Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 56976, 10 pages
doi:10.1155/2007/56976

*Research Article*

# Observations on Power-Efficiency Trends in Mobile Communication Devices

**Olli Silven[1] and Kari Jyrkkä[2]**

*[1] Department of Electrical and Information Engineering, University of Oulu, P.O. Box 4500, 90014 Linnanmaa, Finland*
*[2] Technology Platforms, Nokia Corporation, Elektroniikkatie 3, 90570 Oulu, Finland*

Received 3 July 2006; Revised 19 December 2006; Accepted 11 January 2007

Recommended by Jarmo Henrik Takala

Computing solutions used in mobile communications equipment are similar to those in personal and mainframe computers. The key differences between the implementations at chip level are the low leakage silicon technology and lower clock frequency used in mobile devices. The hardware and software architectures, including the operating system principles, are strikingly similar, although the mobile computing systems tend to rely more on hardware accelerators. As the performance expectations of mobile devices are increasing towards the personal computer level and beyond, power efficiency is becoming a major bottleneck. So far, the improvements of the silicon processes in mobile phones have been exploited by software designers to increase functionality and to cut development time, while usage times, and energy efficiency, have been kept at levels that satisfy the customers. Here we explain some of the observed developments and consider means of improving energy efficiency. We show that both processor and software architectures have a big impact on power consumption. Properly targeted research is needed to find the means to explicitly optimize system designs for energy efficiency, rather than maximize the nominal throughputs of the processor cores used.

## 1. INTRODUCTION

During the brief history of GSM mobile phones, the line widths of silicon technologies used for their implementation have decreased from 0.8 $\mu$m in the mid 1990s to around 0.13 $\mu$m in the early 21st century. In a typical phone, a basic voice call is fully executed in the baseband signal processing part, making it a very interesting reference point for comparisons as the application has not changed over the years, not even in the voice call user interface. Nokia gives the "talk-time" and "stand-by time" for its phones in the product specifications, measured according to [1] or an earlier similar convention. This enables us to track the impacts of technological changes over time.

Table 1 documents the changes in the worst case talk-times of high volume mobile phones released by Nokia between 1995 and 2003 [2], while Table 2 presents approximate characteristics of CMOS processes that have made great strides during the same period [3–5]. We make an assumption that the power consumption share of the RF power amplifier was around 50% in 1995. As the energy efficiency of the silicon process has improved substantially from 1995 to 2003, the last phone in our table should have achieved around an 8-hour talk-time with no RF energy efficiency improvements since 1995.

During the same period (1995–2003) the gate counts of the DSP processor cores have increased significantly, but their specified power consumptions have dropped by a factor of 10 [4] from 1 mW/MIPS to 0.1 mW/MIPS. The physical sizes of the DSP cores have not essentially changed. Obviously, processor developments cannot explain why the energy efficiency of voice calls has not improved. On the microcontroller side, the energy efficiency of ARM7TMDI, for example, has improved more than 30-fold between 0.35 and 0.13 $\mu$m CMOS processes [5].

In order to offer explanations, we need to briefly analyze the underlying implementations. Figure 1 depicts streamlined block diagrams of baseband processing solutions of three product generations of GSM mobile phones. The DSP processor runs radio modem layer 1 [6] and the audio codec, whereas the microcontroller (MCU) processes layers 2 and 3 of the radio functionality and takes care of the user interface.

Table 1: Talk times of three mobile phones from the same manufacturer.

| Year | Phone model | Talk time | Stand by time | Battery capacity |
|---|---|---|---|---|
| 1995 | 2110 | 2 h 40 min | 30 h | 550 mAh |
| 1998 | 6110 | 3 h | 270 h | 900 mAh |
| 2003 | 6600 | 3 h | 240 h | 850 mAh |

Table 2: Past and projected CMOS processes development.

| Design rule ( nm) | Supply voltage (V) | Approximate normalized power∗delay/gate |
|---|---|---|
| 800 (1995) | 5.0 | 45 |
| 500 (1998) | 3.3 | 15 |
| 130 (2003) | 1.5 | 1 |
| 60 (2010) | 1 | 0.35 |

Table 3: An approximate power budget for a multimedia capable mobile phone in 384 kbit/s video streaming mode.

| System component | Energy consumption (mW) |
|---|---|
| RF receiver and cellular modem | 1200 |
| Application processors and memories | 600 |
| User interface (audio, display, keyboard; with backlights) | 1000 |
| Mass memories | 200 |
| Total | 3000 |

During voice calls, both the DSP and MCU are therefore active, while the UI introduces an almost insignificant portion of the load.

According to [7] the baseband signal processing ranks second in power consumption after RF during a voice call, and has a significant impact on energy efficiency. The baseband signal processing implementation of 1995 was based on the loop-type periodically scheduled software architecture of Figure 2 that has almost no overhead. This solution was originally dictated by the performance limitations of the processor used. Hardware accelerators were used without interrupts by relying on their deterministic latencies; this was an inherently efficient and predictable approach. On the other hand, highly skilled programmers, who understood the hardware in detail, were needed. This approach had to be abandoned after the complexity of DSP software grew due to the need to support an increasing number of features and options and the developer population became larger.

In 1998, the DSP and the microcontroller taking care of the user interface were integrated on to the same chip, and the DSP processors had become faster, eliminating some hardware accelerators [8]. Speech quality was enhanced at the cost of some additional processing on the DSP, while middleware was introduced on the microcontroller side.

The implementation of 2003 employs a preemptive operating system in the microcontroller. Basic voice call processing is still on a single DSP processor that now has a multilevel memory system. In addition to the improved voice call functionality, lots of other features are supported, including enhanced data rate for GSM evolution (EDGE), and the number of hardware accelerators increased due to higher data rates. The accelerators were synchronized with DSP tasks via interrupts. The software architecture used is ideal for large development teams, but the new functionalities, although idling during voice calls, cause some energy overhead.

The need for better software development processes has increased with the growth in the number of features in the phones. Consequently, the developers have endeavoured to preserve the active usage times of the phones at a constant level (around three hours) and turned the silicon level advances into software engineering benefits.

In the future, we expect to see advanced video capabilities and high speed data communications in mobile phones. These require more than one order of magnitude more computing power than is available in recent products, so we have to improve the energy efficiency, preferably at faster pace than silicon advances.

## 2. CHARACTERISTIC MODERN MOBILE COMPUTING TASKS

Mobile computing is about to enter an era of high data rate applications that require the integration of wireless wideband data modems, video cameras, net browsers, and phones into small packages with long battery powered operation times. Even the small size of phones is a design constraint as the sustained heat dissipation should be kept below 3 W [9]. In practice, much more than the capabilities of current laptop PCs is expected using around 5% of their energy and space, and at a fraction of the price. Table 3 shows a possible power budget for a multimedia phone [9]. Obviously, a 3.6 V 1000 mAh Lithium-ion battery provides only 1 hour of active operation time.

To understand how the expectations could be met, we briefly consider the characteristics of video encoding and 3GPP signal processing. These have been selected as representatives of soft and hard real time applications, and of differing hardware/software partitioning challenges.

### 2.1. *Video encoding*

The computational cost of encoding a sequence of video images into a bitstream depends on the algorithms used in the implementation and the coding standard. Table 4 illuminates the approximate costs and processing requirements of current common standards when applied to a sequence of 640-by-480 pixel (VGA) images captured at 30 frames/s. The cost of an expected "future standard" has been linearly extrapolated based on those of the past.

If a software implementation on an SISD processor is used, the operation and instructioncounts are roughly equal. This means that encoding requires the fetching and decoding

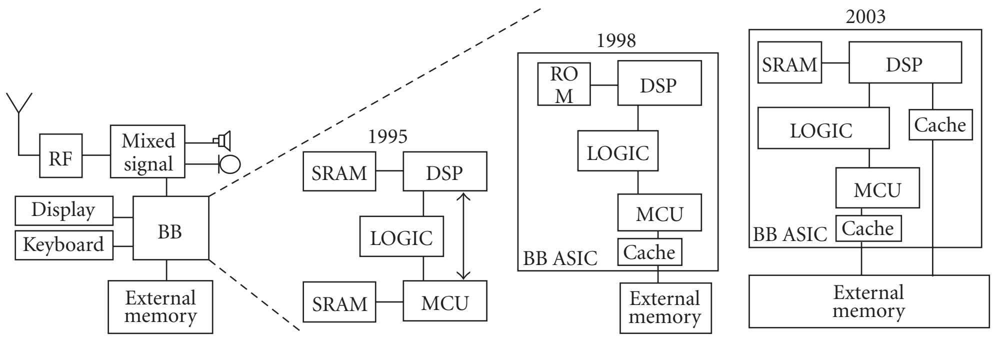

FIGURE 1: Typical implementations of mobile phones from 1995 to 2003.

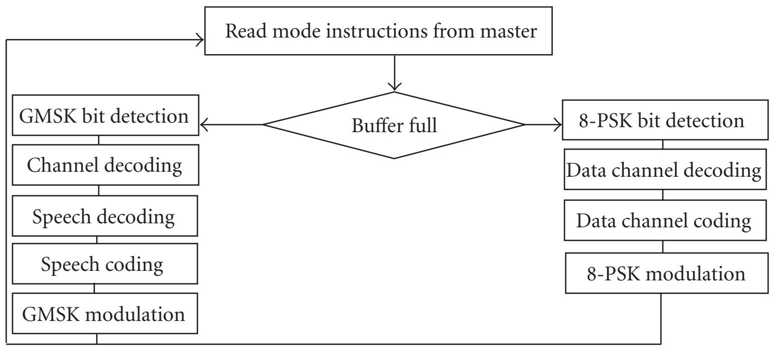

FIGURE 2: Low overhead loop-type software architecture for GSM baseband.

TABLE 4: Encoding requirements for 30 frames/s VGA video.

| Video standard | Operations/pixel | Processing speed (GOPS) |
|---|---|---|
| MPEG-4 (1998) | 200–300 | 2-3 |
| H.264-AVC (2003) | 600–900 | 6–10 |
| “Future” (2009-10) | 2000–3000 | 20–30 |

of at least 200–300 times more instructions than pixel data. This has obvious implications from energy efficiency point of view, and can be used as a basis for comparing implementations on different programmable processor architectures.

Figure 3 illustrates the Mpixels/s per silicon area (mm$^2$) and power (W) efficiencies of SISD, VLIW, SIMD, and the monolithic accelerator implementations of high image quality (> 34 dB PSNR) MPEG-4 VGA (advanced simple profile) video encoders. The quality requirement has been set to be relatively high so that the greediest motion estimation algorithms (such as a three-step search) are not applicable, and the search area was set to 48-by-48 pixels which fits into the on-chip RAMs of each studied processor.

All the processors are commercial and have instructions set level support for video encoding to speed-up at least summed absolute differences (SAD) calculations for 16-by-16 pixel macro blocks. The software implementation for the SISD is an original commercial one, while for VLIW and SIMD the motion estimators of commercial MPEG-4

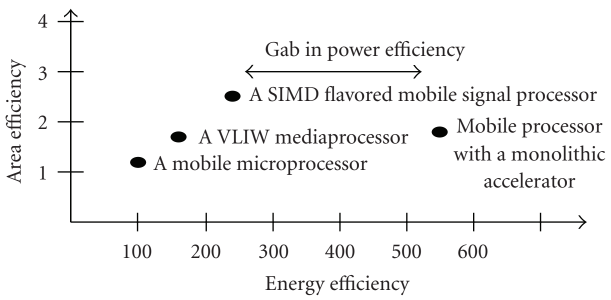

FIGURE 3: Area (Mpixels/s/mm$^2$) and energy efficiencies (Mpixels/s/W) of comparable MPEG-4 encoder implementations.

ASP codecs were replaced by iterative full search algorithms [10, 11]. As some of the information on processors was obtained under confidentiality agreements, we are unable to name them in this paper. The monolithic hardware accelerator is a commercially available MPEG-4 VGA IP block [12] with an ARM926 core.

In the figure, the implementations have been normalized to an expected low power 1 V 60 nm CMOS process. The scaling rule assumes that power consumption is proportional to the supply voltage squared and the design rule, while the die size is proportional to the design rule squared. The original processors were implemented with 0.18 and 0.13 $\mu$m CMOS.

Table 5: Relative instruction fetch rates and control unit sizes versus area and energy efficiencies.

| Solution | Instruction fetch/decode rate | Control unit size | Area efficiency | Energy efficiency |
|---|---|---|---|---|
| SISD | Operation rate | Relatively small | Lowest | Lowest |
| VLIW | Operation rate | Relatively small | Average | Average |
| SIMD | Less than operation rate | Relatively small | Highest | Good |
| Monolithic accelerator | Very low (control code) | Very small | Average | Highest |

We notice a substantial gap in energy efficiency between the monolithic accelerator and the programmed approaches. For instance, around 40 mW of power is needed for encoding 10 Mpixels/s using the SIMD extended processor, while the monolithic accelerator requires only 16 mW. In reality, the efficiency gap is even larger as the data points have been determined using only a single task on each processor. In practice, the processors switch contexts between tasks and serve hardware interrupts, reducing the hit rates of instruction and data caches, and the branch prediction mechanism. This may easily drop the actual processing throughput by half, and, respectively, lowers the energy efficiency.

The sizes of the control units and instruction fetch rates needed for video encoding appear to explain the data points of the programmed solutions as indicated by Table 5. The SISD and VLIW have the highest fetch rates, while the SIMD has the lowest one, contributing to energy efficiency. The execution units of the SIMD and VLIW occupy relatively larger portions of the processor chips: this improves the silicon area efficiency as the control part is overhead. The monolithic accelerator is controlled via a finite state machine, and needs processor services only once every frame, allowing the processor to sleep during frames.

In this comparison, the silicon area efficiency of the hardware accelerated solution appears to be reasonably good, as around 5 mm$^2$ of silicon is needed for achieving real-time encoding for VGA sequences. This is better than for the SISD (9 mm$^2$) and close to the SIMD (around 4 mm$^2$). However, the accelerator supports only one video standard, while support for another one requires another accelerator, making hardware acceleration in this case the most inefficient approach in terms of silicon area and reproduction costs.

Consequently, it is worth considering whether the video accelerator could be partitioned in a manner that would enable re-using its components in multiple coding standards. The speed-up achieved from these finer grained approaches needs to be weighted against the added overheads such as the typical 300 clock cycle interrupt latency that can become significant if, for example, an interrupt is generated for each 16-by-16 pixel macroblock of the VGA sequence.

An interesting point for further comparisons is the hibrid-SOC [13], that is, the creation of one research team. It is a multicore architecture, based on three programmable dedicated core processors (SIMD, VLIW, and SISD), intended for video encoding and decoding, and other high bandwidth applications. Based on the performance and implementation data, it comes very close to the VLIW device in Figure 2 when scaled to the 60 nm CMOS technology of Table 2, and it could rank better if explicitly designed for low power operation.

Table 6: 3GPP receiver requirements for different channel types.

| Channel type | Data rate | Processing speed (GOPS) |
|---|---|---|
| Release 99 DCH channel | 0.384 Mbps | 1-2 |
| Release 5 HSDPA channel | 14.4 Mbps | 35–40 |
| "Future 3.9G" OFDM channel | 100 Mbps | 210–290 |

### 2.2. 3GPP baseband signal processing

Based on its timing requirements, the 3GPP baseband signal processing chain is an archetypal hard real-time application that is further complicated by the heavy computational requirements shown in Table 6 for the receiver. The values in the table have been determined for a solution using turbo decoding and they do not include chip-level decoding and symbol level combining that further increase the processing needs.

The requirements of the high speed downlink packet access (HSDPA) channel that is expected to be introduced in mobile devices in the near future characterize current acute implementation challenges. Interestingly, the operation counts per received bit for each channel are roughly in the same magnitude range as with video encoding.

Figure 4 shows the organization of the 3GPP receiver processing and illuminates the implementation issues. The receiver data chain has time critical feedback loops implemented in the software; for instance, the control channel HS-SCCH is used to control what is received, and when, on the HS-DSCH data channel. Another example is the power control information decoded from "release 99 DSCH" channel that is used to regulate the transmitter power 1500 times per second. Furthermore, the channel code rates, channel codes, and interleaving schemes may change anytime, requiring software control for reconfiguring the hardware blocks of the receiver, although for clarity this is not indicated in the diagram.

The computing power needs of 3GPP signal processing have so far been satisfied only by hardware at an acceptable

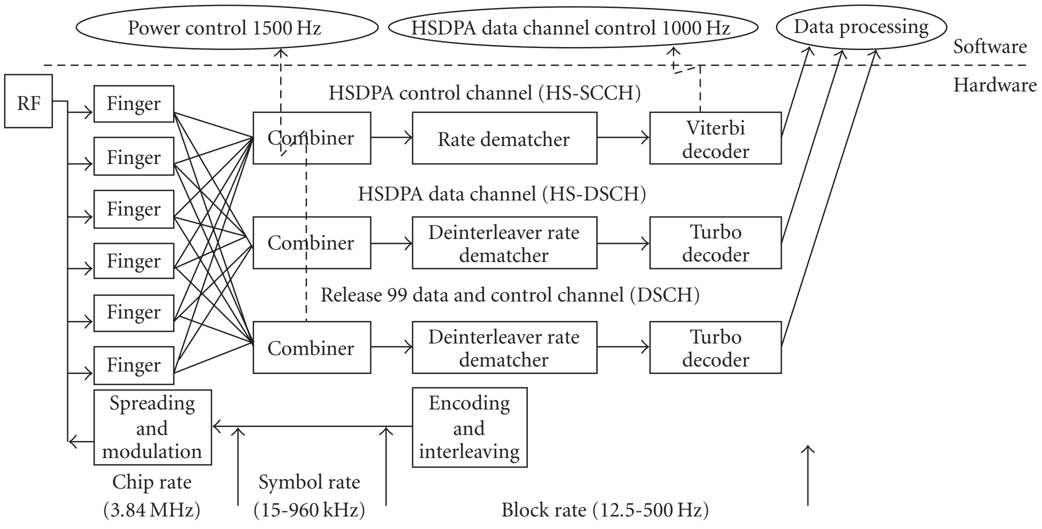

FIGURE 4: Receiver for a 3GPP mobile terminal.

energy efficiency level. Software implementations for turbo decoding that meet the speed requirement do exist; for instance, in [14] the performance of analog devices' TigerSHARC DSP processor is demonstrated. However, it falls short of the energy efficiency needed in phones and is more suitable for base station use.

For energy efficiency, battery powered systems have to rely on hardware, while the tight timings demand the employment of fine grained accelerators. A resulting large interrupt load on the control processors is an undesired side effect. Coarser grain hardware accelerators could reduce this overhead, but this is an inflexible approach and riskier when the channel specifications have not been completely frozen, but the development of hardware must begin.

With reservations on the hard real-time features, the results of the above comparison on the relative efficiencies of processor architectures for video encoding can be extended to 3GPP receivers. Both tasks have high processing requirements and the grain size of the algorithms is not very different, so they could benefit from similar solutions that improve hardware reuse and energy efficiency. In principle, the processor resources can be used more efficiently with the softer real-time demands of video coding, but if fine grained acceleration is used instead of a monolithic solution, it becomes a hard real-time task.

## 3. ANALYSIS OF THE OBSERVED DEVELOPMENT

Based on our understanding, there is no single action that could improve the talk-times of mobile phones and usage times of future applications. Rather there are multiple interacting issues for which balanced solutions must be found. In the following, we analyze some of the factors considered to be essential.

### 3.1. *Changes in voice call application*

The voice codec in 1995 required around 50% of the operation count of the more recent codec that provides improved voice quality. As a result, the computational cost of the basic GSM voice call may have even more than doubled [15]. This qualitative improvement has in part diluted the benefits obtained through advances in semiconductor processes, and is reflected by the talk-time data given for the different voice codec by mobile terminal manufacturers. It is likely that the computational costs of voice calls will increase even in the future with advanced features.

### 3.2. *The effect of preemptive real-time operating systems*

The dominating scheduling principle used in embedded systems is "rate monotonic analysis (RMA)" that assigns higher static priorities for tasks that execute at higher rates. When the number of tasks is large, utilizing the processor at most up to 69% guarantees that all deadlines are met [16]. If more processor resources are needed, then more advanced analysis is needed to learn whether the scheduling meets the requirements.

In practice, both our video and 3GPP baseband examples are affected by this law. A video encoder, even when fully implemented in software, is seldom the only task in the processor, but shares its resources with a number of other tasks. The 3GPP baseband processing chain consists of several simultaneous tasks due to time critical hardware/software interactions.

With RMA, the processor utilization limit alone may demand even 40% higher clock rates than was necessary with the static cyclic scheduling used in early GSM phones in which the clock could be controlled very flexibly. Now, due to the scheduling overhead that has to be added to the task durations, a 50% clock frequency increase is close to reality.

We admit that this kind of comparison is not completely fair. Static cyclic scheduling is no longer usable as it is unsuitable for providing responses for sporadic events within a short fixed time, as required by the newer features of the

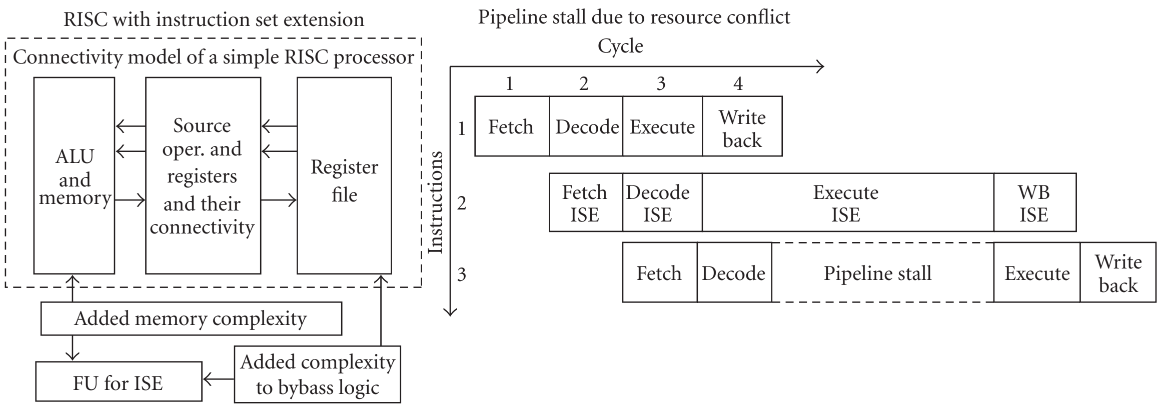
RISC with instruction set extension
Connectivity model of a simple RISC processor
ALU and memory
Source oper. and registers and their connectivity
Register file
Added memory complexity
FU for ISE
Added complexity to bybass logic
Pipeline stall due to resource conflict
Cycle
1
2
3
4
Instructions
1
2
3
Fetch
Decode
Execute
Write back
Fetch ISE
Decode ISE
Execute ISE
WB ISE
Fetch
Decode
Pipeline stall
Execute
Write back

Figure 5: Hardware acceleration via instruction set extension.

phones. The use of dynamic priorities and earliest-deadline-first (EDF) or least-slack algorithm [17] would improve processor utilization over RMA, although this would be at the cost of slightly higher scheduling overheads that can be significant if the number of tasks is large. Furthermore, embedded software designers wish to avoid EDF scheduling, because variations in cache hit ratios complicate the estimation of the proximity of deadlines.

### 3.3. The effect of context switches on cache and processor performance

The instruction and data caches of modern processors improve energy efficiency when they perform as intended. However, when the number of tasks and the frequency of context switches is high, the cache-hit rates may suffer. Experiments [18] carried out using the MiBench [19] embedded benchmark suite on an MIPS 4KE-type instruction set architecture revealed that with a 16 kB 4-way set associative instruction cache the hit-rate averaged around 78% immediately after context switches and 90% after 1000 instructions, while 96% was reached after the execution of 10 000 instructions.

Depending on the access time differential between the main memory and the cache, the performance impact can be significant. If the processor operates at 150 MHz with a 50-nanosecond main memory and an 86% cache hit rate, the execution time of a short task slice (say 2000 instructions) almost doubles. Worst of all, the execution time of the same piece of code may fluctuate from activation to activation, causing scheduling and throughput complications, and may ultimately force the system implementers to increase the processor clock rate to ensure that the deadlines are met.

Depending on the implementations, both video encoder and 3GPP baseband applications operate in an environment that executes up to tens of thousands of interrupts and context switches in a second. Although this facilitates the development of systems with large teams, the approach may have a significant negative impact on energy efficiency.

More than a decade ago (1991), Mogul and Borg [20] made empirical measurements on the effects of context switches on cache and system performance. After a partial reproduction of their experiments on a modern processor, Sebek [21] comments "it is interesting that the cache related preemption delay is almost the same," although the processors have became a magnitude faster. We may make a similar observation about GSM phones and voice calls: current implementations of the same application require more resources than in the past. This cycle needs to be broken in future mobile terminals and their applications.

### 3.4. The effect of hardware/software interfacing

The designers of mobile phones aim to create common platforms for product families. They define application programming interfaces that remain the same, regardless of system enhancements and changes in hardware/software partitioning [8]. This has made middleware solutions attractive, despite worries over the impact on performance. However, the low level hardware accelerator/software interface is often the most critical one.

Two approaches are available for interfacing hardware accelerators to software. First, a hardware accelerator can be integrated into the system as an extension to the instruction set, as illustrated with Figure 5. In order to make sense, the latency of the extension should be in the same range as the standard instructions, or, at most, within a few instruction cycles, otherwise the interrupt response time may suffer. Short latency often implies large gate count and high bus bandwidth needs that reduce the economic viability of the approach, making it a rare choice in mobile phones.

Second, an accelerator may be used in a peripheral device that generates an interrupt after completing its task. This principle is demonstrated in Figure 6, which also shows the role of middleware in hiding details of the hardware. Note that the legend in the picture is in the order of priority levels.

If the code in the middleware is not integrated into the task, calls to middleware functions are likely to reduce the cache hit rate. Furthermore, to avoid high interrupt overheads, the execution time of the accelerators should

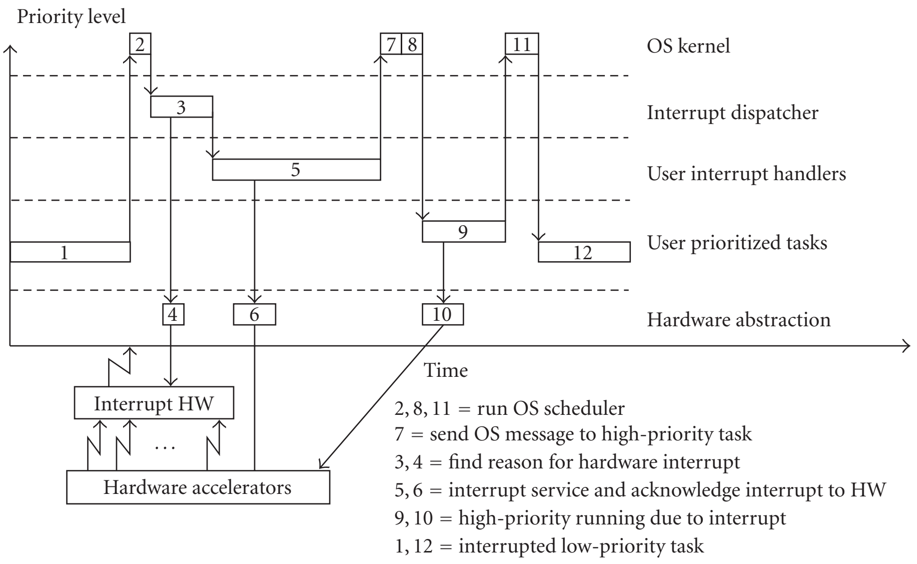
Priority level
OS kernel
Interrupt dispatcher
User interrupt handlers
User prioritized tasks
Hardware abstraction
Time
Interrupt HW
Hardware accelerators
1
2
3
4
5
6
7
8
9
10
11
12
2, 8, 11 = run OS scheduler
7 = send OS message to high-priority task
3, 4 = find reason for hardware interrupt
5, 6 = interrupt service and acknowledge interrupt to HW
9, 10 = high-priority running due to interrupt
1, 12 = interrupted low-priority task

FIGURE 6: Controlling an accelerator interfaced as a peripheral device.

TABLE 7: Energy efficiencies and silicon areas of ARM processors.

| Processor | Processor max. clock frequency (MHz) | Silicon area ($mm^2$) | Power consumption ( mW/MHz) |
|---|---|---|---|
| ARM9 (926EJ-S) | 266 | 4.5 | 0.45 |
| ARM10 (1022E) | 325 | 6.9 | 0.6 |
| ARM11 (1136J-S) | 550 | 5.55 | 0.8 |

preferably be thousands of clock cycles. In practice, this approach is used even with rather short latency accelerators, as long as it helps in achieving the total performance target. The latencies from middleware, context switches, and interrupts have obvious consequences for energy efficiency.

Against this background, it is logical that the monolithic accelerator turned out to be the most energy efficient solution for video encoding in Figure 3. From the point of view, the 3GPP baseband a key to energy efficient implementation in a given hardware lies in pushing down the latency overheads.

It is rather interesting that anything in between 1-2 cycle instruction set extensions and peripheral devices executing thousands of cycles can result in grossly inefficient software. If the interrupt latency in the operating system environment is around 300 cycles and 50 000 interrupts are generated per second, 10% of the 150 MHz processor resources are swallowed by this overhead alone, and on top of this we have middleware costs. Clearly, we have insufficient expertise in this bottleneck area that falls between hardware and software, architectures and mechanisms, and systems and components.

### 3.5. The effect of processor hardware core solutions

Current DSP processor execution units are deeply pipelined to increase instruction execution rates. In many cases, however, DSP processors are used as control processors and have to handle large interrupt and context switch loads. The result is a double penalty: the utilization of the pipeline decreases and the control code is inefficient due to the long pipeline. For instance, if a processor has a 10-level pipeline and 1/50 of the instructions are unconditional branches, almost 20% of the cycles are lost. Improvements offered by the branch prediction capabilities are diluted by the interrupts and context switches.

The relative sizes of control units of typical low power DSP processors and microcontrollers have increased during recent years due to deeper pipelining. However, when executing control code, most of the processor is unused. This situation is encountered with all fine grained hardware accelerator-based implementations regardless of whether they are video encoder or 3GPP baseband solutions. Obviously, rethinking the architectures and their roles in the system implementations is necessary. To illustrate the impact of increasing processor complexity on the energy efficiency, Table 7 shows the characteristics of 32-bit ARM processors implemented using a 130 nm CMOS process [5]. It is apparent that the energy efficiencies of processor designs are increasing, but this development has been masked by silicon process developments. Over the past ten years the relative efficiency appears to have slipped approximately by a factor of two.

Table 8: Approximate efficiency degradations.

| Degradation cause | Low estimate | Probable degradation |
|---|---|---|
| Computational cost of voice call application | 2 | 2.5 |
| Operating system and interrupt overheads | 1.4 | 1.6 |
| API and middleware costs | 1.2 | 1.5 |
| Execution time jitter provisioning | 1.3 | 2 |
| Processor energy/instruction | 1.8 | 2.5 |
| Execution pipeline overheads | 1.2 | 1.5 |
| Total (multiplied) | 9.4 | 45 |

### 3.6. *Summary of relative performance degradations*

When the components of the above analysis are combined as shown in Table 8, they result in a degradation factor of at least around 9-10, but probably around 45. These are relative energy efficiency degradations and illustrate the traded-off energy efficiency gains at the processing system level. The probable numbers appear to be in line with the actual observed development.

It is acknowledged in industry that approaches in system development have been dictated by the needs of software development that has been carried out using the tools and methods available. Currently, the computing needs are increasing rapidly, so a shift of focus to energy efficiency is required. Based on Figure 3, using suitable programmable processor architectures can improve the energy efficiency significantly. However, in baseband signal processing the architectures used already appear fairly optimal. Consequently, other means need to be explored too.

## 4. DIRECTIONS FOR RESEARCH AND DEVELOPMENT

Looking back to the phone of 1995 in Table 1, we may consider what should have been done to improve energy efficiency at the same rate as silicon process improvement. Obviously, due to the choices made by system developers, most of the factors that degrade the relative energy efficiency are software related. However, we do not demand changes in software development processes or architectures that are intended to facilitate human effort. So solutions should primarily be sought from the software/hardware interfacing domain, including compilation, and hardware solutions that enable the building of energy efficient software systems.

To reiterate, the early baseband software was effectively multi-threaded, and even simultaneously multithreaded with hardware accelerators executing parallel threads, without interrupt overhead, as shown in Figure 7. In principle, a suitable compiler could have replaced manual coding in creating the threads, as the hardware accelerators had deterministic latencies. However, interrupts were introduced and later solutions employed additional means to hide the hardware from the programmers.

Having witnessed the past choices, their motivations, and outcomes, we need to ask whether compilers could be used to hide hardware details instead of using APIs and middleware. This approach could in many cases cut down the number of interrupts, reduce the number of tasks and context switches, and improve code locality— all improving processor utilization and energy efficiency. Most importantly, hardware accelerator aware compilation would bridge the software efficiency gap between instruction set extensions and peripheral devices, making "medium latency" accelerators attractive. This would help in cutting the instruction fetch and decoding overheads.

The downside of a hardware aware compilation approach is that the binary software may no longer be portable, but this is not important for the baseband part. A bigger issue is the paradigm change that the proposed approach represents. Compilers have so far been developed for processor cores; now they would be needed for complete embedded systems. Whenever the platform changes, the compiler needs to be upgraded, while currently the changes are concentrated on the hardware abstraction functionality.

Hardware support for simultaneous fine grained multithreading is an obvious processor core feature that could contribute to energy efficiency. This would help in reducing the costs of scheduling.

Another option that could improve energy efficiency is the employing of several small processor cores for controlling hardware accelerators, rather that a single powerful one. This simplifies real-time system design and reduces the total penalty from interrupts, context switches, and execution time jitter. To give a justification for this approach, we again observe that the W/MHz figures for the 16-bit ARM7/TDMI dropped by factor 35 between 0.35 and 0.13 $\mu$m CMOS processes [5]. Advanced static scheduling and allocation techniques [22] enable constructing efficient tools for this approach, making it very attractive.

## 5. SUMMARY

The energy efficiency of mobile phones has not improved at the rate that might have been expected from the advances in silicon processes, but it is obviously at a level that satisfies most users. However, higher data rates and multimedia applications require significant improvements, and encourage us to reconsider the ways software is designed, run, and interfaced with hardware.

Significantly improved energy efficiency might be possible even without any changes to hardware by using software solutions that reduce overheads and improve processor utilization. Large savings can be expected from applying architectural approaches that reduce the volume of instructions fetched and decoded. Obviously, compiler technology is the key enabler for improvements.

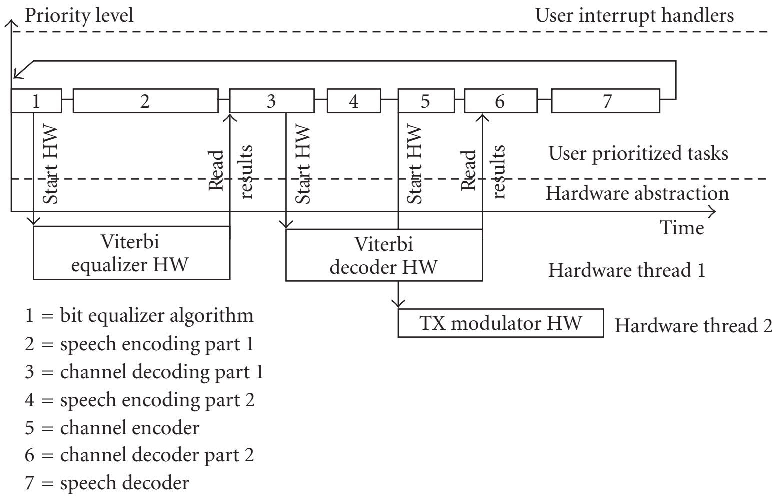

Figure 7: The execution threads of an early GSM mobile phone.

## ACKNOWLEDGMENTS

Numerous people have directly and indirectly contributed to this paper. In particular, we wish to thank Dr. Lauri Pirttiaho for his observations, comments, questions, and expertise, and Professor Yrjö Neuvo for advice, encouragement, and long-time support, both from the Nokia Corporation.

## REFERENCES

[1] GSM Association, "TW.09 Battery Life Measurement Technique," 1998, http://www.gsmworld.com/documents/index.shtml.

[2] Nokia, "Phone models," http://www.nokia.com/.

[3] M. Anis, M. Allam, and M. Elmasry, "Impact of technology scaling on CMOS logic styles," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 8, pp. 577–588, 2002.

[4] G. Frantz, "Digital signal processor trends," *IEEE Micro*, vol. 20, no. 6, pp. 52–59, 2000.

[5] The ARM foundry program, 2004 and 2006, http://www.arm.com/.

[6] 3GPP: TS 05.01, "Physical Layer on the Radio Path (General Description)," http://www.3gpp.org/ftp/Specs/html-info/0501.htm.

[7] J. Doyle and B. Broach, "Small gains in power efficiency now, bigger gains tomorrow," *EE Times*, 2002.

[8] K. Jyrkkä, O. Silven, O. Ali-Yrkkö, R. Heidari, and H. Berg, "Component-based development of DSP software for mobile communication terminals," *Microprocessors and Microsystems*, vol. 26, no. 9-10, pp. 463–474, 2002.

[9] Y. Neuvo, "Cellular phones as embedded systems," in *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC '04)*, vol. 1, pp. 32–37, San Francisco, Calif, USA, February 2004.

[10] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp. 501–504, 2000.

[11] H.-S. Wang and R. M. Mersereau, "Fast algorithms for the estimation of motion vectors," *IEEE Transactions on Image Processing*, vol. 8, no. 3, pp. 435–438, 1999.

[12] 5250 VGA encoder, 2004, http://www.hantro.com/en/products/codecs/hardware/5250.html.

[13] S. Moch, M. Bereković, H. J. Stolberg, et al., "HIBRID-SOC: a multi-core architecture for image and video applications," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 3, pp. 55–61, 2004.

[14] K. K. Loo, T. Alukaidey, and S. A. Jimaa, "High performance parallelised 3GPP turbo decoder," in *Proceedings of the 5th European Personal Mobile Communications Conference (EPMCC '03)*, Conf. Publ. no. 492, pp. 337–342, Glasgow, UK, April 2003.

[15] R. Salami, C. Laflamme, B. Bessette, et al., "Description of GSM enhanced full rate speech codec," in *Proceedings of the IEEE International Conference on Communications (ICC '97)*, vol. 2, pp. 725–729, Montreal, Canada, June 1997.

[16] M. H. Klein, *A Practitioner's Handbook for Real-Time Analysis*, Kluwer, Boston, Mass, USA, 1993.

[17] M. Spuri and G. C. Buttazzo, "Efficient aperiodic service under earliest deadline scheduling," in *Proceedings of Real-Time Systems Symposium*, pp. 2–11, San Juan, Puerto Rico, USA, December 1994.

[18] J. Stärner and L. Asplund, "Measuring the cache interference cost in preemptive real-time systems," in *Proceedings of the ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '04)*, pp. 146–154, Washington, DC, USA, June 2004.

[19] M. R. Gathaus, J. S. Ringenberg, D. Ernst, T. M. Austen, T. Mudge, and R. B. Brown, "MiBench: a free, commercially representative embedded benchmark suite," in *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization (WWC-4 '01)*, pp. 3–14, Austin, Tex, USA, December 2001.

[20] J. C. Mogul and A. Borg, "The effect of context switches on cache performance," in *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '91)*, pp. 75–84, Santa Clara, Calif, USA, April 1991.

[21] F. Sebek, "Instruction cache memory issues in real-time systems," Technology Licentiate thesis, Department of Computer Science and Engineering, Mälardalen University, Västerås, Sweden, 2002.

[22] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, Marcel Dekker, New York, NY, USA, 2000.