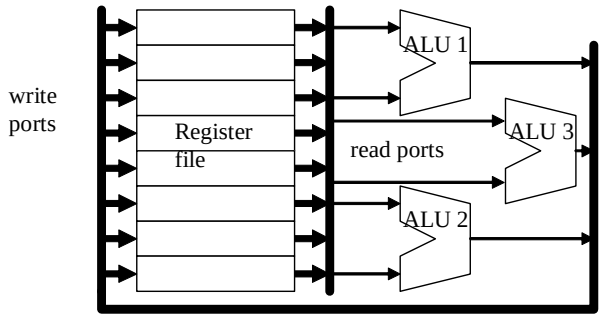

Fig. 2. Structure of the arithmetical unit of RISC processor

counted:

2 memory accesses require: 2 write ports, 2 data buses to memory; 2 read ports for the address register file;

2 address increments require: 2 write ports for the address register file;

1 multiplication and 1 addition require: 2+2 read ports, 1+1 write ports *or*

jointly 1 multiplication + addition (MAC) require: 3 read ports, 1 write port.

Finally, there are 6 (5 if the MAC is used) read ports (2 ports for the address register file) and 6 (5 if the MAC is used) write ports (2 ports for address register file) for the convolution operation.

The most convenient approach for decreasing the size of the register file multiplexer is specialization of the register files. The address register file, the common precision register file (small sized), the accumulator register file (2-4 registers) could be detached. Additionally the address register file is partitioned into two parts, each part controls a distinct memory space. Finally, each part of the address register file has 1 read and 1 write port, the arithmetical register file must have 2 write ports and 2 read ports, accumulators must have 2 read ports and 2 write ports (or 1 write and 1 read ports depending on implementation of the MAC unit). The typical structure of the register file on DSPs is shown at fig. 3.

Here the register file is partitioned into chunks, which are “strongly” connected to particular ALUs (functional units). But, in DSP case the multiplexer size is reduced, so the time for signal passing through the multiplexer is reduced and the multiplexer power consumption is also reduced significantly.

As should be noted, the superscalar RISC and CISC processors are not considered, because they have too much energy consumption per MIPS (MFLOPS), have very large die size and need an external chipset for functioning properly and they have low abilities for external devices support.

Before possible architectures are discussed a reference must be made for research projects devoted to hardware implementation of object-oriented processors. The CIL abstract machine supports a rich and powerful class and object model, which represents the sum of object oriented paradigm implementation practices. Even internally CIL uses objects, utilizing “pure” machine types such as integers and addresses only for compatibility with external APIs. Another important issue is the garbage collector, which services are the only way to allocate memory block for storing an object.

The CIL processor is not the first chip with the support of the object oriented model.

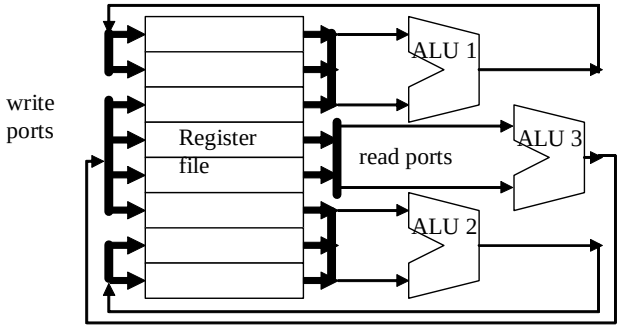

Fig. 3. The structure of the arithmetical unit of the digital signal processor

The first truly object oriented language is the Smalltalk-80. Many researchers tried to define and implement an effective architecture for the Smalltalk code execution, such as Swamp [7] or Sword32 [8]. The main difference between the Smalltalk execution engine and the CIL engine is that Smalltalk was not projected for effective execution on the microprocessors. E.g. all internal types such as integers and floats are treated as objects. The paradigm is very convenient for an application programmer, but extremely complicated for hardware implementation of the Smalltalk processor. To overcome the difficulties of the object oriented paradigm handling the processor uses a special pointer to the current object ("this" in the C++ language) and pointer to the object class run-time information. So, a method call or a field access would be performed much faster with the use of the cached *this* pointer. The projected CIL processor must also have the abilities to speed up the object oriented paradigm instructions, but the paradigm support must be localized for the CIL execution engine.

1.3. Architectures of low power consumption processors and benchmarking results. According to the requirements to the CIL processor, it must support a stack-based instruction set, effectively handle DSP tasks and have low power consumption. That is why much attention will be paid to the large growing market of DSP processors, which satisfies the latter two requirements for the CIL processor (efficient DSP and low power consumption).

A very good study of different DSP processor architectures was performed in BDTI (Berkeley Design Technology, Inc) DSP labs [9]. For more than ten years BDTI specialists have been publishing the "DSP buyer guide", where DSPs and DSP cores are shortly discussed. BDTI covers DSPs available on the market at present time, the peculiarities of different DSPs, the computational efficiency of different algorithms, the energy consumption of different DSP algorithms and other business issues of DSP processors, answering the question "What DSP should I buy?".

Considering the BDTI labs reports about DSP market during 2004 [10], we can find at least four large groups of DSPs on the DSP market now: 1) VLIW-DSPs with high peak performance, 32-bit datapath; 2) high-performance classical DSPs and RISC-like DSPs with 32-bit datapaths; 3) low-end classical DSPs with 16-bit datapath; 4) low-end DSPs with controller abilities.

The DSPs with the highest peak computation power are VLIW-DSPs, such as well-known Texas Instruments TMS320Cx60 family. The TMS Cx60 family has frequencies up to 700 Mhz (2003 year), several chips (C6415) have an integrated floating-point unit and a PCI controller. TMS Cx60 chips have VLIW architecture with a variable-length long instruction word. The chip can execute up to eight atomic instructions per clock and has register files, partitioned into 2 parts, A and B, each part has 16 registers (Cx62/Cx67 chips) or 32 registers (Cx64 chips). The energy consumption of the Cx60 processors is quite high, because the chip has a large register file, caches on its die and a cross-bar unit for handling VLIW instructions, therefore sometimes an external cooling engine is required. The competitors of the Cx60 chips are chips TigerSHARC (by Analog Devices) which have an integrated SIMD engine for multiplying and accumulation numbers, with an operating frequency up to 400-500 Mhz (2004). Another high performance processor is Star Core SC140 chip from Motorola, a multi-issue processor with 4 computational units, which can execute up to 8 atomic commands per cycle, it has 16 general purpose registers and 16 address registers. All those high performance chips have high computational power but high energy consumption in comparison to classical DSP processors.

A wide category of chips are classical DSP processors and DSP processors, which have like-RISC architecture. The Analog Devices ADSP-2106x processor is an example of RISC-like DSP processor. It has a large register file (16 registers), fourty-bit wide, which can handle 32-bit integer numbers or 40-bit extended single precision floating-point numbers. One operation is encoded in a processor instruction usually, but the chip can execute a memory access (dual) jointly with a computational operation or a dual memory access jointly with a computational or MAC operation. For handling MAC operations, the register file is partitioned into 4 chunks, each consists

of 4 registers. The analogous processors on the market are the TMS Cx54 from Texas Instruments, the DSP56311 chip from Motorola, the DSP 16000 chip from Lucent/Agere. These processors provide 32-bit computations, are capable of video operating, audio decoding, etc. The capabilities of these processors are enough for most multimedia tasks, listed in the first part of current chapter.

A good example of the latest trends in classical DSP architectures is Micro Signal Architecture (MSA) developed jointly by Analog Devices and Intel Inc. [11]. Now it is produced by Analog Devices and named BF-535xx "Blackfin". It is based on modified Harward architecture. Its data arithmetic unit contains eight 32-bit registers, each can be used as two 16-bit registers. The ALU of the processor contains two 16*16 multipliers, two 40-bit split ALUs, a 40-bit shifter, four 8-bit video ALUs and two 40-bit split accumulators. The accumulators are separated from the main register file. The address generation units contain six 32-bit general purpose registers, four 32-bit address register for circular buffers, a frame pointer and a dual stack pointer for user and kernel spaces.

Also, there are low-end 16-bit classical DSPs such as ADSP-2189, Motorola DSP1620, etc, which are used for different telecommunication applications for 8-bit or 16-bit sound samples processing. These processors are used if there is no need for high performance DSPs and there are strong requirements to low power consumption.

The additional class of DSP processors is mixed DSP controllers, such as MicroChip dsPICs. Their appearance was caused by the necessity of mixing DSP processing and different controller tasks. These chips usually are microcontrollers, improved with the second memory bus, the MAC and the dual memory access instruction for providing the convolution kernel execution in one instruction. The clock speed of the DSP controllers is near 30 – 40 MHz.

Another large piece of the processor market for mobile devices is processors, which appeared due to the PDA market growth. The best representative of this market piece is Intel® XScale, based on well-known ARM architecture, it is efficient and low-power RISC processor. But, in comparison to most DSPs used in cellular phones, XScale processor, the kernel performance of which is comparable to the performance of the ARM7 kernel, has higher power consumption and lower performance on typical DSP applications in comparison with classical DSPs. XScale performs DSP jointly with a special coprocessor, which has only one bus to the memory, so can not perform typical operations as efficiently as DSP.

As examples of performance cores, discussed in the BDTI 2004 year report [12,13], chips Texas Instruments Cx64, Motorola StarCore SC140, Texas Instrument Cx55, Analog Devices BF53x "Blackfin", Texas Instruments OMAP, and Intel Xscale PXA2xx are discussed. The basic performance marks for the processors are showed in the table 1.

In the first column there are the special marks: "(+)" and "(-)". The "(+)" denotes that the higher value in the row is the better, the "(-)" denotes that the lower value in the row is the better. Here the top speed processor is an 8-issue VLIW TI C6414 from the C64 family. It has the highest frequency and can execute up to 8 simple instruction during one cycle. But it

Table 1. **Performance marks by BDTI Labs**

Mark	TI C5502/ 300Mhz, classic DSP	ADI BF53x/ 600Mhz, RISC-like DSP	TI C6414/ 720Mhz, VLIW DSP	Motorola SC140 /300 Mhz, VLIW-DSP	Intel PXA2xx, 400Mhz, RISC
Speed mark(+)	1460	3360	6480	3430	930
Memory use(-)	146	140	256	144	140
Perf/mW(+)	11.8	16.9	16.1 (300Mhz)	13.7	2.6 (200Mhz)
Cost/$(+)	146.2	375.9 (400Mhz)	98.3 (500Mhz)	29	25.6 (300Mhz)

has too much memory consumption, because of the large command words, moderate energy efficiency and poor cost characteristics. So, this processor must be involved in the application, where the performance is critical and the cost is not under consideration. Of interest can be also is ADI BF53x processor, which is 3-issue, has a high frequency, good energy efficiency and a moderate price. On the other hand, a 6-issue VLIW SC140 has less speed and is less energy efficient than the TI C6414, regardless of the chip frequency. But the SC140 has a high chip price. The TI C5502 has quite a low speed and a moderate price, but consumes low energy and has a low memory use. In comparison to other architectures, the XScale processor, based on RISC ARM kernel, has a high frequency but a low speed and very low energy efficiency at 200MHz frequency, and finally a high price.

So, the conclusion based on the table would be quite unpleasant for commonly used general purpose processors (GPPs): most RISC-based processors aren't so effective in the energy consumption and aren't effective for DSP tasks. Even the most popular RISC-based XScale, involved by success of previous ARM families, which are famous for the good speed and the low energy consumption, loses 5-8 times in media tasks in comparison to classical DSP families. Inside the DSP families it is noticeable that the DSPs have good speed characteristics because of the irregular parallelism, used for computations. Common DSP processors are 3-issue and can execute 2-3 instructions per cycle inside the DSP kernel. The most important peculiarity of most DSP instruction sets is the ability to execute an arithmetic and a data movement instruction in parallel. The high-performance DSPs with VLIW architecture executes at least 4 instructions in parallel. E.g. the Motorola SC140 is a 6-issue processor; the TI C6414 is an 8-issue processor. The surprisingly good results of the Blackfin chip can be explained because it also has the frequency which is twice as higher as the frequency of other chips.

The speed marks for some other processors are in the table 2 [13].

The good results are shown by the Agere Systems DSP16xxx kernel, the Analog Devices TigerSHARC high-end processors, and the Texas Instruments C6414 processor, which has a fantastic frequency 1 GHz.

From the point of view of the CIL hardware processor there is no urgent need for the highest frequency. To make use of fine-grain parallelism hidden in common CIL code it is necessary to have a superscalar parallelizer with shadow registers, reservation stations and other stuff. The parallelizer is a complex thing, it consumes much energy and requires many extra thousands of transistors on the die. Because of its high energy consumption a superscalar processor can not be used inside mobile devices in case of current technology. But without a parallelizer engine, the "wide" parallelism, which is available on 6-issue and 8-issue VLIW processors like the SC140 and the TI C6414 can not be utilized. Also, the VLIW-DSP energy consumption is quite high, but the increased computational power allows to have energy efficiency comparable to classic DSP processors. So, a good preliminary solution is to use DSP as the basis for the hardware CIL

Table 2. **Speed marks by BDTI Labs**

Chip	BDTI2000 speed mark	Chip	BDTI2000 speed mark
Agere DSP 164xx 285 Mhz	1360	ADI ADSP-219x 160 Mhz	410
ADI ADSP-BF-5xx (Blackfin) 750 Mhz	4190	ADI ADSP TS201S (TigerSHARC) 600 Mhz	6400
FreeScale DSP563xx 275 Mhz	820	Freescale MSC81xx (SC140) 300Mhz	3370
Intel PXA255/PXA266 (XScale) 400Mhz	930	Renesas SH772x (SH3-DSP) 200 Mhz	490
Intel PXA27x (XScale + Wireless MMX) 624MHz	2140	TI TMS320C54x 160 Mhz	500
TI TMS320C55x 300 MHz	1460	TI TMS320C62x 300 Mhz	1920
TI TMS320C64x 1GHz	9130		

processor implementation. By the way, the limited parallelism of classic DSPs allows to execute a computation and a data load instructions in parallel, so even quite a simple CIL instruction decoder can combine computation on the top of the stack with data loading for the next instruction.

Additionally to the BDTI benchmarks, the comparison of different procedures for copying the memory region (100 words) for different processors is presented in table 3. The code size and the execution time characteristics are provided. The memory copying function is selected, because it utilizes memory and address computations highly without high ALU utilization and allows to estimate imperfections in the stack processor paradigm.

We assume that there are no penalties for the memory access. The summary information on the code is presented in the table 4.

So, the code size is growing from the DSPs up to the stack processors. Of course, if the stack processor has a special instruction for memory block copying (like *CMOVE*), the code will be much smaller, but the comparison will have no sense. But a real stack processor has a few resources for handling data which can be processed in parallel. In extreme case a stack processor can only operate with the top of the stack. E.g., comparing *arithmetic* stack paradigms, which are used inside CIL [5] and FORTH-94 [2] standard (both are based on the abstract stack machine), we see clearly, that stack implementations are quite different (table 5).

The stack implementation for the FORTH-94 virtual machine requires that all stack elements should be visible and able to be fetched, so all stack elements are completely addressable. This is helpful in the situations, when there are subexpressions, which intermediate results are used more than once. In other cases a recalculation may be used for saving some data into a temporary variable or for future calculations. If only the top the of stack may be used, the possibilities for the temporary storing of local variables and intermediate results are very limited. As FORTH-94 paradigm

Table 3. **Comparison of different procedures for copying a memory region**

DSP	RISC	Stack with address reg (ADR1 ADR2) ON STACK	Stack without address register (ADR1 ADR2) on stack

R1 = DM(I0, M0);
DO LCNTR = 99, M2
R1 = DM(I0, M0) ||
DM(I4, M4) = R1;
M2:
DM(I4, M4) = R1;
	LD R0,100
	M1:
	LD R1,[A0++]
	ST [A1++],R1
	SUB R0,R0,1
	BRNZ M1
		A!
		LIT 100
		LIT 0
		(DO)
		DUP @ A++!
		WORD+
		(LOOP) DROP
			LIT 100
			LIT 0
			(DO)
			2DUP @ !
			LIT WORD
			DUP D+
			(LOOP) 2DROP

Table 4. **Code sizes for copying a memory region**

	DSP	RISC	Stack with address register	Stack without address register
Internal cycle size, bytes	4 (8 if without dual memory access)	16	6	9
Time, cycles	101 (200 if without dual memory access)	400	600	800

Table 5. **Stack-based instructions for CIL and Forth paradigm**

CIL instructions for stack operations	Forth-94 instructions for stack operations
DROP – removes top of stack element DUP – duplicates top of stack	DROP – removes top of stack DUP – duplicates top of stack 2DUP – duplicates two topmost elements of stack ROLL <n> - moves n-th element to the top of stack, other elements are moved deeper into stack up to n-th cell

requires thar all stack elements should be addressable, the only possible implementation of such a stack is the random access memory (RAM) with hardware implemented the top-of-stack pointer. Such RAM implementation is very inefficient, because it uses at least one write or one read operation from the RAM per an instruction. On the contrary, the stack implementation with access only to the top-of-stack register does not require RAM-like implementation of the stack, but the overflow or the underflow conditions require large memory block transitions (from the stack to the memory and from the memory to the stack). The stack with a limited depth where only 4 or 8 top elements of the stack are accessible, and the other elements are not is implemented in the Novix processors [1]. But in case of the CIL processor the statement that it involves only the top elements to be visible is incorrect in general. A called method uses a set of parameters and local variables, also *this* pointer is employed. The parameters are loaded into the stack by the callee method; the local variables also must be located in some stack. So, the current arithmetical stack requires only two top elements to be visible, but all arguments and local variables are located in the stack which must be organized as RAM. Here there is a field for optimizations in a JIT compiler – all method arithmetic may be optimized for utilizing the register file only.

In the chapter the most interesting implementations of embedded processors used in industry were discussed. The hardware CIL processor will not be something unusual and so different from any existing processor. In the next chapter the architecture of the hardware CIL processor will be discussed.

2. The architecture of the hardware CIL processor

Due to the list of the main properties of the CIL processor, it is necessary for the processor: 1) to consume low power from energy supply; 2) to handle efficiently DSP tasks; 3) to execute directly or through moderate-sized decoder the hardware CIL code.

The basic idea which underlies the CIL processor implementation is a direct execution of the DSP code and of the hardware CIL code, so that the target is a mix of a real DSP processor and a hardware CIL decoder and control unit.

2.1. The high-level model of the CIL processor. The essence of the trick is that the CIL processor consists of two chips. So the processor is able to execute library code used for media operations efficiently. Such processor implementation is very useful for the software implementation of different communication protocols in smart phones, e.g. convolution, adaptive filtration, Viterbi decoding, software radio procedures. Communication protocols, wireless link protocols, audio codecs, video codecs must be implemented as libraries in the DSP native code, that allows to have a very efficient and low energy consuming library code.

On the other hand, the CIL instruction decoder is intended for direct execution of downloaded (from Internet) business application code. Usually the applications, oriented for visual processing (database front-ends, Internet forms, accounting information), have no need for complex code optimizations, because most of work inside the front-end applications is done inside visual forms and components, but all the database processing with large data amounts is carried on external servers. The media information is processed with the use of the DSP-optimized multimedia libraries.

From the point of view of the processor software model, the programmer has two instruction sets in one processor: a DSP set and a CIL set. This scheme is similar to ARM/Thumb instruction sets available in ARM cores. Depending on the desired execution mode, the processor decoder can be switched between the DSP and the CIL instruction sets decoding.

From the point of view of the hardware, there is a DSP processor, which has all classical DSP attributes as two address and data buses, a dual address generation unit and a high-performance MAC unit. The processor has two instruction decoders, they allow to decode the native DSP instructions and the CIL instructions. The execution of the CIL code is supported by internal control buses (for the access to the meta-information) and additional direct-mapping or associative caches, intended for caching most often used object and class information.

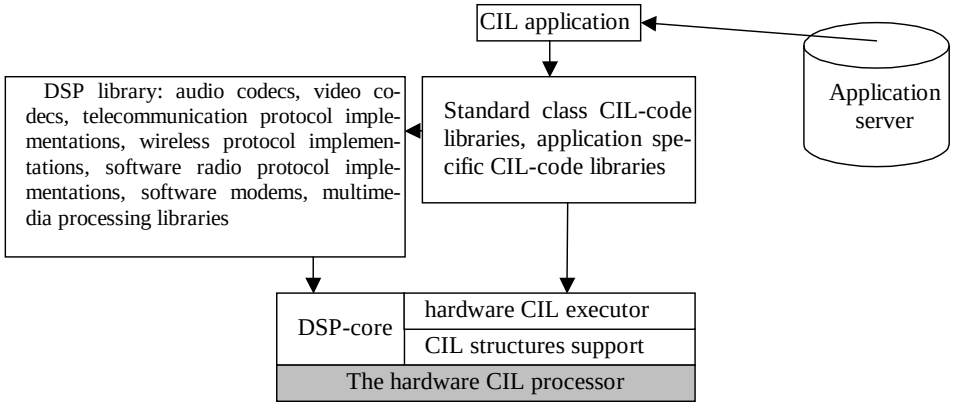

Fig. 4. High-level structure of the CIL processor implementation

The high-level hardware structure of the CIL processor is shown in fig. 5.

The CIL processor has the structure, which is an extended version of the classical DSP one. The processor has four external buses. The main data bus (X-bus) and the secondary data bus (Y-bus) are used for arithmetical data transfers from/to caches and external memory. The X and the Y space address bus are used for address generation for the memory access. The main arithmetic unit is used for general computations, during a clock cycle two data words may be read or written to/from memory spaces. The X and the Y bus address generation units form addresses for the memory accesses, the instruction and data cache unit stores data (at least 32 code words for instructions) and the system control unit for interrupt handling. The native DSP instruction set decoder decodes native DSP instructions, the additional CIL instruction decoder maps a CIL instruction into DSP core control signals. The CIL decoder operates jointly with caches for the CIL meta-information, which are intended for speeding up the common object model operations for the CIL. An example for such operation is the translation of the object field handler for a particular class into an offset from the start of the object location into the memory and the appropriate access operation for current field type. If compared to any Java processor, CIL involves many extra difficulties, because the Java code is "ready" for direct execution without any additional translation operations. Here the CIL meta-information caches help to shadow time taken for the "handler-to-address" translation in the pipeline, so that the memory access operations using object/field/method/etc handlers are performed in short time in case of cache hits. The precise size of different caches and the method for the data mapping in each cache will be determined after profiling the target CIL applications set. For any software application these caches are invisible.

2.2. The ALU architecture for the processor. The most interesting unit is the main arithmetic unit, which is operating in two modes. The first mode is the DSP mode. The ALU performs "register-to-register" arithmetic

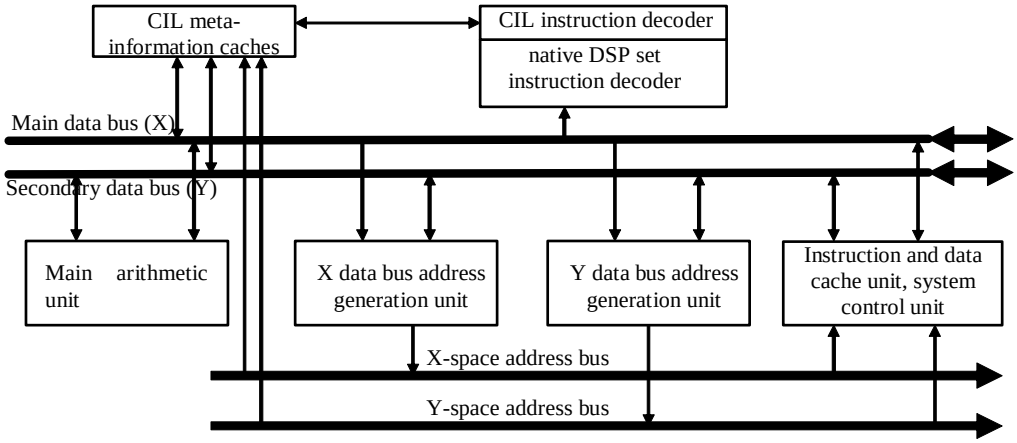

Fig 5. High-level structure scheme of the CIL processor

instructions and receives and sends data for the memory transfers. The second mode is the stack computation mode; where several internal DSP registers are used as the topmost stack registers (for the CIL model only two registers are needed). The other stack registers are implemented as a shadow stack for one of the DSP-mode registers, which is used in the stack mode as the "under-the-top" register.

The arithmetic unit architecture must be discussed from the point of view of the low energy consumption. As there is no way to decrease the number of arithmetic functional units (there is a minimal set of necessary functionality), the most appropriate way to decrease the energy consumption is simplification of the connection network between the register file and the functional units. Another important issue is extension of the available internal pipeline parallelism.

The examples of such low-power DSP design are Agere (former Lucent) cores DSP1600 and DSP16000 [14]. The main idea of the design is high specialization of the functional units and the register file. Due to the structure of the most often used DSP algorithms, there is no need in full interconnection scheme between all register files and the functional units. For example, the register file can be divided at least into two parts: the single length register file (integer or fixed point) and the double-length register file (fixed-point) or accumulators. In case of the most often used DSP operations such as matrix multiplication or convolution $y = \sum_i x_i h_i$ ($y = \sum_i x_i h_{N-i}$), the MAC unit inputs are always connected to the single precision register file, and the intermediate result (double-precision) can be connected to input of the accumulator register file. The addition of the intermediate results may be performed only using extended precision adder connected to the accumulator register file only.

The structure of the DSP specialized ALU unit with reduced internal connection is presented in fig. 6.

Here the ALU is divided into two parts. The first (upper) part consists of two multipliers, which can multiply single precision words in one cycle. All data loaded into the core is routed using the X-space and the Y-space vector input registers. These registers are vector (twice wide as the integer) so a double precision number can be loaded during one data transfer and further can be processed as two single precision numbers. The scheme allows performing two multiplications simultaneously. There are some practical limitations on operands of the multiplier, e.g. it is impossible to perform identical computations on both multipliers. The cross-bar commutation unit performs result shifting before the multiplication. The results of multiplications are handled in the temporary product registers. All data from the X space and the Y space registers can be shifted before the multiplication and after passing to the product register. The data from the X (or the Y if necessary) register can pass the scheme by if there is no need in the multiplication. Also the temporary result registers can be uploaded to the data memory or the program memory buses.

The second part of the ALU consists of several functional units and the register file. There are strong restrictions on the commutation scheme, but usually these restrictions do not affect performance computations. E.g. one of the operands of the ALU can be only an accumulator register. Also operands of the quadruple adder are restricted to input registers, both the temporary product register and two accumulator registers from different halves of the register file. The quadruple adder is intended for complex number computations, in particular for the complex number multiplication. The special functional unit inputs are even more restricted in operands – usually only the registers from different halves of the register file may be used as operands. Three outputs of the functional units are multiplexed to write ports of the first part (even registers) or the second part (odd registers) of the register file. Also the outputs of the register file are commutated from even or odd registers for reducing the size of the multiplexer unit. Also the output of the register file can be commutated to the memory write port.

The accumulator registers A0, A1 are used as shadow stack registers. The hardware stack with 8 or 16 registers is connected to the A1 register, so that shadow arguments of each stack instruction are registers A0 and A1. But, the hardware stack is not necessary for stack-based computations, because under CIL execu-

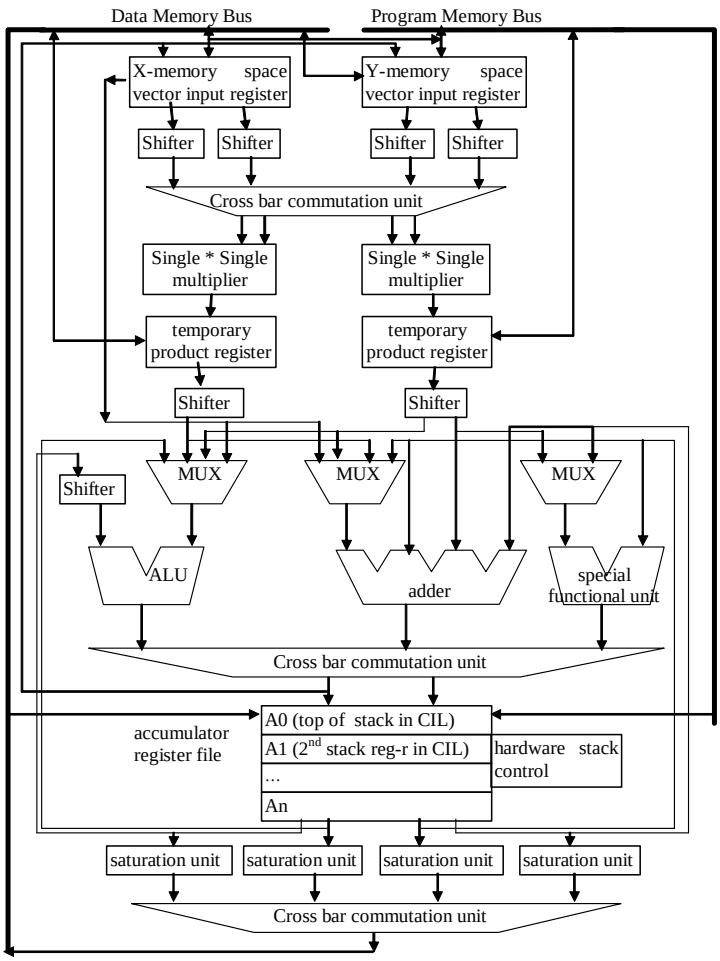

Fig. 6. The structure scheme of the ALU unit

tion one of internal buses (usually for the program memory) is free of memory access near 75% of the time. Most of CIL commands take 1 byte of the memory and during one access to the program memory up to four CIL instructions can be loaded into prefetch buffer. For overflow and underflow cases the hardware stack can be connected to the memory bus for block memory transfers, and so the program memory data bus can be used. The number of upper stack registers, mapped on the accumulators, can not exceed 2. The explanation of this fact is simple: under the CIL stack model all computations use no more than 2 registers and affect only the top of stack register. Because three ALUs use two write ports for the register file, all write ports may be utilized in the CIL execution mode. E.g. a common arithmetical expression like *(a+b)*c* usually evaluates as *LD_A LD_B + LD_C **. Further, the instruction "+" and the instruction "LD_C" can be combined and executed in one cycle, because the data memory bus is always free during CIL computational operations. The result of the arithmetic operation ("+") will be placed into A1 register, the load instruction result - into A0 register.

So in the specified ALU architecture the size of all multiplexers is reduced because the number of input and output ports is reduced. The multiplexer is divided when possible into smaller ones, and even removed if there is no need in extra commutation abilities.

In the chapter a specific option for the CIL execution has not been discussed yet. Several CIL operation such as addition, subtraction, multiplication, division are typed – for different types of operations the operation are coded identically – so the CIL instruction "+" may represent an integer of floating point operation. E.g. the accumulator registers in the register file usually have extra bits ("guard bits") for preventing accumulator overflow during convolution fixed point operations. For supporting the type model, extra tag bits are added to each register to represent the internal type (integer, float, address type). During the instruction decoding, at the last pipeline stage

the operand tags will be analyzed and proper functional unit will be activated. Tag bits require at least 5 bits per each register and have quite simple logic for type analysis and producing a type of result. There is no need in handling additional type index in the tag, because the type of an object may be determined in run-time using the RTTI (Real Tile Type Information) mechanism.

2.3. Address generation units for the processors. The other important units are address generation units, which drive data fetch engine for two memory spaces. The address generation units must support most address generation schemes, which are used in the DSP applications. These addressing schemes are: 1) register + immediate offset; 2) register with automatic programmable increment or decrement, pre- or post; 3) circular window (for the convolution) with programmable automatic decrement or increment, pre- or post; 4) bit reversing addressing for Fourier transformation. The structural scheme of the address generation unit is quite common and represented in fig. 7.

The address generation unit consists of an address adder, several multiplexers and address register files. The width of the adder equals to the width of the address register. There are several sources for the bus address. Any pointer register or pointer to beginning of circular buffer may be added to one of the index registers or the immediate value (from the instruction or byte/word increment addressing mode) and stored in the pointer register. Also, the pointer register or the pointer to circular buffer may be multiplexed in the address bus. So, the possible addressing modes are: 1) *pointer+index*;2) *circular start + index*; 3) *pointer+offset*; 4) *circular+offset*; the offset may be zero. Any pointer register may be incremented by the offset or index register content. The comparator is used for comparing the stored address with the address of the end of the circular buffer and clearing the stored address with the beginning of the circular buffer. The structure of the address generation unit is quite simple, and the sizes of different register files may be simply adjusted for a particular design. On the right side of figure 6 a simplified structure is presented. If there is no need for one of the memory spaces in a circular buffer addressing (usually a circular buffer for only one memory space is necessary) the structure is simplified. Here the only the addressing modes *pointer+index* or *pointer+offset* with increment by the index or the offset are possible, but these available modes cover all user needs. Note, that for negative address increments the address of the start of the circular buffer must be above the address of the end of the circular buffer, in comparison to other processors, where in case of negative increments, the addresses of the start and the end of the circular buffer must not be swapped.

2.4. Decoder unit of the processor. Such simple principles form the kernel of the DSP processor. Further targeting is provided by engineering practice: we mean implementation of particular functional units, integration of necessary memory controllers on the chip, implementation of interrupt controller, watchdog timer and interval counters, input/output ports and interfaces for different external devices. The implementation of such devices is

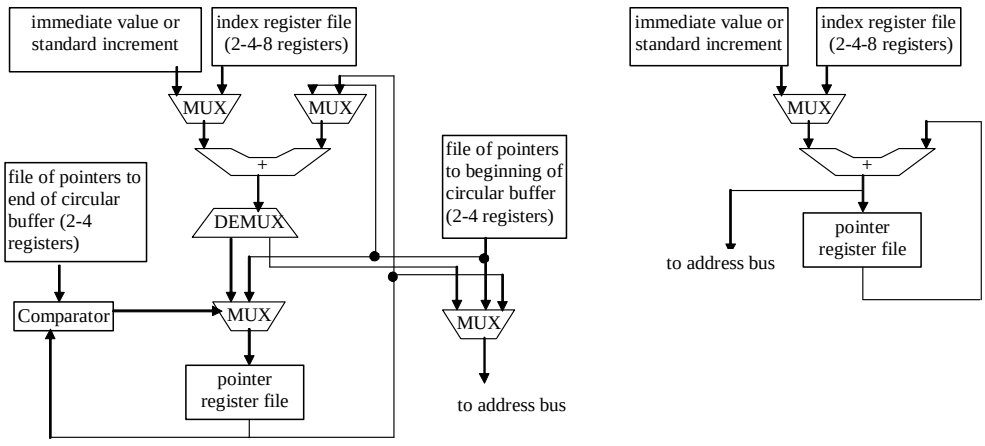

Fig. 7. Structural schemes of the common address generation units

not principal and is not covered in the paper, as most devices can be added into VHDL/Verilog design without many efforts, and do not influence computational power of the projected processor. The ALU and the address generation units are the key features for high computational power, high memory bandwidth and define peak performance of the processor.

The last complicated thing in the CIL processor is the decoding unit, which must operate in the two modes: the genuine DSP and the CIL interpretation mode. The DSP interpretation mode is quite straightforward and may be realized using commonly used tri-staged pipeline for the instruction decoding in simple DSPs. The described ALU unit is internally pipelined for the multiplication and common ALU functional units, so during one DSP instruction the internal scalar parallelism of the pipelined ALU is utilized. Additionally, a DSP instruction in common cases drives both address generation units for forming necessary addressing modes. The full list of DSP instruction will be generated after considering all design issues.

The CIL instruction set has only one way to implement, so the task is to design a decoder, which will transform (at duplicated pipeline stages) a CIL instruction into a set of control signals for the ALU and the address generation units. Complex commands will generate an exception, and such an instruction can be implemented internally in fast ROM as native DSP code. Meta-information caches will be used for speeding up meta-information access.

A special instruction will be provided for changing current instruction set from the CIL to the DSP and backwards. The cached registers may be mapped directly to A0 and A1 accumulators (i.e. without a special switching circuit), the stack pointer will be mapped to one of the pointer registers in the address generation unit for the program memory bus.

2.5. Meta-information cache memory. No doubt searching inside the meta-information tables takes too much time even if access is provided in a sorted out column with the help of logarithmic search algorithms. The only solution is to provide several distinct small cache memories for handling the most often used data in scratchpad memories. The scratchpad memories must be tuned for caching only some fragments of the main memory, organized as a table with rows of constant length with the key value at the beginning of each row. Selected rows of the table are cached in the scratchpad memory (fig. 8).

In fig. 8 the principle of cache organization is presented. As each meta-information object is addressed in a unique way by the index, the index is used as a tag in the cache memory. The scheme is useful e.g. for parameters, when a small consequent region of the data table is used because a multi-way cache is useless, in such case the one way cache may be built with the help of a small SRAM block and a comparator (as shown in fig. 8). If cache mapping is not direct, the number of ways in the current cache may be increased, up to 2-4 or even 8. Inside the cache memory the least significant bits of an item number may be used as the tag. The cache way is constructed on a small SRAM block with a comparator, so the cache way implementation is quite compact and allows implementation of dozens of caches on the die. For example, a preliminary solution for metadata cache organization is represented in table 6.

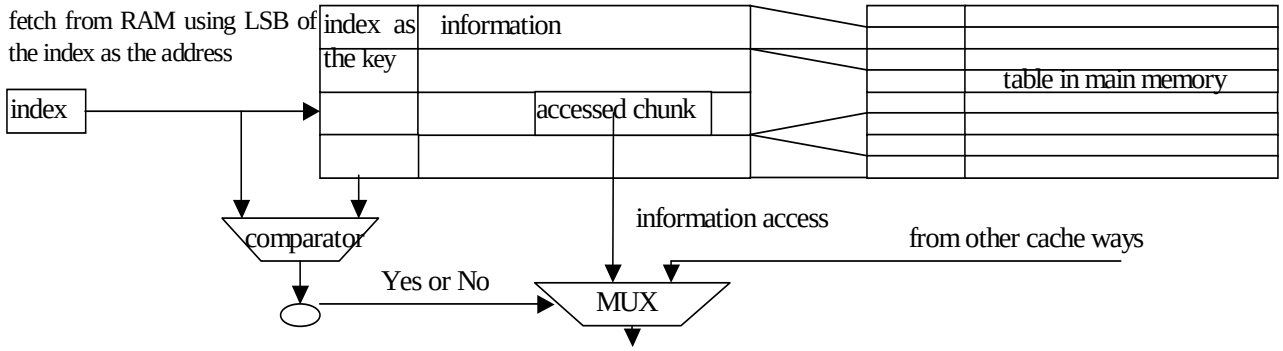

Fig. 8. The meta-information cache memory principle

Table 6. **Cache implementation parameters**

Direct cache mapping	2- or 4- way cache mappings
ManifestResource, ExportedType, Param, MethodDef, InterfaceImpl	DeclSecurity, Constant, Field, Property, MemberRef, MethodImpl, TypeDef, TypeRef, EventMap, Event

Note, that several metadata tables must not be implemented as the cache memories, for example such as File and Assembly tables. These tables are used only by system software (for example, the software for downloaded CIL code verification) and very rarely. Also, most metadata tables are transformed during loading for the most appropriate format for future data utilization inside the hardware. For example, Field metadata is not very useful for direct utilization, because the actual information is contained in the five meta-information tables related to the "Field" table. To fill the processor pipeline in time the value of the field offset and the field access type (representing actual type and length for a loading/storing operation) must be located in the cache. The instructions for field access must start search for the field descriptor in the field descriptor cache at the decoding stage and load into the decoder the field offset and the appropriate access instruction (read or write) in one or two cycles. At the next pipeline stage (or at the current stage) the base object address, which is directly transformed from object descriptor in parallel to the field decoding, is added to the field offset, and at the next pipeline stage a value may be loaded from or stored to the object. So that, the object access instruction may be implemented during maximum three or four processor cycles. Of course, in case of cache misses, time of the instruction performing will be increased.

The minimal set of caches metadata tables can be determined considering all CIL instructions which use metadata tables in some way. The following metadata tokens are used: 1) *Method*, represents entries from *MethodDef* or *MemberRef* tables; 2) *Field*, represents entries from *FieldDef* or *MemberRef*; 3) *Type*, represents entries from *TypeDef*, *TypeRef* or *TypeSpec* tables; 4) *Signature* (entry from *StandAloneSig* table); 5) *String*; 6) *Constant*. The *this* pointer also must be cached for speeding up *ldarg.0* instruction and access to the RTTI information of the current object.

We assume that metadata tables are saved for further processing. But most tables are not suitable for the direct use in the cache. Some summary of the tables will be used as "cached metadata tables". The content of the cached tables allows to speed up most often prepared operations like field access, method invocation, determining RTTI address. The preliminary layout for the cached tables is represented in table 7.

All rows in the cache tables starts with unique index which is used for the row location in the cache. In most cases run time type checking for method invocation and field access is superfluous, so technically the method

Table 7. **Layout for the cached meta-information tables**

Table	Fields	Comment
1. Method	index	key value, 4 bytes
	implementation address (VMT offset)	4 bytes
	number of arguments	2 bytes
	number of locals	2 bytes
	returned value type	1 byte
2. Field 6. Constant	index	key value, 4 bytes
	offset from the start of object	3 bytes
	internal field type	1 byte
3. Type	Index	key value, 4 bytes
	address of definition	4 bytes
	Length	3 bytes
	Tag	1 byte
4. String 5. MemberRef	Index	key value, 4 bytes
	String address	4 bytes

invocation and the field access uses the same principles as in common RISC processor. The "internal field type" is a tag for the CIL processor type.

The basic operation for the cache access may be implemented in the following way. The arguments for such instruction are: the cache number for a metadata table (programmed inside the instruction), the index for the cache access (for searching inside the cache), the offset for the necessary information in the cache line (programmed into the instruction too) and some optional information (like the width of data). So, the cache access instructions must be performed in two steps: 1) using the cache number and a row index it tries to locate row in the cache, in case of cache miss the missed row must be loaded from the main memory, a located row will be loaded into the output buffer; 2) the instruction may read the required information using the offset from the output buffer. At high frequencies these operations must be performed in two cycles.

The Java object model is quite simple, because all fields and object references are expressed in bytes and there is a direct instruction for data loading and storing instructions. The .NET metadata model requires additional runtime checking and quite high computational overhead. The special caches and fast access decrease time waste, but in each metadata-related instruction several cycles will be wasted by filling the processor pipeline by additional atomic instructions for the cache access.

Of course additional cache memories can only increase the energy consumption of the processor, but here there is a tradeoff between efficient implementation and the energy consumption. The energy consumption of caches is spread over the chip, and only one or two caches are used simultaneously, so the energy consumption is not significant in comparison with the ALU and the memory controllers' energy consumption. But the metadata caches can speed up most of instructions related to object model, such as *jmp, call, callvirt, ldftn, ldvirtfnt, ldfld/ldflda, stfld, ldsfld/ldsflda, stsfld, cpobj, ldobj, stobj, box/unbox, initobj, sizeof, ldstr*. (Other instructions implement a complex functionality and it is better to implement them in the DSP microcode using the hardware exception mechanism.

3. Software support for the CIL processor

The hardware CIL processor is only part of development works. The final device is a sum of a DSP-enhanced CIL processor, an exception microcode, a firmware, reduced system libraries and possible end-user application.

The hierarchy of the system software must be something like following:

Exception microcode. If a processor instruction can not be implemented in hardware (e.g. floating point operation in a fixed-point processor), an exception will be raised. The exception involves execution of a sequence of microcodes, stored into a fast ROM inside the processor. This method also addresses all cases, there class hierarchy checks are involved. The exception microcode is completely supported in the processor instructions, including the DSP and the CIL sets. In reality, the exception microcode is part of the processor. Only DSP instructions may be used in the microcode, because the CIL set has a higher level of abstraction than the DSP set and needs extra support in hardware. The DSP set includes all instructions that drive the CIL code execution.

Class library, as covered in the ECMA-335 standard. All standard classes must be implemented as the system library for providing basic functionality for external applications. There is a tradeoff between many additional class implementations on the chip and downloading the class libraries via wireless or cable channel from neighbor servers. The basic class library must be localized, e.g. RS-232 channel or display may serve as the standard input and the standard output, and the file system may be based on simple file systems such as ROM file system or RAM file system (used in Unix world) or FAT16/32 file system, depending on a particular application.

Supporting system libraries, intended for loading and executing remote applications and supporting the basic network protocols like IP, UDP and TCP, optionally wireless communications may be provided. These libraries may include a request broker, a linker, a loader, a code verification system, a meta-information transformation library for converting it into cache-ready format. Because of high complexity of network protocols networking libraries creation is not part of the project.

The supporting libraries depend on available system devices for the input and output. The minimal projected set of system devices are: 1) the output device, like LCD text panel, LCD graphics panel or external SVGA monitor with library support; 2) the input device like joystick, mouse or keyboard with library support; 3) an external memory storage with library support of a simple file system (e.g. FAT16); 4) a network adapter with device driver support; 5) a RS-232 channel with library support.

Multimedia libraries, intended for processing multimedia content. These libraries may be implemented in DSP instructions and can include multi-format video and audio playback code. Creation of these libraries either is not a part of the project.

User application. At the top of the software hierarchy there is a user application, stored in ROM/Flash memory or downloaded from the Internet. In real life, the application may be implemented in the DSP or the CIL code, utilizing all the libraries available on the chip and from the network channels.

Just in time compiler for the CIL code. The CIL processor has dual architecture - it incorporates the CIL and the DSP instruction sets support. But also the current DSP architecture has good mapping abilities for JIT compiler. Like in hardware implementation, the arithmetic stack will be mapped on the accumulator register file, so performance of computational instructions generated by a JIT-compiler is nearly equal to the projected CIL processor. But real speedup will be achieved in the JIT-compiler in object-model related optimizations. E.g. for static object access most of common type-checking operations and field descriptor translations are superfluous. Usually the JIT-compiler utilizes well-known techniques of graph rewriting (and code generation) [15]. For processors with large register files and scalar parallelism additional optimizations for improving register allocation are used. The DSP processor has quite a simple system architecture, so direct techniques of consequent CIL code transformation into the DSP code step by step, instruction by instruction are applicable. Further, the JIT can outperform the hardware implementation in case of optimizing stack accesses, required by executed methods and proper optimization of the object information. But, possible JIT optimizations for CIL model are subject for a separate paper.

Also there is an important question about available development toolkit. It is possible to design the DSP chip which will have a well-known instruction set, but none of the existing compilers can support *effectively* joint DSP+CIL model. There are GNU CC [16] based compilers, with CIL and some DSP support, but the joint paradigm has not been implemented yet. Independent of existing software, the basic development tools such as assembler can be made for the CIL processor. Also, using well known abilities of the GNU CC package, the GCC compiler can be retargeted to a small subset of DSP and CIL instructions, so enabling the existing C code to be used on the CIL processor.

Conclusions

In the paper the basic concepts for the CIL processor implementation are discussed. We have supposed the processor target market – the market of mobile devices connected to the Internet, where the .NET technology can compete with other integrating technologies. The requirements for the CIL processor are suggested: 1) good performance in multimedia applications; 2) low power consumption. An effective solution for the CIL processor architecture has been suggested: the CIL processor is based on the classic DSP kernel with the additional CIL instruction set decoder. The DSP kernel was selected as kernel for the CIL processor because of high speed in multimedia tasks, low energy consumption and quite simple in implementation. Such kernel is well suited for mobile applications, multimedia applications, DSP applications and network software. In the paper the DPS kernel implementation was discussed and necessary explanations are given. The structural schemes for the units which affects the chip performance (the ALU the address generation units, the meta-information caches) are presented in the paper. Finally, all levels of software support (from the ROM exception code up to user applications) are discussed in the paper. The paper gives the good basics for future technical development and implementation practices, also the paper describes technical implementation issues and gives a good overview of the project outline.

1. *Koopman Ph. J.*, Stack Computers: the new wave. Chichester: Ellis Horwood Ltd., 1989.
2. *American* National Standards Institute, Inc. Technical Committee X3J14. American National Standard for Information Systems - Programming Languages // Forth. ANSI X3.215-1994. - 250 p.
3. *Patriot* Scientific Ignite Core. - http://www.ptsc.com/ Download/download/IGNITE_Processor_Reference_Manual.pdf
4. *ARM* Jazelle technology. Overview. – http://www.arm.com/products/ solutions/Jazelle.html
5. *Standard ECMA-335*. Common Language Infrastructure (CLI). - http://www.ecma-international.org/ publications/files/ecma-st/ECMA-335.pdf.
6. *Register* Organization for Media Processing. // S. Rixner, W. Dally, B. Khailany, P. Mattson, U. Kapasi, J. Owens / In Proc. of 6th annual Intern. Symp. on High-Performance Computer Architecture (HPCA), 2000. - P. 375 – 386.
7. *Swamp:* A fast processor for SmallTalk-80. / D. Lewis, D. Galloway, R. Francis, B. Thomson // ACM OOPSLA'86 Proc., Sept. 1986. - P. 131 – 139.
8. *Nojiri T., Kawasaki S., Sakoda K.* Microprogrammable processor for object-oriented architecture // Proc. of IEEE ACCA, 13th Intern. symp. on Computer Architecture, 1986. - P. 74 – 81.
9. *The BDTIMark2000™:* A Summary Measure of DSP speed. Berkeley Design Technology Inc., Febr. 2003. - http://www.bdti.com/articles/bdtimark2000.pdf.
10. *Benchmarking* Processor for DSP applications. Berkeley Design Technology Inc., 2004. - http://www.bdti.com/articles/20040219_TIDC_Benchmarking.pdf.
11. R. Kolagotla, J. Fridman, B. Aldrich, M. Hoffman, W. Anderson, M. Allen, D. Witt, R. Dunton, L. Booth / High performance dual MAC DSP architecture. // IEEE Signal Processing Magazine, July 2002. - P. 42 – 53.
12. Trends and performance in Processors for Digital Signal Processing. Berkeley Design Technology Inc, 2003. - http://www.bdti.com/articles/ 20030522_Trends_and_Performance.pdf
13. BDTIMark2000™/BDTISimMark2000™ Scores for Fixed Point Packaged Processors. Berkeley Design Technology Inc, July 2004. http://www.bdti.com/ articles/chip_fixed_scores.pdf
14. *Agere* Systems. DSP16000 core reference manual. - http://www.agere.com/enterprise_metro_access/docs/MN02027.pdf
15. . *Overview* of the IBM Java Just-in-Time Compiler / T. Suganuma, T. Ogasavara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, T. Nakatani // IBM Systems Journal. 2000. - **39**, No 1, p. 175 – 193. - http://www.research.ibm.com/journal/sj/391/suganuma.pdf.
16. *GNU* Compiler Collection. - http://gcc.gnu.org

Date received 05.07.05

Об авторах

Алексей Владимирович Чепыженко
старший инженер

Место работы автора:
ЗАО А/О Интел, Санкт-Петербург,
E-mail: alex_drom@yahoo.com.

Дмитрий Васильевич Рагозин:
канд. техн. наук.,
старший научный сотрудник

Место работы автора:
ЗАО А/О Интел, Нижний Новгород,
E-mail: ragozin@wl.unn.ru.

Алексей Львович Умнов:
канд. физ.-мат. наук,
доцент кафедры электродинамики

Место работы автора:
Нижегородского государственного университета им. Лобачевского,
E-mail: umnov@wl.unn.ru.