Special Session on Low-Power Systems on Chips (SOCs)

Organiser: Christian Piguet[1]
Presenters: Christian Piguet[1], Marc Renaudin[2] and Thierry J-F. Omnès[3]

[1]CSEM, Rue Jaquet-Droz 1, P.O. Box **CH**-2007 Neuchâtel
[2]Laboratoire TIMA, 46 Avenue Félix Viallet, **F**-38031 Grenoble Cédex
[3]Interuniversity Micro-Electronics Centre (IMEC), Kapeldreef 75, **B**-3001 Leuven

Abstract

Low-power Issues for SoCs *by Christian Piguet, CSEM*[1].
For innovative portable products, Systems on Chips (SoCs) containing several processors, memories and specialised modules are obviously required. Performances but also low-power are main issues in the design of such SoCs. Are these low-power SoCs only constructed with low-power processors, memories and logic blocks? If the latter are unavoidable, many other issues are quite important for low-power SoCs, such as the way to synchronise the communications between processors as well as test procedures, online testing, software design and development tools. This paper is a general framework for the design of low-power SoCs, starting from the system level to the architecture level, assuming that the SoC is mainly based on the re-use of low-power processors, memories and logic peripherals.

Asynchronous and Locally Synchronous low-power SoCs *by M. Renaudin, TIMA*[2] *and C. Piguet, CSEM*[1].
SoCs with many processors, co-processors, memories and peripherals cannot be synchronised with a single master clock, due to larger and larger wire delays in deep submicron technologies. Several clocking schemes have been proposed, such as GALS (Globally Asynchronous Locally Synchronous) but also full asynchronous architectures. This paper will present the advantages and disadvantages of these SoC clocking strategies as well as the impacts on low-power.

Low-power Software for SoCs *by T. Omnès, C. Kulkarni, K. Danckaert and E. Brockmeyer, IMEC*[3].
For embedded SoCs containing several processors, one has to write several pieces of software for each processor starting typically from a high-level specification using the C/C++ language. In order to tackle this problem, we propose to first transform the original specification by means of a systematic script of platform-independent source code transformations. That is illustrated by applying global loop transformation techniques to identify asynchronous partitions exhibiting little communication and high locality of access characteristics. In a second stage, we explore multiple–instruction multiple–data (MIMD) mapping onto a given (partly) predefined platform using advanced space–time analysis techniques to maintain low data transfer rates while achieving high system throughput. At the SoC level, accurate cost feedback including high-level power estimation is required. From this essential information, energy trade-offs between application sub-modules can for example be used to refine the solution further. In the case of mapping onto programmable cores with a shared memory hierarchy, a final refinement consists in reorganising the data layout for efficient cache utilisation.

1 Low-power issues for SOCs by Christian Piguet, CSEM[1].

1.1 Introduction

According to the SIA 1999-2000 Roadmap, the move toward embedded Systems on Chip is quite clear. Systems on Chip (SoCs) consist to integrate several components on the same chip in order to improve performances and reduce the cost. Few years ago, a system was a multichip device containing a microprocessor, several memory chips, DRAM chips, FPGA chips, RF (BiCMOS) or analog chips. More and more, all these components will be integrated on a single chip, including microprocessors, glue logic, memories, ROM and SRAM, conventional analog blocks, DRAM memories, FPGA-like logic, RF receiver or transmitter and finally MEMS. The total number of transistors on a single chip could be over one billion (predicted [27] to be between 4 to 19 billions in 2014 depending in the circuit type).

For SoCs, very important design problems have to be solved. They are mainly the silicon complexity (reliability,

power, interconnect), the system complexity (logic, MPU, memories, RF, FPGA, RF, MEMS), the design procedures (300 to 800 people for the design of a single chip, I.P re-use, design levels), verification and test. Some partial answers have been given to the complexity problem, such as design re-use. The SIA Roadmap [27] predicts that in 2012 reuse of processors, logic blocks, and peripherals, would reach about 90% of the embedded logic on the chip.

1.2 Low-power Design Methodologies and CAD tools

Future SoCs will contain several different processor cores on a single chip. It results in parallel architectures, which are known to be less power hungry than fully sequential architectures based on a single processor [24]. The design of such architectures has to start with very high-level models in languages such as System C, DL or MATLAB. The very difficult task is then to translate such very high-level models in application software in C and in RTL languages (VHDL, Verilog) to be able to implement the system on several processors.

One could think that many tasks running on many processors require a multitask but centralised operating system (OS), but regarding low-power, it would be better to have tiny OS (2K or 4K instructions) for each processor [16], assuming that each processor executes several tasks. Obviously, this solution is easier as each processor is different even if performances could be reduced due to the inactivity of a processor that has nothing to do at a given time frame.

Each more or less specialised processor will be programmed in C and will execute after compilation its own code. Low-power software techniques have to be applied to each piece of software, including pruning, in-lining, loop unrolling and so on. For re-configurable process or cores, retargetable compilers have to be available. The parallel execution of all these tasks has to be synchronised through communication links between processors and peripherals. It results that the co-simulation development tools have to deal with several pieces of software running on different processors and communicating between each other. Such a tool has to provide a high-level power estimation tool to check which are the power hungry processors, memories or peripherals as well as the power hungry software routines or loops.

Such a tool is far from being commercially available. Embedded low-power software emerges as a key design problem. The software content of SoC will increase as well as the cost of its development.

1.3 Low-power SoC Architectures

Low-power SoCs will be based on low-power components, such as processor cores, memories and libraries that are available with nice performances, *i.e.* 20'000 to 100'000 MIPS/watt [1] for some cores (using LP techniques such as gated clocks). However, memories are the main consumers on a SoC and several techniques at the architecture and electrical levels have to be applied to reduce their power consumption (generally based on caches, DWL, bitline splitting, low swing).

However, low-power architectures have also to be used. As mentioned earlier, parallel architectures with various specialised cores [24] exploit a natural parallelism, which is not based on processor arrays for which software parallelising is too difficult and for which the communication cost is too high. Furthermore, the use of specialised and reconfigurable cores could improve performances in such a way that supply voltage could be reduced resulting in lower power consumption. Using a very dedicated co-processor to a given task could improve the speed/power performances of several orders of magnitude.

The 1997 SIA Roadmap [27] recognises that, in 2007, asynchronous design will be used in many designs. If ∂ is the distance travelled by a signal in one clock cycle, due to higher frequencies and increasing interconnect delays, a chip will contain several time zones of sizes $\partial * \partial$. Due also to the increasing die size, this number of time zones will grow very rapidly with the new technologies, up to 10'000 zones in 2012. To synchronise 10'000 time zones is a true asynchronous problem [27]. It is why asynchronous design is strongly required for chip architectures in the future [25].

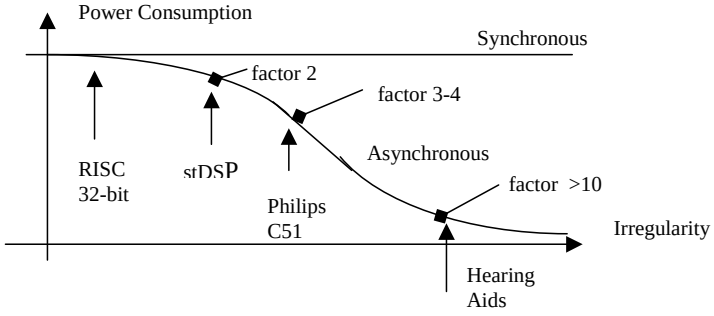

Figure 1. Comparison of the Power Consumption of Asynchronous versus Synchronous Architectures

Asynchronous architectures are often presented as being capable of reducing significantly the power consumption [25]. Looking at the results from many papers, it turns out that the largest power savings are obtained with circuits or applications that present a very irregular behaviour. An ir-

regular behaviour is, for instance, a processor that presents a quite tricky instruction set with multi-bytes instructions executed in a various number of cycles and phases. Or an application for which the controller has to very often stop and restart depending on the application. A very regular behaviour is a 32-bit RISC core for which all instructions are always executed in one clock. Figure 1 illustrates this basic law. Basically, SoCs will present more an irregular behaviour than a regular one.

At the architecture level, power and Vdd management with behaviour prediction of the user will be used extensively, as well as low-power communication protocols between the various processors on a single chip. These protocols have to be kept simple and will be asynchronous due to the fact that the various cores will be clocked (if not asynchronous cores) with many different frequencies.

1.4 Low-power Design and Testability Issue

At the low-level, low-power libraries and logic synthesis embedded with place and route are required, as well as estimation of interconnect delays with copper, low-k and SOI. However, the main issue is Vdd as low as 0.6 to 0.3 Volt in 2014. With very deep sub-micron technologies and very low Vdd, the static power will increase significantly due to low Vt. Several techniques with double Vt, source impedance, well polarisation, dynamic regulation of Vt are today under investigation and will be necessarily used in the future.

SoCs testability and debug when first silicon has been returned from the foundry are a main issue. Generally, the mean time to fix a bug is one week. Today, it is not possible do determine if more bugs will be present in a one billion transistor chip, and if a one week per bug is realistic or not. However, it has to be mentioned that it should be much more difficult to fix a bug in I.P. blocks that you do not have designed than current practise. In this new design context, it is estimated that half of the total design effort will be devoted to verification tasks, including debug of the embedded software [27].

2 Asynchronous and Locally Synchronous low-power SOCs by Marc Renaudin, TIMA[2] and Chritian Piguet, CSEM[1].

2.1 Introduction

SoCs chips will contain with many processors, co-processors, memories and peripherals. As it is quite difficult today to design a clock tree for a single processor (clock skew, power consumption), one can be easily convinced that the situation will be worse in a few years, due to interconnect delays, many processors and different frequencies. It results that future SoCs cannot be synchronised with a single master clock. It is therefore mandatory to propose other clocking schemes including fully asynchronous architectures.

2.2 Clocking schemes

One has fundamentally the choice between two clocking schemes:

- GALS (Globally Asynchronous Locally Synchronous)
- fully asynchronous architectures.

The first choice results from the evolution of synchronous architectures. Basic building blocks are synchronous processors and peripherals, but due to the clock distribution problem, the communications between processors are performed asynchronously. The second choice is a more radical choice that selects asynchronous cores as well as asynchronous communications. From the conceptual point of view, this solution is better as it is a coherent technique and not a mix of synchronous/asynchronous techniques as proposed by the first solution.

2.3 Globally Asynchronous Locally Synchronous SoCs

The high-level model of GALS is many processor cores that can be clocked at any and possibly variable frequency. There can also be asynchronous logic blocks on the same SoC. These cores and blocks have to communicate through asynchronous interfaces. There is no relationship between the frequency of a given block that has to send some messages to any other (or others) block(s). There are two basic solutions to allow the blocks to communicate properly:

- to synchronise their clocks by adjusting the PLL of the two (or more) blocks. This solution proposed in [15] consists in N*N PLL for N*N time zones, two adjacent PLL being synchronised by PFDs (Phase Frequency Detectors) in case of a communication.
- to perform an asynchronous communication for which meta-stability has to be avoided. One has to check the data and clock timing to avoid collisions, by introducing a clock delay (stretchable or pausable clock). Several registers clocked by delayed clocks generated locally avoid metastability. Such a technique works well for two communicating blocks, but is not suited for communicating from one to several blocks which could result in a clock-data collision for the chosen delayed clock. It is why delaying data instead of the clock is better [14], but it increases hardware.

2.4 Fully Asynchronous SoCs

A more drastic change is to adopt asynchronous logic by default for designing all the components of a SoC and use the ”clock” as a resource only when required. Such an approach requires to master asynchronous circuit design, which is not yet an easy task, but on the other hand brings very interesting facilities that the goal of this session is to discuss. It is shown that asynchronous blocks are very easy to reuse and easy to integrate in a complex heterogeneous integrated system because they are locally controlled, they provide flexible and efficient interfacing mechanisms, they have a lower mean power consumption while providing maximum performance, and finally they generate lower electromagnetic noise and smaller currents peaks in the supply [25] [20]. Significant prototypes designed by industrial and academic teams are reported and discussed to illustrate these properties.

2.5 Power consumption issues in GALS and fully asynchronous SoCs

Comparison:

- GALS: less area than fully asynchronous, due to larger area of fully asynchronous cores
- GALS: more power than fully asynchronous, PLL are large consumers, delayed clocks or data generally require adjustable delays to take into account process variations, temperature,..., but these delays require consuming hardware. Furthermore, for irregular behaviour, fully asynchronous cores consume less than synchronous cores. And SoCs will be mainly designed for applications with irregular behaviour.
- GALA SoCs are by construction consuming minimum energy, their power consumption may be automatically and dynamically adapted to the processing power required by the data stream and they do not require specific hardware/software layers to turn-off inactive processing parts or peripherals.

3 Low-power software for SOCs by T. Omnès, C. Kulkarni, K. Danckaert and E. Brockmeyer, IMEC[3].

For embedded SoCs containing several processors, one has to write several pieces of software for each processor starting typically from a high-level specification using the C/C++ language. Because the cost-critical part of this high-level specification is the data-dominated part [28], we have developed a systematic script of code transformations known as the Data Transfer and Storage Exploration (DTSE) methodology for power, area, price and speed multi-objective data-dominated embedded software design [7]. The most tedious and error–prone parts of the DTSE methodology are being supported by IMEC’s ACROPOLIS/ATOMIUM computer-aided design (CAD) environment [22] [13]. In this paper, we discuss the main steps of DTSE involved in producing power vs. speed software for each processor of a programmable platform [9] consisting of several processors and offering memory and interconnect reconfiguration opportunities [29] in the ideal case.

3.1 Data-locality improving Loop Transformations

Loop transformations are at the heart of DTSE. As one of the first steps (after preprocessing and pruning) in the script, they are able to significantly reduce the required amount of storage and transfers. As such however, they only increase the locality and regularity of the code. This enables later steps in the script (notably the data reuse, memory (hierarchy) assignment and in-place mapping steps) to arrive at the desired reduction of storage and transfers.

Crucial in our methodology is that the code transformations have to be applied globally, *i.e.* with the entire algorithm as scope. This is in contrast with most existing loop transformation research, where the scope is limited to one procedure or even one loop nest. This can enhance the locality within that loop nest, but it does not solve the global data flow and associated buffer space needed between the loop nests or procedures. In order to allow these global transformations, the step has to be applied before any partitioning decision is taken *w.r.t.* the HW/SW-system being designed [12].

To perform global loop transformations, we make use of a methodology based on the polytope model [19, 30]. In this model, each n-level loop nest is represented geometrically by an n-dimensional polytope. The order in which the iterations are executed can be represented by an ordering vector which traverses the polytope. To perform the transformations, we have developed a two-phase approach. In the first phase, all polytopes are placed in one common iteration space. During this phase, the polytopes are merely considered as geometrical objects, without execution semantics. In the second phase, a global ordering vector is defined in this global iteration space.

3.2 Logically Shared but Physically Distributed Data Mapping

After data-locality improving loop transformations have been applied, the cost of a data-dominated Logically Shared but Physically Distributed (LSPD) memory system is determined mostly by:

- The inter-processor bandwidth, which is given by the maximum sum of all ingoing and outgoing dependency edges from each partition;
- The intra-processor bandwidth, which is given by global basic group level conflict analysis at the processor-level [33].

The conventional approach used by automatic parallelisation compilers is to partition the iteration space in a Single Program Multiple Data (SPMD) fashion, as it appears after data-locality improving loop transformations. This often results in a high intra-processor bandwidth requirement by the principle of pushing the bottleneck one abstraction-level further down. To alleviate this situation where parallelisation is performed regardless of processor-level constraints and processor-level optimisation is performed only locally we believe that *space–time* analysis of (partly) unrolled loop bodies can help in:

- Reducing the intra-processor bandwidth requirements with limited impact on data locality by re-mapping data accesses onto processors using the full Multiple Instruction Multiple Data (MIMD) capabilities of modern distributed systems [32] [10];
- Discovering low-cost *space–time* computation *patterns*, that can serve as the basis for (very) low-cost distributed object architectures in platform-based design.

Using our proposed *pattern* analysis, we show (at least) an overall time $\times$ consumption (C $\times$ T) improvement above 100% compared to a custom shared memory approach on a small C/C++ platform-based design example [23]. By construction, this technique supports the *interactive* embedded software design [21] (even) of large-scale and parallel multimedia and is being integrated within our prototype ACROPOLIS environment.

3.3 Cycle Budget vs. Power trade-offs

In contrast to current design practise for (programmable) processor mapping, which mainly targets performance, we focus on a systematic trade-off between cycle budget and energy consumed in the background memory organisation. The latter is a crucial component in many of today's designs, including multi-media, network protocols and telecom signal processing. We have a systematic way and tool to explore both freedoms and to arrive at Pareto charts, in which for a given application the lowest cost implementation of the memory organisation is plotted against the available cycle budget per sub-module. This by making optimal usage of a parallelised memory architecture. We indicate, with results on a digital audio broadcasting receiver and an image compression demonstrator, how to effectively use the Pareto plot to gain significantly in overall system energy consumption within the global real-time constraints.

The new prototype tool has been applied to drivers from multiple application domains to prove the effectiveness, as demonstrated by results in other recent papers [2, 5, 3, 31]. The actual technique underlying the SCBD exploration is discussed in [4]. Here we will analyse the results and the detailed evolution of the SCBD tool for the Binary Tree Predictive Coder driver only. The experiment clearly shows the trade-off between memory organisation cost (power and area) and the memory subsystem cycle budget. All the results are obtained within reasonable tool execution time (several minutes) on a Pentium II-400. A Motorola library memory model is used to estimate the on-chip memory cost (see [8]). For the off-chip components, we have used an EDO DRAM series of Siemens.

Binary Tree Predictive Coding (BTPC) [26] is a loss-less or lossy image compression algorithm based on multi resolution. The image is successively split into a high resolution image and a low resolution quarter image, where the low-resolution image is split up further. The pixels in the high-resolution image are predicted based on patterns in the neighbouring pixels. The remaining error is then expected to achieve high compression ratios with a Huffman coder. The power numbers in this section are based on real memory models.

Figure 2 shows the trade-off for the complete cycle budget range. This is obtained by letting the tool explore the cycle budget starting from the fully sequential budget, and then progress through the most interesting memory organisations (from a cost point of view) to reduce the cycle budget for multiple differently optimised implementations [3]. The number of allocated on chip memories is four for the entire graph shown in this figure. In the sequential budget (about 18M cycles), only single-port memories are employed. When a dual-ported memory has to be added, a clear discontinuity is present in the energy function. In order to reduce the budget below 8M cycles, dual-ported memories are needed though. The allocation of two dual-ported memories allows to decrease the cycle budget up to the critical path (6.5M cycles). The worst case needed "image" bandwidth can be guaranteed by inserting three intermediate on-chip memories (two dual port and one single

port). These three intermediate memories can deliver up to five pixels per memory cycle without the cost (or infeasibility) of a five-port memory.

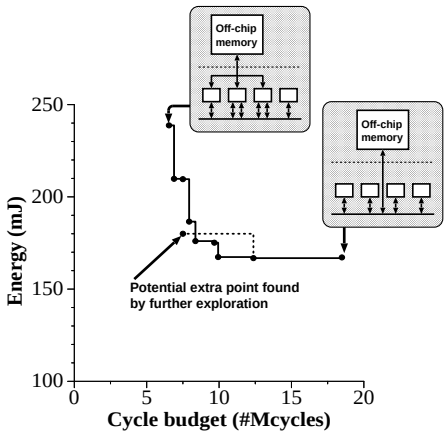

Figure 2. Pareto curve for Binary Tree Predictive Coder.

The memory subsystem cost can be traded of for costs in the other subsystems. The global system cost in terms of power/energy consumption, but also chip/board area or dollars, can be significantly reduced at the price of an acceptable increase in cycles spent in a particular sub-module at the cost of less cycles for another less cost-sensitive submodule. Exploiting this trade-off is only feasible for real complex systems if it can be systematically explored and when the useful trade-off space is effectively visualised so that a designer can quickly evaluate the cost of different cycle budgets for a given sub-module. We have a methodology supported by tools to effectively achieve this. It has been demonstrated for several other real-life applications (from different domains) like DAB channel decoder, cavity detector and an IP-switch.

3.4 Cache conscious Main Memory Data Layout

Cache misses form a major bottleneck for real-time multimedia applications due to the off-chip accesses to the main memory. This results in both a major access bandwidth overhead and related power consumption as well as performance penalties.

In this section, we illustrate a new technique for organising data in the main memory for data dominated multimedia applications so as to reduce the majority of the conflict cache misses. The main goal of this step is to statically organise data in the main memory taking into account the cache parameters and program characteristics. This involves a combination of array splitting and array interleaving (or array merging) so as to recursively allocate data in the main memory, where each recursive size is equal to the cache size [18]. The main advantage of this technique is the ability to trade-off memory size versus power versus performance as compared to other existing techniques. A more detailed discussion of this step is available in [17]. This step has been automated as a prototype tool in the ACROPOLIS project at IMEC.

Experiments on real-life demonstrators illustrate that we are able to reduce up to 82% of the conflict misses for applications which are already aggressively transformed at source-level. At the same time, we also reduce the off-chip data accesses by up to 78%. Thus our approach is complimentary to the more conventional way of reducing misses by reorganising the execution order.

3.5 Conclusion

For embedded SoCs containing several processors, designing efficient software for each processor is a (very) difficult and (very) time-consuming task that can be performed manually by a trained designer (only) at the price of a (very) significant degradation of the quality of his final implementation. Embedded software design tools that could help in this process are far from commercially available today. In order to tackle this problem for data-dominated applications, we propose the Data Transfer and Storage Exploration (DTSE) methodology to systematise the organisation of the designer work. In addition, we show that novel CAD support consisting of four main steps can be introduced to automate (almost) all the critical work involved in this multi-objective application-specific platform mapping task **before** applying conventional hardware-software co-design [6] [11]. This support is currently exactly the purpose of IMEC's ACROPOLIS/ATOMIUM environment [22] [13].

References

[1] C. Arm, J.-M. Masgonty, and C. Piguet. Double-latch clocking scheme for low-power I.P. cores. In *Proc. PATMOS'2000*, Goettingen, Germany, September 13–15, 2000.

[2] E. Brockmeyer, J. D'Eer, N. Busà, F. Catthoor, P. Lippens, and J. Huiskens. Code transformations for reduced data transfer and storage in low power realization of DAB synchro core. In *Proc. PATMOS'99*, Kos, Greece, 1999.

[3] E. Brockmeyer, A. Vandecappelle, and F. Catthoor. Systematic cycle budget versus system power trade-off: a new perspective on system exploration of real-time data-dominated applications. In *Int. Symp. on Low Power Electronics and Design (ISLPED)*, pages 137–142, Rapallo, Italy, July 26–27, 2000.

[4] E. Brockmeyer, A. Vandecappelle, S. Wuytack, and F. Catthoor. Low power storage cycle budget distribution tool support for hierarchical graphs. In *13th Int. Symp. on System Synthesis (ISSS)*, pages 137–142, Madrid, Spain, Sept. 20–22, 2000.

[5] E. Brockmeyer, S. Wuytack, A. Vandecappelle, and F. Catthoor. Low power storage for hierarchical graphs. In *Proc. 3rd ACM/IEEE Design and Test in Europe Conf. (DATE)*, Paris, France, Mar. 2000.

[6] Cadence Design Systems. Cierto virtual component co-design (VCC) environment. `http://www.cadence.com/datasheets/vcc_environment.html`, 2000.

[7] F. Catthoor, K. Danckaert, C. Kulkarni, and T. J.-F. Omnès. Data transfer and storage architecture issues and exploration in modern DSPs. In Y. H. Yu, editor, *Programmable Digital Signal Processors: Architecture, Programming, and Applications*. Marcel Dekker, Inc., New York, USA, Dec. 2000.

[8] F. Catthoor, S. Wuytack, E. de Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. *Custom Memory Management Methodology - Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, ISBN 3-540-64105-X, 1998.

[9] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd. *Surviving the SOC Revolution : A Guide to Platform-Based Design*. Kluwer Academic Publishers, ISBN 0-7923-8679-5, Nov. 1999.

[10] G. Coulson and S. Baichoo. Experiences in implementing a distributed object platform for multimedia applications. *Software Practice and Experience*, 30(6):663–683, May 2000.

[11] Coware. Coware N2C design system overview. `http://www.coware.com/`, 2000.

[12] K. Danckaert, K. Masselos, F. Catthoor, H. De Man, and C. Goutis. Strategy for power efficient design of parallel systems. *IEEE Trans. on VLSI Systems*, 7(2):258–265, June 1999.

[13] K. Denolf, P. Vos, J. Bormans, and I. Bolsens. Cost-efficient C-Level Design of an MPEG-4 Video Decoder. In *Proc. Int. Wsh. on Power and Timing Modeling, Optimization and Simulation*, Goettingen, Germany, Sept. 13-15, 2000.

[14] R. Ginosar et al. Adaptive synchronization. In *Proc. AINT 2000*, Delft, Netherlands, July 18–20 2000.

[15] V. Gutnik and A. Chandrakasan et al. Active GHz clock network using distributed PLLs. In *Proc. IEEE Int. Solid-state Circuits Conf. (ISSCC)*, San Francisco, USA, Feb. 2000.

[16] A. Jerraya. Hardware/software codesign. In *Summer Course*, Orebro, Sweden, August 14–16, 2000.

[17] C. Kulkarni. *Cache optimization for multimedia applications*. PhD thesis, ESAT, EE Department, Katholieke Universiteit, Leuven, Belgium, Feb. 2001.

[18] C. Kulkarni, C. Ghez, M. Miranda, F. Catthoor, and H. de Man. Cache conscious data layout organization for embedded multimedia applications. In *Proc. 4th ACM/IEEE Design Automation and Test in Europe Conference (DATE)*, Munich, Germany, Mar. 2001.

[19] C. Lengauer. Loop parallelization in the polytope model. In *Proc. 4th Int. Conf. on Concurrency Theory (CONCUR)*, Hildesheim, Germany, Aug. 1993.

[20] S. M. Nowick, M. B. Josepphs, and C. H. V. Berkel. *Special Issue of the Proceedings of the IEEE on "Asynchronous Circuits and Systems"*. Feb. 1999.

[21] T. Omnès, T. Franzetti, and F. Catthoor. Interactive algorithms for minimizing memory bandwidth in high throughput telecom and multimedia. In *Proc. 37th ACM/IEEE Design Automation Conf. (DAC)*, pages 328–331, Los Angeles, USA, June 2000.

[22] T. J.-F. Omnès. Acropolis*: a System-Level Precompiler for Data Transfer and Storage Exploration (DTSE) in High-Throughput Embedded System Design*. Ph.D dissertation in Real-time Systems, Automation and Robotics, Centre de Recherche en Informatique de l'Ecole Nationale Supérieure des Mines, Paris, France, May 2001.

[23] T. J.-F. Omnès. Space–time memory access *patterns*: a step towards the object-oriented design of low-cost distributed platforms. In *5th Int. Wsh. on Software and COmPilers for Embedded Systems (SCOPES)*, St Goar, Germany, March 20-22, 2001.

[24] C. Piguet. Parallelism and low-power. In *Proc. Symp. Architectures de Machines (SymA'99)*, Rennes, France, June 8, 1999.

[25] M. Renaudin. Asynchronous circuits and systems : a promising design alternative. In *MIGAS Summer School on Microelectronics for Telecommunications : Managing High Complexity and Mobility*, Autrans, France, June 28 – July 4, 2000.

[26] J. Robinson. Efficient general-purpose image compression with binary tree predictive coding. *IEEE Trans. on Image Processing*, 6(4):601–608, Apr. 1997.

[27] Sematech. SIA rodmap. `http://notes.sematech.org/ntrs/rdmpmem.nsf`, 1997.

[28] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power optimisation. In *Proc. 31th ACM/IEEE Design Automation Conf. (DAC)*, pages 384–390, Santa Clara, CA, Nov. 1994.

[29] C. Ussery. Configurable Platforms: The ASIC Revolution. In *Invited talk, 37th ACM/IEEE Design Automation Conf. (DAC)*, Los Angeles, USA, June 2000.

[30] M. van Swaaij, F. Franssen, F. Catthoor, and H. de Man. Modeling data and control flow for high-level memory management. In *Proc. 3rd ACM/IEEE European Design Automation Conf. (EDAC)*, pages 8–13, Brussels, Belgium, Mar. 1992.

[31] A. Vandecappelle, M. Miranda, E. Brockmeyer, F. Catthoor, and D. Verkest. Global multimedia system design exploration using accurate memory organization feedback. In *36th IEEE/ACM Design Automation Conf. (DAC)*, pages 327–332, New Orleans, LA, June 1999.

[32] W. Wolf. Object-oriented cosynthesis of distributed embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 1(3):301–314, July 1996.

[33] S. Wuytack, F. Catthoor, G. de Jong, B. Lin, and H. de Man. Flow graph balancing for minimizing the required memory bandwidth. In *Proc. 9th ACM/IEEE International Symposium on System-Level Synthesis (ISSS)*, pages 127–132, La Jolla, CA, Nov. 1996.