Architectures and Design techniques for energy efficient embedded DSP and multimedia processing

Ingrid Verbauwhede[1,2], Patrick Schaumont[1], Christian Piguet[3], Bart Kienhuis[4]

UCLA[1], K.U.Leuven[2], CSEM[3], Leiden[4]

ingrid@ee.ucla.edu, schaum@ee.ucla.edu, christian.piguet@csem.ch, kienhuis@liacs.nl,

Abstract

Energy efficient embedded systems consist of a heterogeneous collection of very specific building blocks, connected together by a complex network of many dedicated busses and interconnect options. The trend to merge multiple functions into one device makes the design and integration of these "systems-on-chip" (SOC's) even more problematic. Yet, specifications and applications are never fixed and require the embedded units to be programmable.

The topic of this paper is to give the designer architectures and design techniques to find the right balance between energy efficiency and flexibility. The key is to include programmability (or reconfiguration) at the right level of abstraction and tuned to the application domain. The challenge is to provide an exploration and programming environment for this heterogeneous architecture platform.

1 Introduction

Embedded systems (e.g. a cell phone, a GPS receiver, a portable DVD player, a HDD camcorder) use an architecture that is a heterogeneous collection of very specific building blocks, connected together by a complex network of many dedicated busses and interconnect options. General-purpose programmable processors are not used for energy efficiency reasons. Typically, multiple small embedded processor cores with accelerators, IP cores, etc. are used. The trend to merge multiple functions into one device (e.g. a cell phone with video capabilities) makes the design and integration of these "systems-on-chip" (SOC's) even more challenging.

Yet, specifications and applications are never fixed and require the embedded units to be programmable. A good balance between energy efficiency and programmability can be obtained by using programmable domain-specific processors. A well known example are the programmable digital signal processors (DSPs). DSPs are developed for wireless communication systems (mostly driven by cellular standards). In a first generation this meant that DSPs were adapted to execute many types of filters (e.g. FIR, IRR), later communication algorithms such as Viterbi decoding and more recently Turbo decoding are added.

A first trend we notice is that more applications and multiple applications run in parallel or on demand on the device, e.g. video decoding, data processing, multiple standards, etc. A second trend we notice is that these new applications tend to run either on a separate domain specific programmable processor or on a hardware accelerator (the distinction between the two being rather blurry) next to the embedded DSP or micro-controller instead of being tightly coupled into the instruction set of the host processor.

A third trend we notice is that general-purpose programming environments are getting more heterogeneous and domain-specific. The general-purpose solutions are for energy efficiency reasons augmented with domain specific units, accelerators, IP cores, etc. This is clearly visible in FPGA's, as the new generations now include specialized blocks such as embedded core's, block RAM's and large numbers of multipliers. One successful example is the Virtex-Pro family of Xilinx [17]. These devices contain up to four Power PC cores, multiple columns of SRAM, multiple columns of multipliers, Gbits IO transceivers, etc.

The architecture design of this heterogeneous SOC is a search in a three dimensional design space, which we call the reconfiguration hierarchy [12]. First in the Y direction: at what level of abstraction should the programming be introduced? Secondly in the X direction: which component of the architecture should be programmable? Thirdly in the Z direction: what is the timing relation between processing and the configuration/programming? Programming can be introduced at multiple levels of abstraction. When it is introduced at the instruction set level, it is called a "programmable processor". When it is introduced at the CLB level of an FPGA, it is called a reconfigurable device. Regarding components, a processor has four basic components: data paths, control, memory and interconnect. One has a choice of making some or all of them programmable. Then the third question is to compare the processing activity to the binding time. It makes a system configurable, reconfigurable, or dynamic reconfigurable.

The challenge is to develop a design environment to navigate in this three dimensional design space.

Several SOC platforms have been presented in literature. Most of them focus on general -purpose regular architectures, e.g. [2]. Very few focus on the low power issue and the need to tune the architecture towards the application. One example is the low power Maya platform [18]. Unique to our design approach is that we combine the design and programming of the architecture with an environment to explore the best options.

The paper is organized as follows. Section 2 and 3 look at the architecture design, while section 3 and 4 discuss the design exploration, co-design and co-simulation challenges.

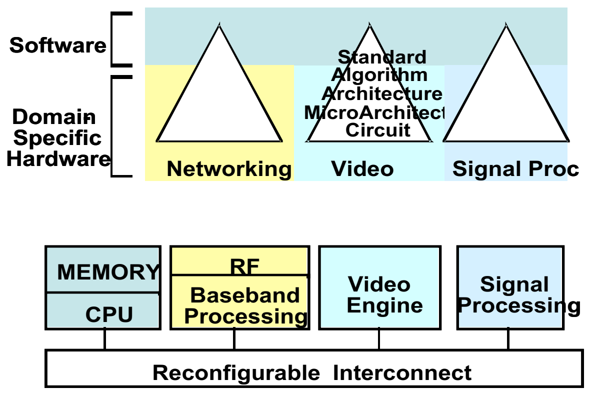

Fig. 1. Example RINGS architecture.

2 Energy efficient heterogeneous SOCs.

The system designer needs an architecture platform that gives him the lowest energy consumption, but at the same time provides enough flexibility to allow re-programming or re-configuration. The key to energy efficiency is to tune the architecture to the application domain. This means freezing flexibility in the X (components) and Y (level of abstraction) direction of the reconfiguration hierarchy. A hierarchy of so-called “Y charts” allows us to do this in a top-down fashion [5].

A complex SOC will consist of multiple domain specific processing engines. Each processor is programmable to a more or less degree. It can be highly programmable if the processor is a micro-controller or a DSP engine or a blank box of CLB units. The efficiency goes up as domain specific instructions are added. An example of this is the addition of a MAC instruction to a DSP processor. Loosely coupled co-processors will be more energy efficient but less flexible as they fit a narrower application domain. An example is the Turbo coder acceleration unit. The ultimate energy efficient block is the optimized hard IP unit. Yet, it does not provide any flexibility. In SOC a range and collection of these blocks are used.

Similarly arguments can be made for the interconnect component of a SOC. Currently, we see only two extreme options: either dedicated one-to-one connections and specialized busses, which have the lowest power consumption (to a first order) or general-purpose global busses or interconnect, as provided by FPGA’s [17] or networks on chip [2]. The latter two are both general-purpose solutions at *different levels of abstraction* to give the designer a maximum flexibility and programmability.

The RINGS architecture [16] is an architecture platform that gives the designer the option to explore the energy flexibility trade-offs. An example is shown in Fig. 1. A RINGS architecture contains a heterogeneous set of building blocks: programmable cores, both DSP’s and micro-controllers, programmable and/or reconfigurable hardware accelerator units, specialized IP building blocks, front-end blocks, and so on. When designing a solution based on RINGS, it is important that the domain expert has freedom to select the appropriate level of flexibility, ranging from fully programmable approaches, such as embedded micro controllers or FPGA blocks to highly optimized IP blocks. For different domains, the flexibility will be supported in different ways as domains have different characteristics. This domain specific flexibility can be expressed as a domain specific abstraction pyramid as shown for Networking, Video, and Signal Processing on Fig. 1. In case of Video, the engine will consist of elements expressed in the Video pyramid, for example dedicated co-processors.

The SOC is connected together at the top level by a supervising software program, which typically runs on an embedded micro-controller. At the bottom level, the reconfigurable interconnect glues it together. The programming paradigm used in RINGS is a reconfigurable network-on-chip. Also in this network, flexibility can be traded for energy efficiency at different levels of abstraction. Designers can instantiate an arbitrary network of 1D and 2 D router modules leading to an architecture illustrated in Fig. 2.

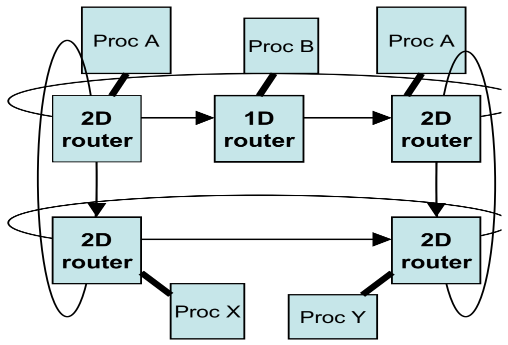

Fig. 2. Example of Network-on-chip [1].

This network illustrates the three binding time concepts. At the level of configuration, the static network architecture with routers is instantiated. Reconfiguration is done by means of reprogramming the routing tables and programming by giving each packet a target address.

A traditional reconfiguration is obtained by reprogramming the routing tables in each node. An alternative approach is to use an easy to reconfigure physical channel. One example of this is a CDMA based reconfigurable interconnect [6][16]. Fig. 3. shows a conceptual picture of a source-synchronous CDMA implementation. Each sender and receiver gets a unique spreading code. By changing the Walsh code, a different configuration is obtained. Traditional busses, which are a TDMA channel, require hardware switches for reconfiguration. CDMA interconnect has the advantage that reconfiguration can occur "on-the-fly."

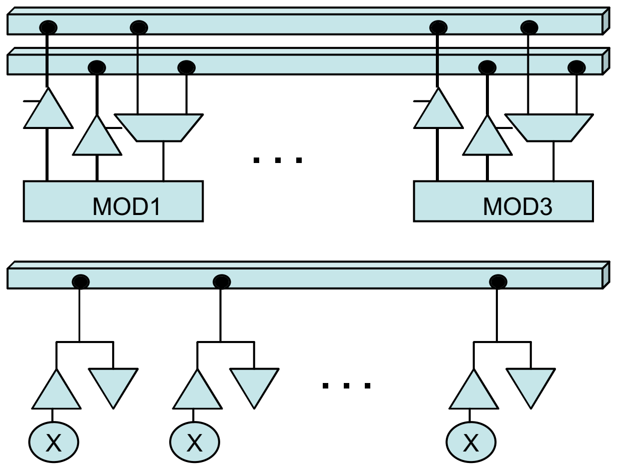

Fig. 3. Reconfigurable interconnect (a) TDMA (b) SS-CDMA bus interface [6].

3 Ultra low power components.

The focus of this section is on the architecture design options to design ultra low power processor components, in many cases without losing performance.

DSP processors have real-time constraints or need to maximize their throughput for a given task while at the same time minimize the power or energy consumption. Therefore, the design of DSP processors is very challenging, as it has to take into account contradictory goals: an increased throughput request at a reduced energy budget. On top there are new issues due to very deep submicron technologies such as interconnect delays and leakage. For instance, hearing aids used analog filters 15 years ago, were designed as digital ASIC-like circuits 5 years ago. Today they are designed with powerful DSP processors below 1 Volt and 1 mW of power consumption [8]. Hearing aids companies require DSP processors just because they require flexibility, i.e. to program the applications in-house.

The design of ultra-low power DSP cores has to be performed at all design levels, i.e. system, architecture, circuit and technology levels. We will focus in this section to DSP architectures, but VHDL implementations as well as cell libraries are important too. Latch-based implementations including gated clocks described in VHDL or Verilog, low-power standard cell libraries and leakage reduction circuit techniques are necessary to reduce power consumption at these low levels.

Various DSP architectures can be and have been proposed to reduce significantly the power consumption while keeping the largest throughput. Beyond the single MAC DSP core of 5-10 years ago, it is well known that parallel architectures with several MAC working in parallel allow the designers to reduce the supply voltage and the power consumption at the same throughput. It is why many VLIW or multitask DSP architectures have been proposed and used even for hearing aids. The key parameter to benchmark these architectures is the number of simple operations executed per clock cycle, up to 50 or more. However, there are some drawbacks. The very large instruction words up to 256 bits increase significantly the energy per memory access. Some instructions in the set are still missing for new better algorithms. Finally the growing core complexity and transistor count becomes a problem because leakage is roughly proportional to the transistor count.

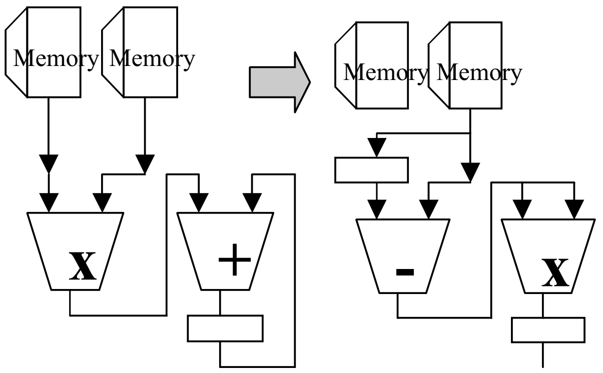

Fig. 4. Hardware reconfiguration example [3] .

To be significantly more energy efficient, there are basically two ways, however impacting either flexibility or the ease of programming: (1) to design specific very small DSP engines for each task, in such a way that each DSP task is executed in the most energy efficient way on the smallest piece of hardware [9]. For N DSP tasks within a given application, the resulting architecture will be N co-processors or hardware accelerators around a controller or a simple DSP core as illustrated on Fig. 1. (2) to design reconfigurable architectures such as the DART cluster [3], in which configuration bits allow the user to modify the hardware in such a way that it can much better fit to the executed algorithms. Fig. 4 shows an example.

Option (1) is definitively the best one regarding power consumption. Each DSP task uses the minimal number of transistors and transitions to perform its work. The control code unavoidable in every application is also efficiently executed on the controller or on the simple DSP, and some unexpected DSP tasks can be executed on the simple DSP if no accelerator is available. However, the main issue is the software mapping of a given application onto so many heterogeneous processors and co-processors (see Section 4). Transistor count could be high and some co-processors fully useless for some applications. Regarding leakage, unused engines have to be cut off from the supply voltages, resulting in complex procedures to start/stop them.

Reconfigurable DSP architectures are much more power efficient than FPGAs. E.g. the MAGIC DSP consumes 1mW/MHz in 1.8V, 0.18µm CMOS. The same MAGIC in an Altera Stratix FPGA consumes about 10mW/MHz of dynamic power, but has a huge static power of 900 mW. So at 10 MHz, it consumes 1000 mW. The key point is to reconfigure only a limited number of units within the DSP core, such as some execution units and addressing units [11]. The latter are interesting, as the operands fetch is generally a severe bottleneck in parallel machines for which 8-16 operands are required each clock cycle. So, sophisticated addressing modes can be dynamically reconfigured depending on the DSP task to be executed. However, the power consumption is necessarily increased due to the relatively large number of reconfiguration bits that have to be loaded in the configuration registers. Similarly, the reconfigurable units are necessarily more complex that non-reconfigurable units in terms of transistor count and therefore consume more. Software issues are also difficult, as users can define new instructions or new addressing modes that are difficult to support by the development tools.

4 Design & architecture exploration.

The way a system behaves depends on the architecture, the way the applications are written, and how these applications are mapped onto the architecture as compactly expressed by the Y-chart [5]. Examples of architectures for low-power have already been given in other sections. On such architecture, mapping is typically done in case of reconfigurable fabrics by the behavioral synthesis tool and the place and route tools. In case of DSPs and CPUs, the mapping is typically performed by C-compilers dedicated to a particular type of DSP or CPU. An important question remains: how to specify the applications that they can take advantage of the architecture in an effective manner.

A low-power architecture will typically employ different levels of parallelism like bit-level parallelism, instruction parallelism or task-level parallelism to take advantage of voltage scaling as already explained in the previous section. To successfully map a DSP application at a high level, the applications need to express task-level parallelism. This parallelism is typically not present, as the applications are written in sequential languages like C or Matlab. Therefore, mapping them is often a manual process that is very tedious and time consuming, leading to a sub optimal system.

A designer would like to have tool support that converts automatically the sequential specification into a parallel format. Moreover, the tool should allow him to 'play' with the amount of parallelism extracted from the specification. In general, such tools are lacking in embedded system design. Some companies, like *Pico* and *Art* (ARM/Adelante) try to provide limited commercial solutions but this field is still very much subject to research. The *Compaan tool* suite [13] aims at providing designers the option to play with parallelism for applications that are so-called "Nested Loop Programs," a very natural fit for DSP applications. A DSP application is specified in a subset of Matlab and is automatically converted by Compaan into a network of parallel processes. These processes can be specified in "C' and mapped, using a conventional C compiler, onto a DSP or CPU. On the other hand, they can also be specified in VHDL and mapped using the appropriate tools onto some reconfigurable fabric or realized as a dedicated IP core [19]. Hence, "programming" the RINGS architecture is reduced to putting some processes onto the CPUs and DSPs while others are mapped onto FPGAs or use dedicated IP cores.

There are many ways we can find parallelism in the application and in the way we partition the processes of the CPUs, DSPs and reconfigurable resources. Being able to explore these options early on in the design phase is crucial to get efficient embedded low-power systems. To allow designers to do this exploration, Compaan is equipped with a suite of techniques [14] like Unfolding, Skewing and Merging, to allow designers to play with the level of parallelism exposed in the derived network of processes. Skewing and Unfolding increase the amount of parallelism, while Merging reduces parallelism. By performing these techniques, many different networks can be created that can be mapped in different ways onto the architecture. When applied in a systematic way, the design space can be explored and the best performing network of processes can be picked.

The difference in utilization of the architecture for a particular network can be huge. By rewriting a DSP application (like Beam-forming) using the presented techniques, we are able to achieve performances on a QR algorithm (7 Antenna's, 21 updates) ranging from 12MFlops to 472MFlops. We realized QR using commercial floating-point IP cores from QinetiQ, that are pipelined 55 (Rotate) and 42 (Vectorize) stages. We achieved this performance increase without doing *anything* to the architecture or mapping tools, but only by playing with the way the QR application is written, effectively improving the way the pipelines of the IP cores are utilized. Using a system like Compaan, an experienced designer should be able to obtain very different performing networks in days, having the opportunity to explore different systems and picking the one that uses the least amount of power.

5 Domain-Specific Co design Environments

As discussed in the previous section, parallelism and distributed processing are key to energy efficient architectures. Because the ensemble of architecture elements (processors, busses, memories) cooperate towards a common application, the designer faces a considerable co-simulation and co-design problem. A key requirement is to have a good design model. Such a model allows building of simulation tools, compilers and code generators. We will look at a highly successful design model for programmable systems: the instruction-set architecture (ISA). Next we will consider the approach taken by the RINGS architecture.

In a classic Von-Neumann architecture, the instruction-set-architecture (ISA) model maintains a single, consistent and abstracted view to the operation of the system. Such a view ties four independent architecture concepts together: control, interconnect, storage, and data operations [15]. This way the ISA becomes a template for the underlying target architecture, for which compiler algorithms (scheduling etc) can be developed. Often however, the ISA is unable to offer the right target template – in terms of parallelism, storage capabilities or other.

In the RINGS architecture, we do not use an ISA as an intermediate design model, but approach each of the four components that make up an ISA independently. We enumerate them below and look at the requirements they impose on co-simulation and co-design.

- **Data Operations**: Energy efficient operation requires us to specialize each operator as much as possible. A RINGS system contains multiple processing cores. These can include hardwired or programmable (DSP or RISC) processors. We thus need to be able to combine instruction-set simulation with hardware simulation.
- **Storage**: Energy efficient operation requires us to distribute storage. In addition to the high-level design transformations discussed in the previous section, we target to minimize storage bandwidth and use multiple distributed memories. Each processor in RINGS will work inside of a private memory space. Many operations in multimedia can be implemented with dedicated storage architectures that take only a fraction of the energy cost of a full-blown ISA. Examples are matrix transposition or scan-conversion. Such dedicated storage can be captured as a hardwired processor.
- **Interconnect**: The energy efficient interconnect architecture discussed in section 2 requires explicit expression of interconnect operations – in contrast to an ISA where this is implicitly encoded in the instruction format. A network-on-chip can be modeled as a dedicated hardware architecture [1]. On top of the network-on-chip a suitable network protocol must be implemented, for example message-passing with the MPI standard [7]. However, also this protocol is subject to specialization and/or hard-coding. For example, a hardwired DCT coding unit attached to a DSP core through RINGS will have a fixed communication pattern. This pattern can be hard-coded in a collapsed and optimized protocol stack.

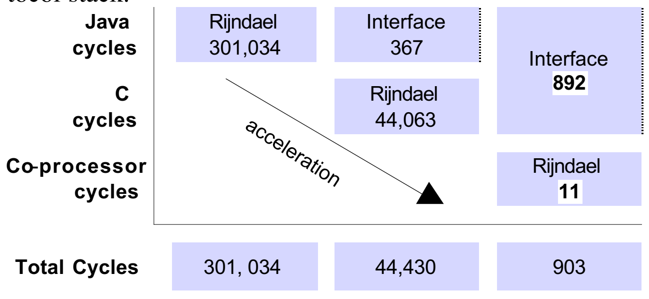

Fig. 5. Overhead of Tightly Coupled Data/Control Flow

- **Control**: Energy efficient operation requires us to split the data-flow and control-flow in a RINGS architecture and handle them independently. Fig. 5 clarifies this point. It shows the effect of moving an AES encryption operation gradually from high-level software (Java) implementation to dedicated hardware implementation, while at the same time maintaining the interface to the high level Java model. It can be seen that the interface overhead goes from 0.8% for a C-accelerated AES to 8000% for a hardware-accelerated AES! This overhead obviously is caused by all the interfaces moving data from Java to C to hardware and back. With the MPI message passing scheme, we have the freedom to route control flow and a data flow independently as messages. This way, we can eliminate or minimize this interface overhead.

When we put the elements together, we conclude that the RINGS co-design environment should accommodate multiple instruction-set simulators with user-specified hardware models. All of these must be embedded in a model of an on-chip network. The timing accuracy of the simulation should be precise enough to simulate interactions such as network-on-chip communication conflicts. On the other hand, the simulation must also be fast enough to support reasonable design exploration capabilities.

We have built the ARMZILLA environment to evaluate one class of RINGS architectures, namely those that can be built with one or more ARM cores, a network-on-chip, and dedicated hardware processors. Fig. 6 illustrates the ARMZILLA setup. There are three components: a hardware simulation kernel (GEZEL), one or more instruction-set simulators (ISS), and a configuration unit. The GEZEL kernel [4] captures hardware models with the FSMD (Finite-State-Machine with Datapath) model-of-computation. It uses a specialized language and a scripted approach to promote interactive design exploration. The cycle-true models of GEZEL can also be automatically converted to synthesizable VHDL. For the ARM ISS we use the cycle-true

SimIT-ARM environment [10]. The ARM ISS uses memory-mapped channels to connect to the GEZEL hardware models. Finally, the configuration unit specifies a symbolic name for each ARM ISS, and associates each ISS with an executable. This way the memory-mapped communication channels can be set up, and the hardware GEZEL models can address each ARM memory space uniquely.

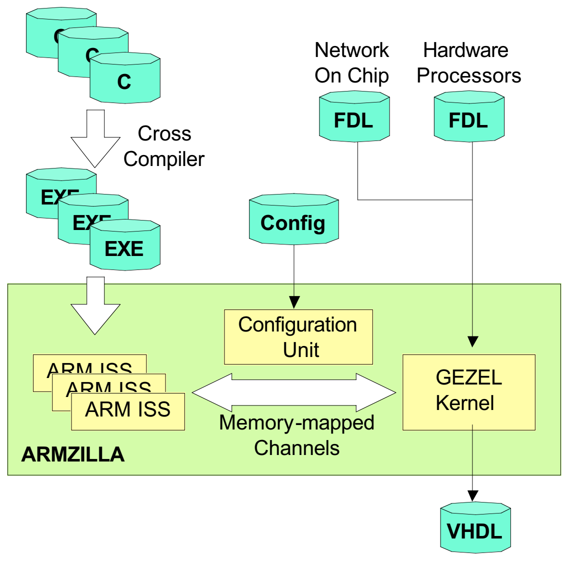

Fig. 6. The ARMZILLA Design Environment for ARM-based RING Processors.

An example of what can be done with the ARMZILLA environment is shown in Table 1. This table shows cycle counts that were obtained after partitioning a JPEG encoding algorithm. The reference implementation runs on a single-ARM ISS model. In the second implementation, we separate the chrominance and luminance channels over two ARM processors. This seems a logical partition that splits the data operations roughly in two parts. But, it also creates a communication bottleneck in the on-chip network and the resulting implementation becomes *slower* then the O3-level optimized single-processor implementation. The third implementation shows a better partitioning. In this case, the data streams are routed out of the ARM and into dedicated hardware processors for JPEG encoder subtasks. These processors can communicate directly amongst themselves.

Table 1. Multiprocessor JPEG Encoding Performance

Partition	Cycle count 64x64 block
One single ARM	1.223 M
Dual ARM using split chrominance/ luminance channels	1.336 M
Single ARM with color conversion, transform coding, Huffman coding as stand-alone hardware processors	313K

All these simulations are cycle-accurate yet they can run efficiently. For the H.264 decoding on a dual ARM with network-on-chip for example, ARMZILLA offers a simulation speed of 176K cycles per second. The simulation speed varies with the complexity of the hardware model used. A single, stand-alone SimIT-ARM simulator runs at 1 MHz cycle-true on a 3GHz Pentium.

6 Conclusions.

In this paper, we presented architecture design and design exploration for low power systems-on-chip. Low power is obtained by tuning all components of the architecture (datapaths, control, memory and interconnect) to the application. This can occur at different levels of abstraction. The design of this type of SOC requires support by design models and methods. The design environments Compaan and Gezel /Armzilla are illustrations of supporting tools for this design space exploration.

7 References.

[1] D. Ching, P. Schaumont, I. Verbauwhede, "Integrated Modeling and Generation of a Reconfigurable Network-On-Chip," UCLA IVgroup Technical Report, September 2003.
[2] W. Dally, B. Towles, "Route Packets, not wires: on-chip interconnection networks," Proc. DAC 2001.
[3] R. David et al. « Low-Power Reconfigurable Processors", Chapter 20 in "Low Power Electronics Design", edited by C. Piguet, CRC Press, will be published in April 2004.
[4] GEZEL kernel, http://www.ee.ucla.edu/~schaum/gezel
[5] B. Kienhuis, et al.``A Methodology to Design Programmable Embedded Systems'', LNCS, Vol 2268, Nov. 2001.
[6] J. Kim, et al., "A 2-Gb/s/pin Source Synchronous CDMA Bus Interface with simultaneous Multi-Chip Access and Reconfigurable I/O capability", CICC, Sept 2003.
[7] MPICH – A portable implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich/
[8] P. Mosch et al. « A 720 µW 50 MOPS 1V. DSP for a Hearing Aid Chip Set", Proc. ISSCC, pp. 238-239, Feb. 2000.
[9] zg n Paker et al. "A heterogeneous multi-core platform for low power signal processing in systems-on-chip", ESSCIRC 2002.
[10] W. Qin, S. Malik, "Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation," Proceedings of DATE 2003, Mar, 2003, pp.556-561.
[11] F. Rampogna et al. « Magic, a Low-Power, re-configurable DSP", Chapter 21 in "Low Power Electronics Design", ed. C. Piguet, CRC Press, published in April 2004.
[12] P. Schaumont, I. Verbauwhede, M. Sarrafzadeh, K. Keutzer, "A quick safari through the reconfiguration jungle," Proceedings DAC 2001, pg. 172-177, June 2001.
[13] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, E. Deprettere,``System Design using Kahn Process Networks: The Compaan/Laura Approach'', DATE2004, Feb 2004, Paris, France.
[14] T. Stefanov, B. Kienhuis, E. Deprettere, "Algorithmic Transformation Techniques for Efficient Exploration of Alternative Application Instances'', Proc. CODES'2002, Colorado, May 2002.
[15] I. Verbauwhede, J. M. Rabaey. "Synthesis of Real-Time Systems: Solutions and challenges" Journal of VLSI Signal Processing, Vol. 9, No. 1/2, Jan. 1995, pp. 67-88.
[16] I. Verbauwhede, M.C. F. Chang, "Reconfigurable Interconnect for next generation systems", Proc. SLIP, pp. 71-74, April 2002.
[17] Xilinx: Virtex-II-Pro Platform FPGAs: Introduction and Overview and Functional Description, Aug. 2003, Oct. 2003, www.xilinx.com/bvdocs/publications/ds083-1.pdf, ds083-2.pdf.
[18] H. Zhang, et al., "A 1V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications," IEEE Journal on Solid State Circuits, November 2000.
[19] C. Zissulescu, et al., ``Laura: Leiden Architecture Research and Exploration Tool'', Proc. FPL 2003.