The Imsys processor and its feasibility for advanced applications

1. About this document

This document aims to show that Imsys' processor technology can be a good choice for modern embedded systems, not only for general IoT-applications but also particularly in certain applications of interest in advanced automotive systems. Typical for these will be that the processor will act not only as a CPU but also as a DSP. In addition to audio, where Imsys already has working microcode, the application could be radar and lidar, and also neural networks for processing sensor data from these.

2. Background and comments

Relevant information on silicon area for competing processor cores (and especially their auxiliary logic) can be challenging to find, and restricting the search to implementations in only one particular technology node will further limit the availability of published data. This report will therefore first briefly discuss the scaling between different technology nodes. This makes it possible to scale not only area but also speed and consumption, and thereby to do meaningful comparisons between different processor designs, also when the known examples happen to be implemented in different technologies.

Note that the memory that stores the compiled code can require much larger area than the core. This means that code density – a property of the instruction set architecture, is important for the cost of the die.

For DSP algorithms the situation is different. Typically these are small, while their speed and energy consumption is important. With a given technology node and a given task, battery life depends inversely on energy consumption, i.e., the product of power consumption and execution time for the task.

With this background, this report will discuss, more in general,

- Scaling
- Silicon area
- Code density
- DSP algorithms
- Execution speed
- Energy consumption

before going into the characteristics of the Imsys processor that are relevant for the new applications.

3. Scaling (Moore's Law)

The purpose of this section is to explain how silicon area is scaled in Section 4. It is here also shown how speed and power consumption changes between nodes, which is used in Sections 9, 10 and 11.

CMOS IC fabrication processes are developed in preplanned steps, with new technology nodes appearing at intervals of about two years. For computers it takes around three years for a new node to reach its peak and replace the previous one, and the peak is then passed within a year because the next one is already in production and increasing.

This is not true for embedded systems, because execution speed is here not the only thing that is important – in fact it is now usually less important than energy efficiency and cost, because

- speed is now almost always sufficient
- products are increasingly mass-produced and cost sensitive
- they are also increasingly battery powered.

Also, the newer nodes have higher leakage current, much higher tooling cost, and after 28 nm even the manufacturing cost per transistor increases. Therefore, several technology nodes, perhaps more than ever before, are in production and also used in new IC designs.

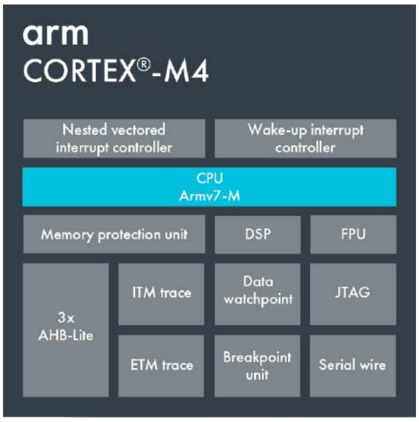
Every new node doubles the transistor density, and divides transistor delay by the square root of two (these are approximate average changes). For many years, voltage and current were also divided by that amount, and the power density at full (i.e. doubled) speed could therefore be kept unchanged. This is the classical scaling (a.k.a. Dennard scaling). Beyond 65 nm the power density is doubling for every new node (Leakage limited scaling). The scaling is described in the following table, where the scale factor is the square root of two:

	Classical scaling	Leakage limited scaling
Linear dimensions of transistors (W, L)	1/S =0,71	1/S =0,71
Transistor area (A=W*L)	$(1/S)^2$ = 0,5	$(1/S)^2$ = 0,5
Transistors/mm^2 (N)	S^2 = 2	S^2 = 2
Supply voltage (V)	1/S =0,71	1
Switching current (i)	1/S =0,71	1
Transistor delay (switching time)	1/S =0,71	1/S =0,71
Max switching frequency	S	S
Transistor pwr consumption (p) at max freq	V*i = 0,5	V*i = 1
Pwr/mm^2 at max freq	N*p = 1	N*p = 2
Utilization at given pwr level	1	0,5

This is discussed more in relation to many-core implementations in Section 11.

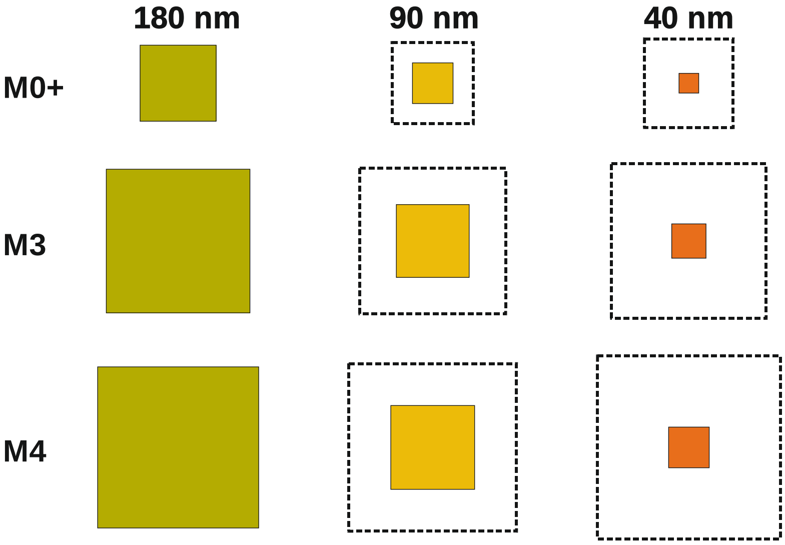
4. Silicon area

It would be relevant to compare Imsys processors with both CPU and DSP cores (and combinations of them), but the competing cores that first come to mind are those of the Cortex-M series from ARM.



These processors come with a set of options, but on ARM's website only the size of their central CPU block is published.

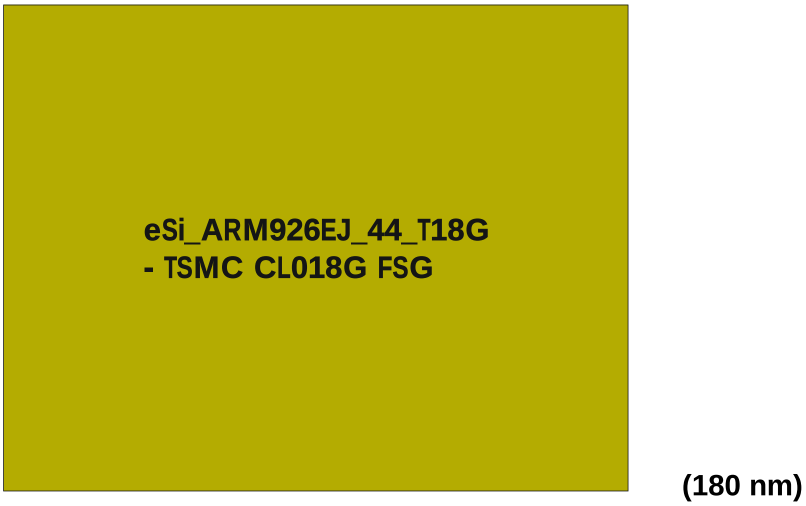
We are mainly interested in M0+, M3 and M4 cores, the relative sizes of which are shown below for three technology nodes (which are two generations apart):



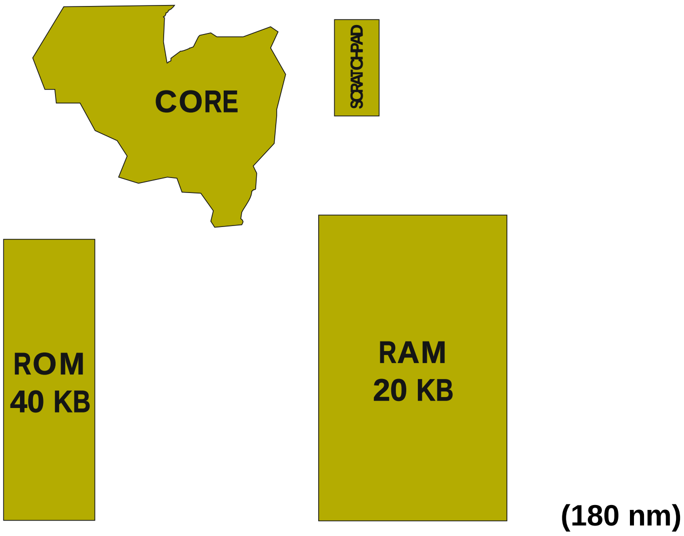
The colored squares above show the relative areas of these three Cortex-M cores, according to ARM's published data. The dashed contours show the area that would result if the core were scaled to 180 nm. Note that the scale factor S is not exactly the square root of 2; the dimensions have been scaled with the factors 180/90 and 180/40, respectively.

Note that these are the basic cores, of Harvard type, i.e., for separate program and data memory, and that debug, data trace, memory protection, and floating point are excluded.

Below is shown, to the same scale, the actual outline of a complete processor IP block, in 180 nm technology, with core similar to Cortex-M3 and including 4KB I-cache and 4 KB D-cache.



The figure below shows some blocks in the IM3000 IC from Imsys.

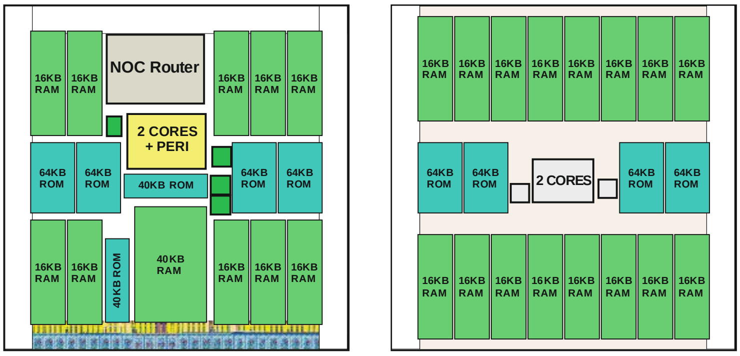


The irregular shape is the actual outline of the core logic block. The IM3000 is made in UMC 180G process, similar to that used for the ARM926 IP block shown in the same scale in the previous figure. The Imsys core can of course have another shape (e.g. rectangular) in another implementation (and it can also be smaller since there was no need for density optimization here). The small rectangle is the scratchpad, which is also part of the core.

4.1.1. The 65 nm generation from Imsys

The figures below show Imsys technology in 65 nm (UMC65LL), still in the same scale as used above. To the left is the verified dual core prototype described in Ning Ma's doctoral thesis. It contains two Imsys cores, full peripheral system containing Ethernet, a total of 336 KB ROM + 240 KB RAM, and a 5-port NOC router (because this chip also serves as proof of concept for a many-core tile; the active area, which is 2,5 mm^2, can be combined with other tiles like it in an array, with no glue logic in between the cells.

To the right is shown a preliminary layout for Imsys' next generation chip. The dimensions are the same (1.875 x 1.875 mm). The router has been removed and replaced with additional memory; the chip contains 256KB ROM + 256 KB RAM. Two rows of pads have also been removed, giving ample room for high-speed interface to a companion chip – either an FPGA or an ASIC (in less expensive technology), which may contain application specific ROM, RTC, GPIO (capable of motor driving and supply switching), analog circuitry, and various I/O interfaces that may include DDR memory interface for secondary storage, and Imsys' GbE interface with timestamping for IEEE1588. The processor chip itself will be equipped with an ultra high speed SD Card interface, allowing instantaneous startup.



(65 nm)

The two processor cores in the center of the die are shown below scaled to 180nm:

Cores scaled to 180 nm

4.1.2. *Area comparison tables*

Area comparison for processors implemented in UMC L180 GII process (ARM data from UMC)	**mm2**	**normalized**
ARM926EJ with 16KB+16KB cache	8,98	**6,5**
ARM922T with 8KB+8KB cache	8,08	**5,9**
ARM946E with 8KB+8KB cache	5,88	**4,3**
Imsys core with 80KB control store	1,38	**1,0**

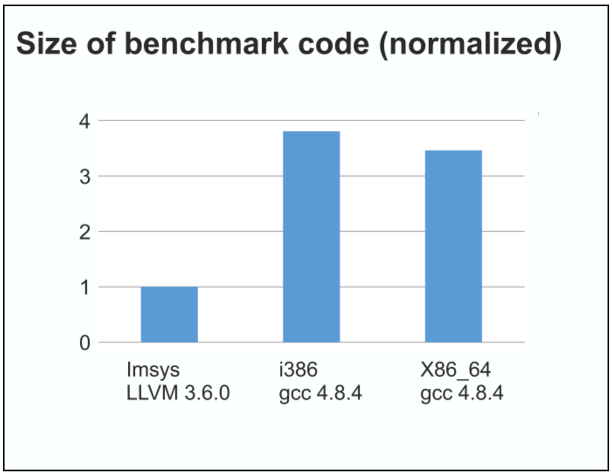
Comparison with ARM Cortex-A35 (data available only for 28 nm)	**mm2**	**normalized**
ARM Cortex-A35 without all options	0,40	**3,1**
ARM Cortex-A35 with all options	4,00	**31,0**
Imsys core with 80KB control store (scaled to 28 nm)	0,13	**1,0**

5. Code density

The Instruction set architecture of CISC machines generally has higher code density because

- each one of their instructions can, independently, use as many clock cycles as it needs, combining memory accesses and operations, and thus accomplish more work, which means that a typical program contains fewer instructions
- there can be a larger number of different instruction types and data formats, which also reduces the number of instructions used in a program
- instructions can have different lengths and thus use as many bytes as they need, which reduces the number of unused bits in the compiled program.

The Imsys processor has the extra advantage of not being bound to any of the ancient established instruction sets, which were optimized for human programmers rather than for compilers.

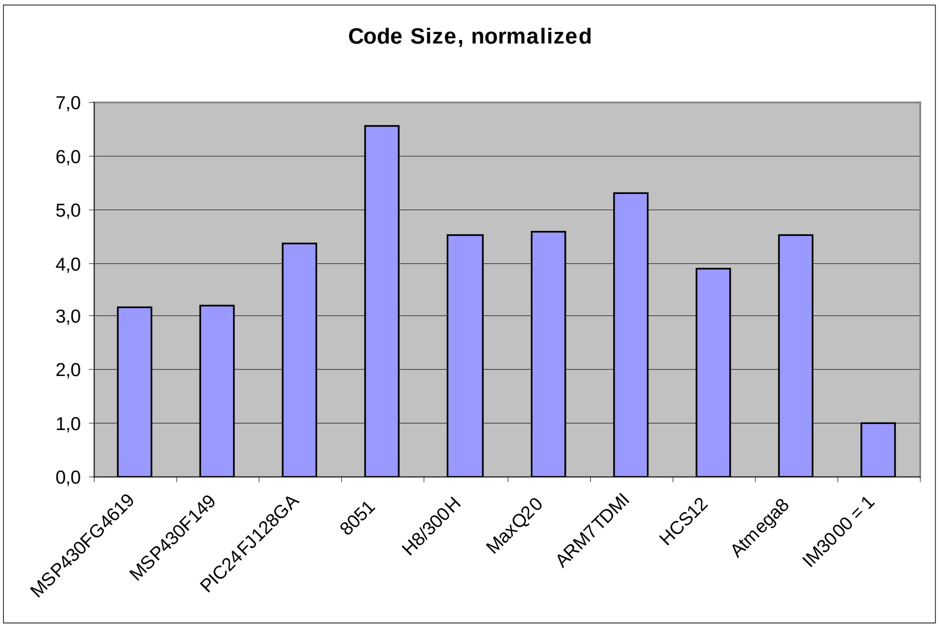


The diagram here shows the relative sizes of object code produced by LLVM compilers for Imsys and for the 32- and 64-bit variants of Intel x86, another CISC machine.

The benchmark source code used here is an independent and relevant benchmark suite for embedded systems, found in the Application Report SLAA205 from TI, "MSP430 Competitive Benchmarking".

The following, older, diagram compares with several processors made for embedded systems. The same benchmark code from TI was used, and all measurements except the one for Imsys have been made and published by TI.

The older, stack-based, Imsys instruction set was used here. The new LLVM-based set has somewhat higher code density, i.e.. the non-Imsys bars would have been even higher if Imsys had used that.



6. DSP algorithms

An example of a signal processing algorithm that has been microcoded is a 64-point complex FFT using the radix-4 method and 16+16-bit data format. This routine, and its inverse, is being used in a 128-point real-value FFT, and its inverse, for a planned audio processing application.

The 64-point CFFT algorithm can, in a straight-forward way, be extended to 256-point and 1024-point. Performance is listed in Section 9.1.

DSP routines, or parts of them, can be made into special CPU instructions, but they can also run autonomously in the background, interacting with software through interrupts and control blocks in memory. These processes can be invoked or synchronized by timers or I/O-units.

7. Crypto algorithms, etc.

27 crypto and hash algorithms (those used by SSL/TLS) have been microcoded and “packaged” as 27 special opcodes in an instruction set amendment.

The core has an internal memory block that is accessible only from the microcode. It is invisible to software and has no external connection. Provided there is a supply or backup voltage to the chip, it can be used for storing crypto keys. The core’s microcode can also generate private RSA keys by a completely internal, inaccessible process, and store them in this memory.

This memory, and the RTC (real-time clock), are powered from any of the core or I/O supplies if no backup battery is connected. Thus, when power is on, the backup battery can be replaced without erasing the contents. It also means that energy from the backup battery is consumed only when power is off.

8. Peripheral control

Examples of microprogrammed peripheral control are:

- the DMA function, which mainly uses the core’s own data path.
- the Ethernet interface (10/100MBps, with IEEE1588 timestamping), which has very little dedicated hardware since the Media Access Control protocol layer is microcoded and uses buffer areas in main memory. (An FPGA version for Gigabit Ethernet also exists but is currently not used by Imsys itself.)
- Audio and video input/output. LCD screens with different resolutions and aspect ratios can be refreshed directly from main memory, through the processor core, with very little glue logic.
- Touch screen interface.
- Motors can be driven with microprogrammed control of the commutation, invisible to the software.
- Microcode could also control the switching of a switch-mode power supply for connected components (e.g. microphone and amplifier) or even for powering the system itself.

9. Execution speed

This section describes Imsys performance for three different kinds of processing for which general-purpose cores are inefficient. DSP cores are good at signal processing but less efficient at general-purpose code. Some common g-p cores have extra hardware, i.e. extra cost, for helping Java VM interpretation. Cryptography is becoming important and involves unusual special operations.

9.1. Signal processing

An independent evaluation using seven representative benchmarks came to the following results, comparing with dsPIC, a DSP chip from Microchip Technology Inc.,, which was made in similar CMOS technology.

Imsys DSP Benchmark Results			
		Execution time, us	
Function	**Name**	**IM3000 Optimized**	**dsPIC Optimized**
Product of conjugate	PROD_CONJ	651	920
Atan2 computation	ATAN2	293	420
Cosin computation	COS_COMP	359	830
Cosin-Sin computation	COS_SIN	161	200
Vector product	DOT_PROD	431	132
Array copy	COPY	33	26
FFT computation	FFT	1928*	2800

*) The FFT result for Imsys is for a new microcode version, mentioned below.

An important finding from this evaluation: While showing slightly better results for Imsys, it also shows Imsys power consumption to be only 1/3 of that of dsPIC, as seen in the last diagram of Section 10, where Platfom C is dsPIC. It is 1/7 of that of ARM Cortex-M3 (Platform A), which is slower than dsPIC.

A complex 64-point FFT has now been developed and tested. Extension to 1024-points is simple and should result in execution time of around 1928 us as shown, clearly lower than the dsPIC result. (Simplification for real inputs, which was what the benchmark related to, will reduce the time further.) The following table shows expected results based on the tested 64-point algorithm:

Imsys FFT performance			
	Execution time, us		
Complex (Q15,Q15) in/out format	**Imsys**	**Imsys 65 nm chip**	
	IM3000	**One core**	**Both cores**
64-point FFT	84	40	20
256-point FFT	453	216	108
1024-point FFT	1928	920	460
64x64-point 2D FFT, suitable for face recognition	11346	5414	2800

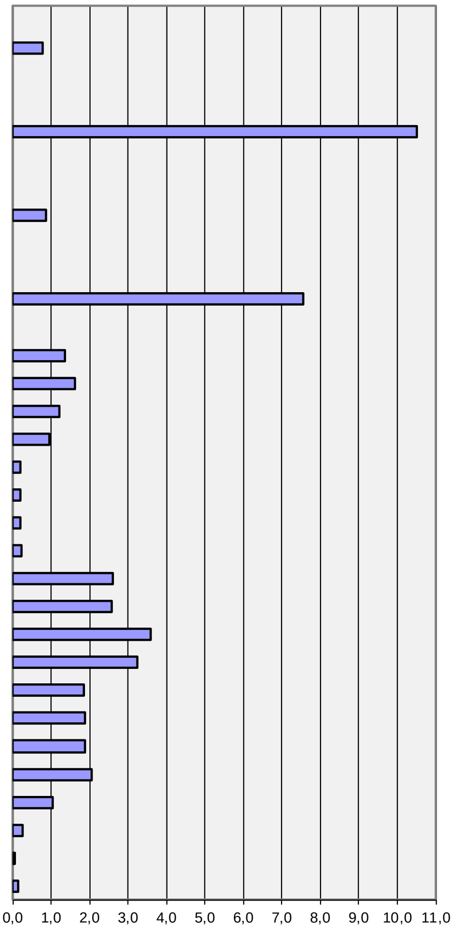
9.2. Java

Systronix Inc. measured performance for JVM bytecodes on a number of processors, among them

- IM3000, at 167 MHz
- ARM920T, at 180 MHz.

These two had on average the same performance, as seen below. As expected, the Imsys processor is faster at moving data in DRAM (due to its patented memory interface), slower at integer operations (due to its smaller ALU), and faster at byte operations. What is most interesting is that Imsys is 3 times faster at floating point operations.
Note that the ARM processor has the benchmark code in its cache all the time, while Imsys doesn't use cache.

JVM byte code group	Speed ratio Imsys/ARM
32768 element byte array 262144 element accesses	0,8
32768 element byte array 262144 element copies (note 5)	10,5
16383 element int array 131072 element accesses (note 1)	0,9
16383 element int array 131072 element copies (note 5)	7,6
byte add	1,4
byte sub	1,6
byte mul	1,2
byte div	0,9
int add	0,2
int sub	0,2
int mul	0,2
int div	0,2
float add	2,6
float sub	2,6
float mul	3,6
float div	3,2
double add	1,8
double sub	1,9
double mul	1,9
double div	2,1
string concat	1,0
string compare	0,3
method calls	0,1
object creations	0,1



GEOMETRIC MEAN, ALL	**1,0**
GEOMETRIC MEAN, float add/sub/mul/div	**3,0**

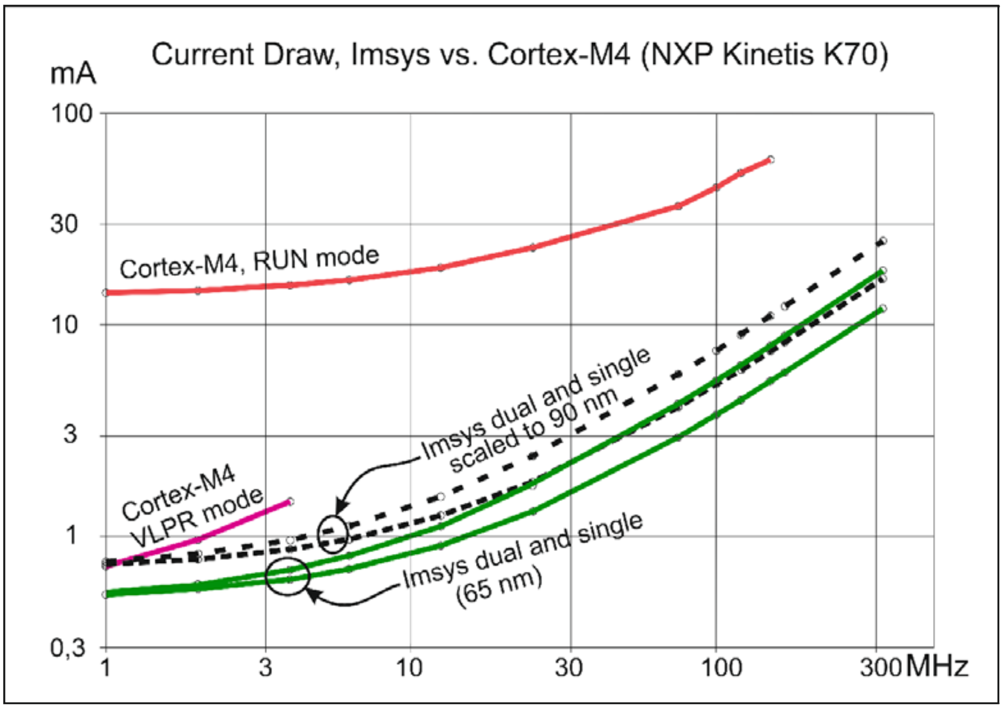
9.3. Crypto

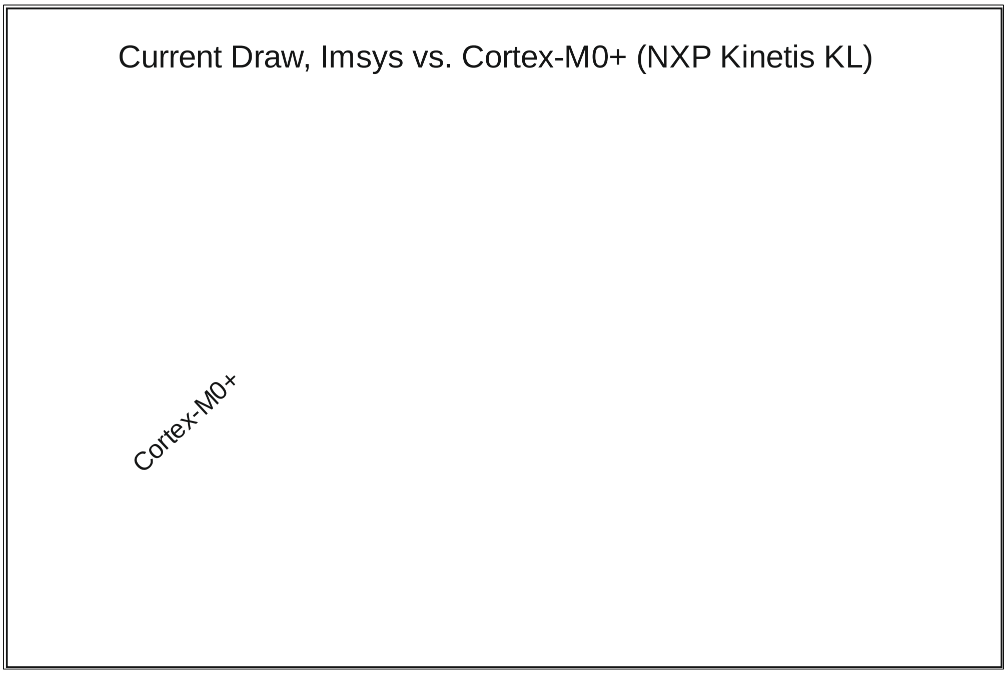
Imsys has 27 opcodes for cryptographic algorithms. Cryptographic algorithms are compute-intensive, and optimization of these can save very much time and energy, and thereby enable secure communication where that would otherwise not be feasible. The table below is not a comparison between different processors; instead it shows the speed-up obtained when important cryptographic algorithms, defined in C language, were optimized by microprogrogramming. The RSA algorithm is a good example to show the reasons for the efficiency: Almost all the work is in loops that are repeated an extremely high number of times. The processor multiplies in every one of the cycles in these loops – there is no time wasted on “overhead”; all such activity is hidden.

Test case	C code (µs)	Special opcodes (µs)	Factor
RSA Test Cases			
1024 encrypt	121611	6554	18
1024 decrypt	5473623	400753	13
2048 encrypt	980828	71086	13
2048 decrypt	41020497	3077878	13
ARC4 Test Cases			
ARC4-40 encrypt	1250	35	35
ARC4-40 decrypt	1250	35	35
ARC4-64 encrypt	1250	37	33
ARC4-64 decrypt	1250	37	33
ARC4-128 encrypt	2458	72	34
ARC4-128 decrypt	2458	72	34
DES Test Case			
DES encrypt	3667	84	43
DES encrypt	3667	84	43
AES Test Cases			
Key128 encrypt	553	47	12
Key128 decrypt	983	74	13
Key192 encrypt	635	74	11
Key192 decrypt	1128	88	12
Key256 encrypt	778	59	13
Key256 decrypt	1372	98	13
HASH Test Cases			
MD5 7 test cases	2703	983	2.7
SHA1 4 test cases	5591	1925	2.9
SHA256 3 test cases	5059	1516	3.3

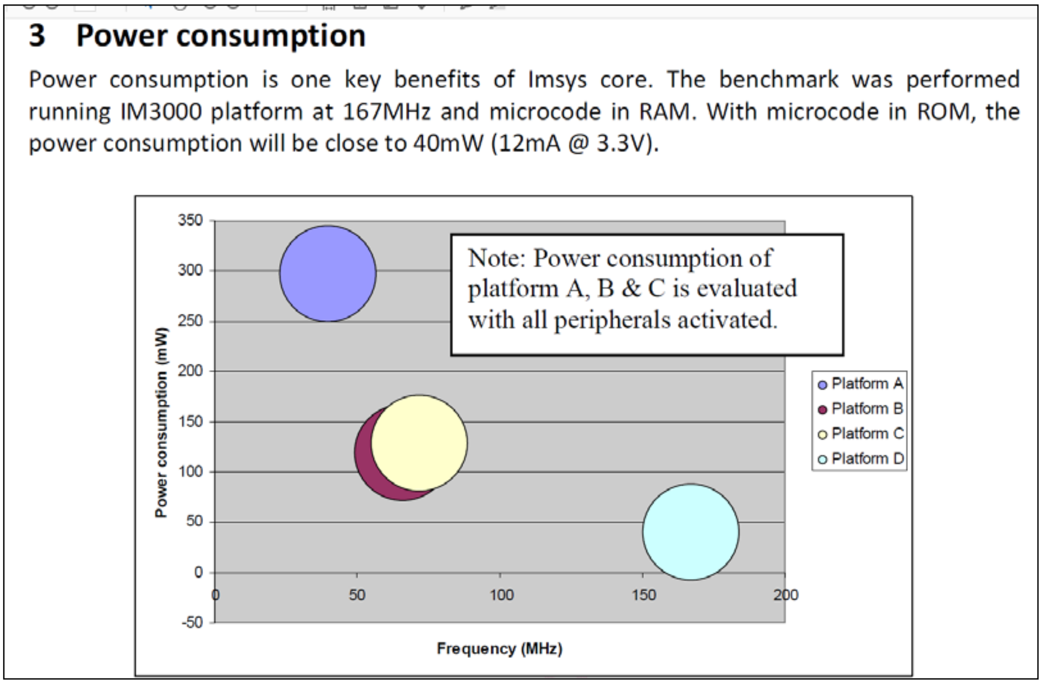
10. Energy consumption

The following two diagrams compare the measured current consumption of Imsys' 65 nm IC with that of ARM Cortex-M4 and Cortex-M0+, from datasheets, (with all I/O clocks turned off). For the Imsys chip, which has two cores, current was measured both with one and both cores active. Imsys data (for the 65 nm chip) has been scaled to 90 nm, which is the node used for the others. Log scale was used in the first diagram due to the big difference in consumption.





Below is an excerpt from an independent evaluation for a planned customer product in 180 nm. Platform A is ARM Cortex-M3 (STM32), Platform B is MIPS (Microchip PIC32) and Platform_C is the signal processor dsPIC from Microchip Technology. Platform D is Imsys IM3000, which has clearly the lowest power consumption, without having lower performance; in fact, it beat the others in most tests that were done.



3 Power consumption

Power consumption is one key benefits of Imsys core. The benchmark was performed running IM3000 platform at 167MHz and microcode in RAM. With microcode in ROM, the power consumption will be close to 40mW (12mA @ 3.3V).

Note: Power consumption of platform A, B & C is evaluated with all peripherals activated.

(Note about the “Note” in the figure: What is said is valid also for Platform D.)

11. Characteristics of the Imsys processors

General purpose processors (CPU, often of RISC type) and signal processors (DSP) normally have hardware cores that are tailormade and optimized for their respective instruction set architectures, and these are very different. The typical operation for the RISC is addition of 32-bit integers, while the typical operation for a DSP is multiplication of fractions, usually 16.bit, which is followed by addition of the product to an accumulation register. The RISC operates only on its registers and does load and store as separate instructions, while the DSP also works on memory and I/O and can step addresses of sources and destination, and do loop counting, while operating on data. A multiplier has more logic levels than an ALU, so the DSP has longer cycle time and doesn’t fit a RISC pipeline. Data formats are different, as mentioned, access patterns in memory are different. Furthermore, a CPU typically has von Neumann organization, while a DSP is a Harvard machine.

Thus, it is hardly possible to do hardware optimization for both kinds of processing.

Consider the following:

- Modern digital applications need, more than ever, both kinds of processing.
- Modern CMOS circuits are around 32 times faster than 20 years ago. That increase is an order of magnitude beyond what expensive hardware speed optimizations like deep pipeline, branch prediction, multiple issue, etc., can accomplish.
- Modern CMOS technology also offers around 1000 times higher ROM and RAM density than 20 years ago.

These three facts call for a different kind of processor:

1) In order to fit all relevant data formats, and utilize the high speed, the width of the basic data path can be reduced to 8 bits, the building block of all data processing and communication. This piece of hardware will be utilized for everything, and it can be efficiently used for any kind of processing since we can afford complex sequential control from an almost arbitrarily large control store. An extra bonus, for the manycore systems that will come, is that the power consumption for a working core is more or less constant, which simplifies power management.

2) CPU-style cache or the corresponding, but different, memory buffers used in DSPs, are not worthwhile since the main memory from now on, in typical embedded systems, is on the same chip as the processor and the speed gap therefore has been reduced, or almost eliminated.

3) Let the control store determine the basic machine cycle. Then the cycle will be long enough for an 8-bit DSP operation. DSP routines are relatively small, and the microarchitecture is Harvard, i.e., DSP routines should be microcoded.

4) For the CPU function, let microcode do instruction fetch, decode, and execute. The CPU is preferably of von Neumann type; it is in fact the simplest with this microarchitecture.

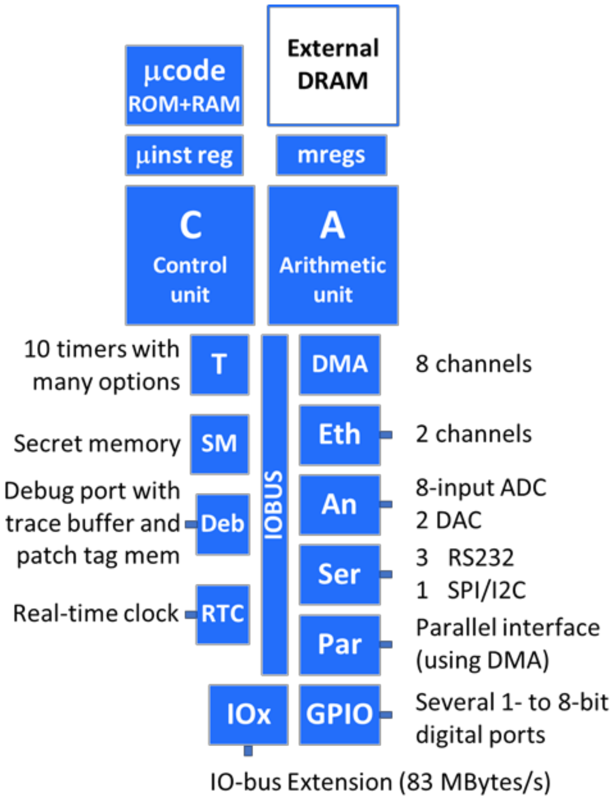
5) Let the microcode also do DMA, using the same hardware resources that are used for instruction execution. Let it also do other peripheral-oriented work, like display refresh and communication with the development system.

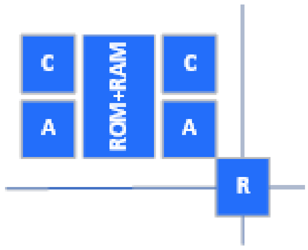
The Imsys processor is of this kind.

11.1. Block diagrams

The IM3000 Microprocessor is shown to the left. This IC also contains configuration registers, oscillator, PLL, voltage regulators, PwrOn reset, battery backup, etc. It boots itself from secondary storage, usually serial flash memory.

Virtual peripherals, implemented mostly in microcode rather than hardware, include autonomous processes for audio/video input/out, Ethernet MAC, crypto, and signal processing.



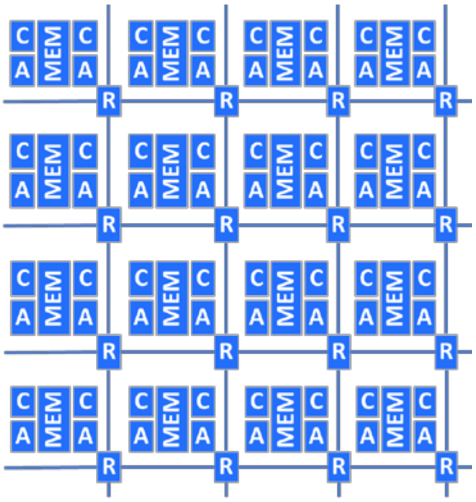


Above is a simplified block diagram of the 65 nm prototype. It has two cores, 256KB ROM, 160KB RAM and a NOC router. The total active area is 2.5 mm^2.

The 65 nm IC is a prototype for a “tile” of a many-core system

These tiles can be placed next to each other to form an array that will scale according to the table in Section3.

See also 11.6 below.



11.2. Optimized signal processing

This core can be seen as a DSP core with unusually wide instruction word. It also has an unusually narrow arithmetic part, which however has many degrees of freedom in its control, allowing it to work efficiently with many different data formats, from bytes up to the extremely wide words used by crypto algorithms.

Unlike the ARM cores shown in Section 4, the core does include debug, and it has what it needs for floating point (actually 3 times faster than the ARM920T) without spending area on any dedicated resources for it.

It also includes a byte-wide I/O channel, and with three small memory instances (not part of the basic core) it has a built-in system for full-speed tracing, a ROM patching block, and high speed DMA.

Other blocks that have been added in IM3000 include a timer system with many functions (it can be used for IEEE1588 timestamping), two 10/100 Mbps Ethernet channels (a very small block, since the Media Access Control layer is microcoded and no memory buffers are needed), and logic blocks for sigmadelta ADC and DAC, which together with the timer block can be used by interrupt driven microcoded algorithms.

Algorithms used for signal processing and security are normally well known and standardized, often available at the lowest possible level. They are small compared to general software, and developers are more concerned with their speed and energy efficiency than with their size or development time. The algorithms often use other data formats and access patterns than those common in general purpose software, and are often using I/O interfaces. For the Imsys core, the microcode level is the best to use for DSP and crypto algorithms. Imsys develops such algorithms on customer demand, and they are then stored together with the software for booting at startup (or when needed).

11.2.1. Scratchpad, etc.

Efficient signal processing requires the processor to be able to access two input operands and one output operand, and also change the pointers to all three, all in one machine cycle. For this purpose (and also others) the core has an internal scratchpad, with several associated pointer registers, in addition to several memory pointers that can address memory in “page mode”. It can also do loop counting and input or output a byte on the I/O channel in the same cycle.

11.2.2. Approximate calculations

The precision needed in signal processing is often lower than that used in traditional digital processors. There are of course traditional 8-bit processor architectures, but those are made for applications where performance doesn’t matter and they are not relevant for signal processing.

6-bit precision is enough to represent a full grayscale that the human eye sees as continuous, and digital telephony uses 8-bit signed floating-point representation with 3-bit exponent and only 4-bit mantissa. For neural networks high precision is needed for the learning process only; it is less important in a system using the learned “knowledge”, and for that system the cost, the speed and the energy consumption is typically very important.

The Imsys processor can execute narrow arithmetic faster and with less consumption than wider arithmetic, and it can efficiently work with logarithms to further reduce the logic activity and consumption.

11.3. High-level language software

If the core is to be used only as a DSP, then the microcode memory is its program memory, and an added RAM memory is used as data memory. It is thus a Harvard machine, like the ARM Cortex-M cores, and not suitable for big general-purpose software.

Also, compiling big programs to microcode is usually not a good idea because big programs need high code density to keep their memory requirement down. That can be achieved by defining high-level CPU instructions, optimized for the compiler rather than for the low-level hardware, and letting microcode in ROM fetch, decode, and execute them. The Harvard machine is then turned into a von Neumann machine, the instructions of which can be arbitrarily complex, e.g., contain loops, do several memory cycles, and use resources in the logic core (e.g., the scratchpad) that are not visible to the high-level software. Such a CISC machine (Complex Instruction Set Computer) can have instructions that correspond to what would be subroutines for a RISC machine (Reduced Instruction Set Computer). This not only saves silicon due to the smaller program size, it also saves energy since fewer instruction bytes need to be fetched, and since the execution can utilize the hidden hardware resources and do hardware operations in combinations that may be unique for the instruction type.

With this microcode the Imsys core becomes a complete CISC core, which can store compiled software into a memory and run it from there. The memory can, like the microprogram memory (also called control store) consist of both ROM (patchable) and RAM. The ROM part contains the boot loader code but can contain much more, in order to save energy and reduce startup time.

The IM3000 has interface for external dynamic RAM (SDRAM), but for the future the main memory for software and data will normally be on the same die as the processor, like it is in the fully verified 65 nm dual-core prototype. (In a system where this memory is not enough, external DDR memory can be used as secondary storage.)

Note that the compiled software in main memory resides in the same kind of RAM (and ROM) as the microcode. In the figure in Section 4, it can be seen that a program only 20 KB in size will take more of the expensive silicon area than the core logic does. This shows that code density, a characteristic of the instruction set architecture, is essential for the cost.

The extensive use of microprogramming is also essential for the energy consumption. The small logic core has less overhead activity (activity not directly requested by the program source code) because of its more flexible control, and the complexity in the use of this flexibility is in memory, which – compared to gate logic – consumes very little in relation to its size.

The 40 KB ROM shown in the figure is big enough for a rich instruction set. The size of that can vary depending on the application. IM3000, which has two resident instruction sets and a rich I/O system, has twice as much ROM, but that would normally not be needed in an ASIC design. A part of the microcode can reside in RAM instead of ROM, which is useful for DSP algorithms written directly in microcode. RAM has the same timing but higher consumption and 4-6 times bigger size for a given capacity. The RAM block shown here contains 20 KB.

Note that the IM3000 doesn’t have cache, even though its main memory is external. It has less need for cache since its instructions do more. The microcode becomes somewhat like an instruction cache, filled with commonly used subroutines. Also, the scratchpad reduces the need for a data cache – it is used for high-speed block read and write in memory, and when executing Java it contains a stack cache. Part of the scratchpad is also used as a stack for register sets. All this complex functionality is invisible to the programmer.

11.3.1. Executing Java

The machine reads JVM bytecode files without modification and executes them natively. Most bytecode instructions are executed directly by microcode like any other CPU instructions. (The least common bytecodes have not been microcoded and have instead been programmed in assembler.)

Imsys’ Java Virtual Machine is currently being updated to version 8. Several languages, e.g., Python, Ruby and Ada, can be compiled to JVM bytecode and executed by the Imsys processor more efficiently than using an interpreter.

11.3.2. Executing C and other languages

Assembly code and the C compiler utilize another ISA (Instruction Set Architecture), which coexists with the JVM ISA in the standard microprogram. This ISA will be replaced by a new ISA optimized for LLVM. Around 1100 instruction types have been specified and a few hundred have been microprogrammed and tested. The instructions are efficient encodings of the output from the intermediate level program representation in the LLVM compiler insfrastructure. Code on this LLVM level is called Bitcode, and is the input for the machine-dependent code generating part of the compiler. By meeting the compiler already at this level, executing Bitcode instructions as native instructions, Imsys avoids translation overhead activity, somewhat like when JVM bytecodes are executed in microcode rather than by a software interpretation program.

The LLVM compiler infrastructure is constantly improved by an active and growing community and will give Imsys access to free software, including important software development tools and operating systems.

11.4. Access to memory

Normally processors don’t know about memory. They just output a read or write command, together with an address of a fixed width, and then a word of a fixed width will be transferred between a processor register and the memory system. It usually takes more time than desired.

Only Imsys does it differently, in a patented way. Like everything else on a chip, memory is two-dimensional. It has rows and columns, and the columns are many more than the bits in a word – in fact they can be thousands. The microprogram can address a row before it knows the full address or whether it will be a read or write. The addressed row will then be accessible – until another row is selected – and any bytes on it can be read or written with short latency. The cycle time is even shorter, allowing the microcode to efficiently pipeline the transfers in parallel with other activity. They are accessed by column address only (just one byte). This is called page mode. The IM3000 can thus utilize the full bandwidth of its external DRAM. The same principle is used for the internal memory in the 65 nm processor chip, and used for both instructions and data.

11.5. Peripheral control

The Imsys processors have an 8-bit wide I/O channel that is directly controlled by the microcode, just as the arithmetic logic and the memory is. It is used both for the on-chip resources, e.g. timers and the analog block of IM3000, and for external peripherals. It can be used for transfer of command, status, and data bytes, e.g., for GPIO ports and timers, but also for DMA transfers. DMA can be used both for internal peripherals (Ethernet) and external, e.g., display and camera. DMA transfers use an internal buffer memory, such that the memory transfers for DMA will always use the full memory bandwidth, by using page mode (see above). The buffer memory is shared among the DMA channels in a configurable way.

11.5.1. Image input/output

Using the timers and the DMA function (Section 11.4) and microprogrammed memory access in page mode (Section 11.3), the system can efficiently input video from a camera or output video to a display. This is an autonomous microcode process, invisible to the software. Usually each pixel consists of three bytes, and they need to be packed in memory. This may be awkward for other processors but is no problem for the Imsys processor.

11.6. Self test and self monitoring

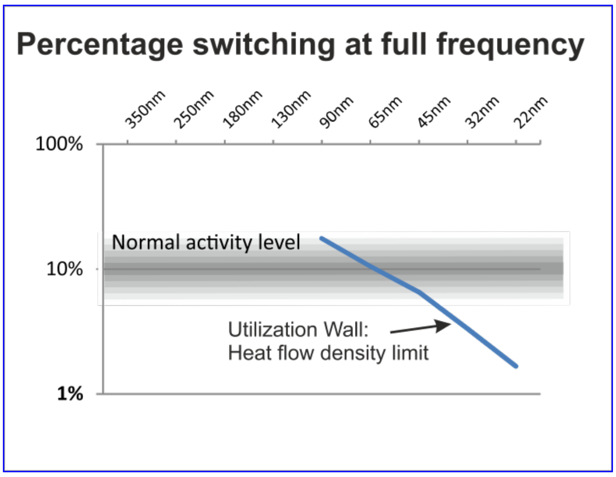
The flexibility of microprogramming and the many ways for the microcode to access and control the hardware logic is used also for test, e.g. at power-on, and can be used for monitoring the health of the system, not only the processor itself, and this is done faster and more thoroughly than would otherwise be possible.

11.7. Many-core ICs

Modern CMOS technologies cannot allow the use of both maximum density and maximum transistor switching frequency over the entire die area, because that would generate more heat than can be cooled off. An interesting consequence of leakage limited scaling is that the allowable utilization is cut in half for every new generation, as is seen at the bottom of the table in Section 3.

Thus, Moore’s law is changing – a physical limit of an insulating layer was reached at the 90 nm node, which had the result than the supply voltage could not continue to decrease as it had for several previous generation shifts, and this means that the heat density for a design that utilizes both maximum speed and density starts to double, rather that remain constant, for each new technology node.

From somewhere around 65 nm, the IC designs must therefore have low activity – i.e. contain memory rather than processor logic – in most of the die area. There are also other reasons for incorporating memory: Even though the cells of dedicated DRAM memory components are much less expensive than memory cells on a processor chip, an external memory requires many interconnections. The area required by bond pads doesn’t shrink, and it gets more expensive as the IC processing gets more advanced. (This is true also for the DRAM components themselves – they need many connections for address and parallel data and are therefore now made almost only with very high capacities – more than the average embedded system needs.)



When a processor gets its main memory onboard, next to the core, then most of the processor-to-memory speed gap – which has steadily grown since the eighties – disappears. This is why ARM Cortex-M processors don’t have cache memory as an integral part of the IP block, as the ARM9xx had. Cortex-M is made for MCU and SoC devices, with on-chip memory, and cache is then not worthwhile (except for some kind of I-cache or prefetch unit that is useful when executing from flash memory).

For a RISC core there is still a speed gap, however, since the basic RISC pipeline should be capable of higher clock frequency than the memory, and due to the non-mitigated speed gap the potential speed advantage of RISC-style pipelined operation may have over CISC is lost. A pipeline that is clocked at only a fraction of its maximum frequency cannot be an optimal design. It uses more hardware logic than needed – i.e. more area and energy than needed – and it doesn’t utilize what once motivated that overhead, i.e., higher frequency of ALU operations.

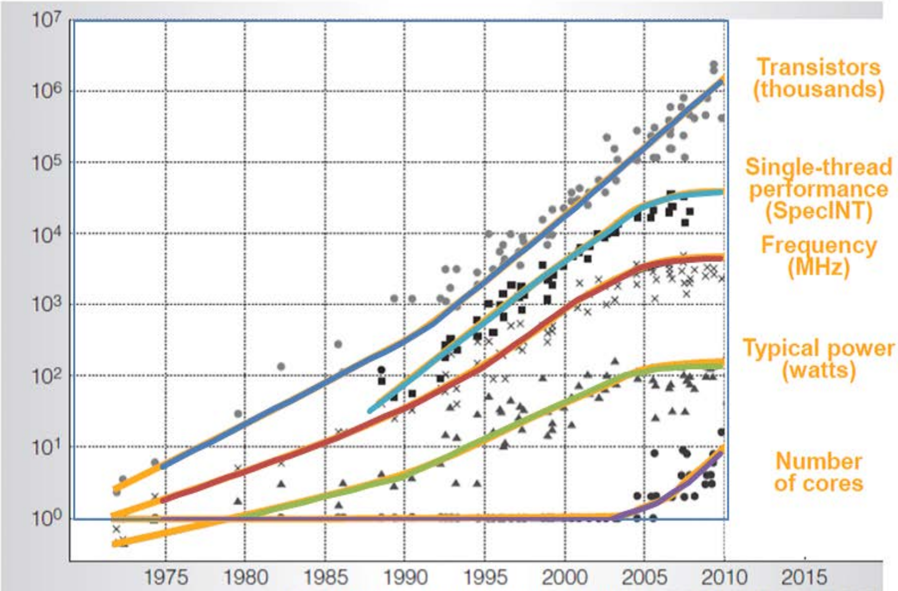
A CISC core with an advanced instruction set, where each instruction does more work by utilizing a small, flexible hardware core at high frequency for as many cycles as it needs, can be more efficient, i.e., it can do what has to be done with less overhead activity.

Such a core is a complex machine, but the complexity is in memory, which is much more dense and energy-efficient than the corresponding complexity in gate logic would have been.

It is therefore more in line with the trends of CMOS IC technology development.

As shown in the table of Section 3 and the diagram above, full utilization of both density and speed can for the modern technology nodes be allowed for only a portion of the die area, and that portion is rapidly approaching zero as the CMOS technology development continues. This is called the Utilization Wall, and it forces processor designers to maximize energy efficiency and increase the number of cores – at higher rate than the memory density increase, i.e., a doubling for each new node – as the only way to utilize Moore’s Law for increased throughput.

As illustrated in the diagram below, computer microprocessors were forced to go multicore around 2005, when chip power consumption could no longer be allowed to increase.



Published by the IEEE Computer Society 0272-1732/10/$26.00 © 2010 IEEE
C Kozyrakis, A Kansal, S Sankar, K Vaid, C Batten, M Horowitz, F Labonte, O Shacham, K Olukotun, L Hammond, C Batten

Further analysis of this diagram reveals that transistor count per core increased by around 500 times in the 15 years preceding the multicore era, but performance increased only around 3 times more than the clock frequency. This means that, from the architectural point of view (i.e. when scaling to the same technology node), energy efficiency of the design decreased enormously, just to get a rather modest performance increase. Typical embedded systems don’t need the inefficient speed-up logic that was added, but do need efficiency. This is the reason why the core of the new Intel Quark processors is, actually, a design from 1989 (see Microprocessor Report, Jan. 11 2016). Furthermore, many-core designs need the cores to be small, and the code to be dense.

The Imsys processor fits very well for manycore ICs. A tile for an Imsys-based manycore chip has been designed in cooperation with KTH and prototyped in 65 nm low leakage technology. It has 2.5 mm2 active area, which contains a dual core, 336 KByte ROM, 200 KByte RAM, and a NOC (Network on Chip) router. Its measured power consumption is 31 microwatts per MHz per core.

97% of the 13 million transistors in the tile are used in memory blocks, and the total activity level is therefore only on the order of 1%. This is far down in the diagram above (“Percentage switching,…”), which allows us to go far right, beyond the 22 nm mark, without hitting the Utilization Wall. This is essential; the competing processors such as the ARM Cortex-M series and MIPS have much more active logic and cannot be used efficiently for a chip with thousands of cores, as the Imsys core can.

A 320 mm2 area on a chip in 65 nm could have 128 tiles of this silicon-verified design, i.e. 256 cores, and would, at full activity, consume 11 mW per core, at 350 MHz. The total current would be 2.3 A, and since the voltage is 1.2V, the total power would be 2.8 W. This can be scaled to finer technology. It is, however, no longer possible to scale down the voltage as was done in older generations. The following table shows what can be achieved in finer technologies assuming voltage reduction stops at 1.0 V (which is conservative):

Node	Cores	Freq(MHz)	ROM+RAM(MByte)	Power(W)	Dimensions(mm)	Area(mm^2)
65 nm	256	350	42+25	2.8	12.6 x 25.3	320
45 nm	512	506	84+50	3.9	17.5 x 17,5	307
32 nm	1024	711	168+100	7.8	12.5 x 24.9	310
22 nm	2048	1034	336+200	15.6	17.1 x 17.1	293
14 nm	4096	1625	672+400	31.3	10.9 x 21.8	238

The power consumed by the NOC (network on chip) is application dependent and is not included here. However, a practical limit for power consumption is on the order of 80 W, and the calculated consumption of the core array is far from that limit.

The reasoning above assumes that the existing, prototyped and verified, logic design of the tile is used, unchanged. Some low risk enhancements are planned, which will further increase performance of important functions without reducing energy efficiency. Memory capacity per tile can of course be chosen differently.

This type of manycore processor may be suitable for “fog computing” and “edge computing” in the Internet of Things. Each of the cores would be software compatible with the IM3000, and be able to use its Java platform and all the microcoded “accelerator” functions. Compared to IM3000, the maximum total throughput would be almost 40 thousand times higher for the 14-nm chip.

Each tile has its most fundamental firmware (OS kernel, library routines) in local ROM. Some of the cores, typically at the edge, will be specialists, e.g. on networking, and will have e.g. crypto and Ethernet MAC in ROM (like IM3000 has). Others may have accelerator hardware connected and therefore have driver firmware for those in ROM. Each tile also has RAM for microcode, which lets any core download special skills, both in software and microcode. The cores, communicating over the NOC, which is capable of both wormhole and circuit switching, can be organized as needed, i.e. hierarchically with supervisors, specialists, and workers having different firmware profiles. They will all have their own partition of the total memory on the chip, but some may have better access to a larger on-chip memory.