Processor-Core Based Design and Test

Peter Marwedel
Universität Dortmund, Informatik XII
44221 Dortmund, Germany
e-mail: marwedel@ls12.informatik.uni-dortmund.de

Abstract— This tutorial responds to the rapidly increasing use of various cores for implementing systems-on-a-chip. It specifically focusses on processor cores. We will give some examples of cores, including DSP cores and application-specific instruction-set processors (ASIPs). We will mention market trends for these components, and we will touch design procedures, in particular the use compilers. Finally, we will discuss the problem of testing core-based designs. Existing solutions include boundary scan, embedded in-circuit emulation (ICE), the use of processor resources for stimuli/response compaction and self-test programs.

I. Introduction

In response to the increasing size of advanced chips and the continuing need for fast design cycles, a major amount of new designs is using complex cores (rather than standard cells and macroblocks) as building blocks. Such cores include: processor cores, communication cores, bus interface cores, and memory cores. These cores are available both from vendors and within system companies. Applications can be found in most segments of the embedded system market, such as automotive electronics and telecommunications.

Due to space and time constraints, the current text focusses on *processor* cores. Essential advantages of these processors include their high flexibility, short design time and (in the case of off-the-shelf processors) full-custom layout quality. Furthermore, they allow an easy implementation of optional product features as well as easy design correction and upgrading. Also, processors are frequently used in cases where the systems must be extremely *dependable*. In such cases, the re-use of the design of an off-the-shelf processor greatly simplifies dependability analysis.

II. Processor core examples

Core processors include core versions of general purpose processors (such as core versions of various RISC architectures [17, 1, 35], core versions of digital signal processors (DSPs) and application-specific instruction set processors (ASIPs). A classification of processors is shown in fig. 1. This *processor cube* results from using three main criteria for classifying processors: availability of domain-specific features, availability of application-specific features, and the form in which the processor is available.

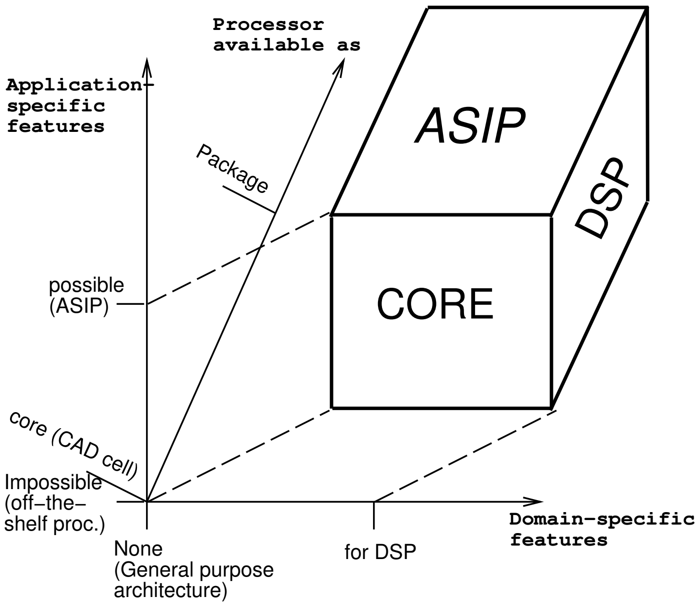

Fig. 1. Cube of processor types

The meaning of these dimensions and their values is as follows:

1. *Form in which the processor is available*

   At every point in time, the design and fabrication processes for a certain processor have been completed to a certain extent. The two extremes considered here are represented by completely fabricated, packaged processors and by processors which just exist as a *cell* in a CAD system. The latter is called a *core processor*. *In-house cores* are proprietary cores available just within one company. They usually have some architectural flexibility. Cores can be instantiated from the library to become part of a larger *system-on-a-chip*. In addition to cores, systems-on-a-chip may contain RAMs, ROMs, and special *accel-*

erators. With these, much of the performance penalty caused by the use of flexible processors can be compensated.

2. *Domain-specific features*

   Processors can be designed to be *domain-specific*. Possible domains are *digital signal processing* or *control-dominated applications*.

   DSP processors [22] contain special features for signal processing: multiply/accumulate instructions, specialized ("heterogenous") register sets, multiple ALUs, special DSP addressing modes (for example, for ring buffers), and saturating arithmetic operations.

3. *Application-specific features*

   At any point in time, the internal architecture of a processor may either be fixed or still allow configurations to take place.

   The two extremes considered here are: processors with a completely fixed architecture and ASIPs. Processors with a fixed architecture or *off-the-shelf processors* have usually been designed to have an extremely efficient layout. Some of them have passed verification procedures, allowing them to be employed in safety-critical applications.

   ASIPs are processors with an application-specific instruction set. Depending upon the application, certain instructions and hardware features are either implemented or unimplemented. Also, the definition of ASIPs may include *generic parameters*. By "generic parameters" we mean compile-time parameters defining, for example, the size of memories and the bitwidth of functional units. A very nice set of references to ASIPs is contained in a recent contribution by Paulin [39]. A well-known example is the EPICs architecture [51]. Optimal selection of instructions, hardware features and values for parameters is a topic which has recently received interest in the literature [2, 42, 16]. ASIPs have the potential of requiring less area or power than off-the-shelf processors. Hence, they are popular especially for low-power applications.

In addition to the three coordinates, there are of course other criteria for classifying processors.

III. Market trends

Statistical data concerning the increased used of processors for implementing information processing systems is available through the web pages of *EE Times* [10] and papers published by Paulin [38, 41, 40]. According to Paulin, about 82 % of all designs analysed in a study at Bell Northern Research (BNR) were essentially based on processors. In recent papers, a clear trend towards ASIPs has been identified.

IV. Designing with core processors

Due to the trend towards using cores in general and processor cores in particular, it is important to analyse design procedures for systems containing processor cores.

Codesign

From a conceptual point of view, designers start with an overall behavioural specification. Standard languages such as C or VHDL are currently very popular for this step. Recently introduced graphical or semi-graphical languages [12], aim at faciliating this step. The specification is then partitioned into software parts and hardware parts. Different approaches to this step of *hardware/software co-design* have been described in an excellent survey by Buchenrieder [7]. Software parts (e.g. a fraction of the C program) are later compiled onto an envisioned processor. Hardware parts are used as input to a hardware synthesis system. A survey on codesign has recently been presented by R. Gupta [14].

Cosimulation

In the codesign environment, simulations are needed at different levels. First of all, the specification has to be simulatable. This is required in order to check whether or not the specified algorithm really performs the intended function. Later, the generated code will be simulated using an instruction set model of the processor. This simulation can take the generated hardware parts into account. Finally, the processor may also be simulated at the structural level. Achieving high simulation speeds also an important issue. See, for example, Valderrama [50].

Runtime environment

Embedded systems using board-level integration frequently take advantage of available real-time operating systems in order to provide an environment suitable for running programs on processors. For chip-level integration, storage requirements for current operating systems do not allow this solution. The knowledge about application programs is exploited in an IMEC approach for avoiding those bulky operating systems [8].

Real-time response

Systems-on-a-chip have to guarantee a certain real-time response to external events. The issue of specifying, analyzing and checking timing constraints is covered, for example, in books by Ku, De Micheli and Gupta [19, 13], and papers by Boriello [4] and by Li, Malik, Wolfe [30].

Compilers

It has been observed (see e.g. Paulin [38]) that the majority of processor-based designs is implemented using assembly languages. The reason for this is the poor performance of current compilers for DSPs. Quantitative comparisons between compiler-generated code and assembly language libraries provided by the processor vendors have been published by Zivojnovic [52]. Recent research has aimed at the design of new compiler optimizations taking the special characteristics of the application area and the target processors into account.

A number of pointers to such algorithms is contained in a paper by Liao, Devadas [31]. Improved address assignment techniques have recently been published by Liao [32], Liem [34] and Leupers [27]. Code compaction for an existing machine has been studied recently by Leupers [29], Timmer [49], Strik [46], and Nicolau [37]. Register and memory bank assignment studied, for example, by Rimey [45], by Bradlee [5], by Hartmann [15], and by Malik [47].

Currently, compilers for fixed target architectures are employed. However, they do not provide the flexibility for experimenting with different target processors. They do not allow trying out different ASIP parameters, and leaving out or adding certain processor features. Code generation which supports this process has to be retargetable. "Retargeting" in this context means: *fast and easy* retargeting, simple enough to be *handled by the user*. In order to provide compiler support for ASIPs, a few so-called retargetable compilers have been designed. There is a book focussing on retargetable compilers [36]. For details about such compilers see Paulin [33], Fauth [11], Leupers [29, 28] and Goossens [21, 43].

V. Testing

Testing of systems-on-a-chip comprising cores is a very difficult issue. Many signals which were previously directly accessable are now only available within a chip. In response to the demands of test engineers, *boundary scan* is available for most cores. This technique originally was intended to be used at the board level and now found a new application.

As a replacement for board-level in-circuit emulation (ICE), so called *embedded* ICE is available for some cores. Such cores allow monitoring selected signals, such as memory or program counter-related signals. As in the case of chip-level boundary scan, there seems to be no agreement between companies, whether or not the expenses for the required additional test hardware can be tolerated.

In the case of processor cores, the existence of data path resources can be exploited for testing. For example, Rajski [44] and Kunzmann [20] propose techniques for test pattern generation and compaction using data path resources. Alternatively, self-test programs can be executed on processor cores. Functional test program generation was first proposed by Thatte and Abraham [48] and later refined [6]. Knowledge about the structure was exploited by Lee and Patel [23, 25, 24, 26] and Krüger [18]. A new approach using the constraint logic programming language ECLIPSE [9] has been implemented by Bieker [3].

VI. Conclusion

Currently, there is a significant shift in how complex systems are designed. The use of cores, and -in particular- the use of processor cores, is an essential characteristic of this shift. This shift demands for new CAD techniques. It is no longer feasible to restrict the designer's scope to hardware design. For example, compiler- and operating system-related issues have to be taken into account. This text aims at motivating research into this direction.

References

[1] Advanced RISC Machines Ltd. ARM. web pages. *http://www.arm.com/*, 1995.

[2] A. Alomary, T. Nakata, Y. Honma, M. Imai, and N. Hikichi. An ASIP instruction set optimization algorithm with functional module sharing constraint. *Int. Conf. on Computer-Aided Design (ICCAD)*, pages 526–532, 1993.

[3] U. Bieker and P. Marwedel. Retargetable self-test program generation using constraint logic programming. *32nd Design Automation Conference*, 1995.

[4] G. Boriello. Software scheduling in the co-synthesis of reactive real-time systems. *Proceedings of the 31th Design Automation Conference*, pages 1–4, 1994.

[5] D. G. Bradlee, S. J. Eggers, and R. R. Henry. Integrating register allocation and instruction scheduling for RISCs. *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 122–131, 1991.

[6] D. Brahme and J. A. Abraham. Functional testing of microprocessors. *IEEE Trans. on Computers*, pages 475–485, 1984.

[7] K. Buchenrieder. *Hardware/Software Co-Design.* ITT Press Hartenstein, ISBN 3-929814-07-4, 1995.

[8] M. Cornero, F. Thoen, G. Goossens, and F. Curatelli. Software synthesis for real-time information processing systems. *in: P. Marwedel, G. Goossens (ed.): Code Generation for Embedded Processors*, Kluwer, 1995.

[9] European Computer Research Center ECRC. ECLIPSE 3.4 user manual, ECRC common logic programming system. *ECRC GmbH, Munich*, 1994.

[10] EE Times. web pages. *http://techweb.cmp.com/techweb/eet-/embedded/embedded.html*, 1995.

[11] A. Fauth and A. Knoll. Automated generation of DSP program development tools using a machine description formalism. *Int. Conf. on Audio, Speech and Signal Processing*, 1993.

[12] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and design of embedded systems.* Prentice Hall, 1994.

[13] R. Gupta. *Co-synthesis of Hardware and Software for Embedded Systems.* Kluwer Academic Publishers, 1995.

[14] R. Gupta. Hardware software co-design of embedded systems. *Tutorial at the VLSI Design Conference, Bangalore*, 1996.

[15] Hartmann. Combined scheduling and data routing for programmable ASIC systems. *EDAC*, pages 486–490, 1992.

[16] I.-J. Huang and A. Despain. Generating instruction sets and microarchitectures from applications. *Int. Conf. on CAD (ICCAD)*, pages 391–396, 1994.

[17] Intel Corp. web pages. *http://www.intel.com/product/techbriefs/index.html*, 1996.

[18] G. Krüger. A tool for hierarchical test generation. *IEEE Trans. on CAD, Vol. 10*, pages 519–524, 1991.

[19] D. Ku and G. De Micheli. *High Level Synthesis Under Timing and Synchroniszation Constraints*. Kluwer Academic Publishers, 1992.

[20] A. Kunzmann. Test pattern generation hardware motivated by pseudo-exhaustive test techniques. *EURO-DAC*, pages 240–245, 1994.

[21] D. Lanneer, J. Van Praet, A. Kifli, K. Schoofs, W. Geurts, F. Thoen, and G. Goossens. CHESS: retargetable code generation for embedded DSP processors. *in: P. Marwedel, G. Goossens (ed.): Code Generation for Embedded Processors, Kluwer Academic Publishers*, 1995.

[22] E. Lee. Programmable DSP architectures, parts i and ii. *IEEE ASSP Magazine*, Oct. 1988 & Jan. 1989, 1988.

[23] J. Lee and J.H. Patel. An architectural level test generator for data path faults and control faults. *Proc. of the Intern. Test Conference*, pages 729–738, 1991.

[24] J. Lee and J.H. Patel. Hierarchical test generation under intensive global functional constraints. *Proc. 29th Design Automation Conf.*, pages 261–266, 1992.

[25] J. Lee and J.H. Patel. An instruction sequence assembling methodology for testing microprocessors. *Proc. of the Intern. Test Conference*, pages 49–58, 1992.

[26] J. Lee and J.H. Patel. Architectural level test generation for microprocessors. *IEEE Trans. on Computer-Aided Design*, pages 1288–1300, 1994.

[27] R. Leupers. Algorithms for address assigment in DSP code generation. *ICCAD*, 1996.

[28] R. Leupers and P. Marwedel. A BDD-based frontend for retargetable compilers. *European Design & Test Conference*, 1995.

[29] R. Leupers and P. Marwedel. Time-constrained code compaction for DSPs. *Int. Symp. on System Synthesis (ISSS)*, 1995.

[30] Y-T. S. Li, S. Malik, and A. Wolfe. Performance estimation of embedded software with instruction cache modeling. *Int. Conf. on Computer-Aided Design (ICCAD)*, pages 380–387, 1995.

[31] S. Liao, S. Devadas, K. Keutzer, and S. Tijang. Code optimization techniques for embedded DSP microprocessors. *32nd Design Automation Conference*, pages 599–604, 1995.

[32] S. Liao, S. Devadas, K. Keutzer, S. Tijang, and A. Wang. Storage assignment to decrease code size. *Programming Language Design and Implementation (PLDI)*, 1995.

[33] C. Liem and P. Paulin. FlexWare – A flexible firmware development environment. *Proc. European Design & Test Conference (EDAC-ETC-EUROASIC)*, pages 31–37, 1994.

[34] C. Liem, P. Paulin, and A. Jerraya. Address calculation for retargetable compilation and exploration of instruction set architectures. *to appear: 33rd Design Automation Conference*, 1996.

[35] LSI Logic Inc. web pages. *http://www.lsil.com/products/-unit5_5.html*, 1996.

[36] P. Marwedel. Introduction. *in: P. Marwedel, G. Goossens (ed.): Code Generation for Embedded Processors*, Kluwer, 1995.

[37] S. Novack, A. Nicolau, and N. Dutt. A unified code generation approach using mutation scheduling. *in: P. Marwedel, G. Goossens (ed.): Code Generation for Embedded Processors, Kluwer Academic Publishers*, 1995.

[38] P. Paulin. DSP design tool requirements for the nineties: An industrial perspective. *High-Level Synthesis Workshop, Dana Point, Cal.*, 1992.

[39] P. Paulin, M. Cornero, C. Liem, F. Na abal, C. Donawa, S. Sutarwala, and C. Valderrama. Trends in embedded systems technology: An industrial perspective. *in: M.G. Sami, G. De Micheli: Hardware/Software Codesign, Kluwer Academic Publishers*, 1996.

[40] P. Paulin and C. Liem. Embedded systems: Trends and tools. *Tutorial at the European Design & Test Conference*, 1996.

[41] P. Paulin, C. Liem, T. May, and S. Sutarwala. DSP design tool requirements for embedded systems: A telecommunications industrial perspective. *Journal of VLSI Signal Processing*, pages 23–47, 1995.

[42] J. V. Praet, G. Goossens, D. Lanneer, and H. De Man. Instruction set definition and instruction selection for ASIPs. *7.th Int. Symposium on High-Level Synthesis*, pages 11–16, 1994.

[43] J. V. Praet, D. Lanneer, G. Goossens, W. Geurts, and H. De Man. A graph based processor model for retargetable code generation. *European Design & Test Conference*, 1996.

[44] J. Rajski and J. Tyszer. Multiplicative window generators of pseudo-random test vectors. *European Design & Test Conference (ED&TC)*, 1996.

[45] Rimey and Hilfinger. Lazy data routing and greedy scheduling for application-specific processors. *21st Annual Workshop on Microprogramming (MICRO-21)*, pages 111–115, 1988.

[46] M. Strik, J. van Meerbergen, A. Timmer, and J. Jess. Efficient code generation for in-house DSP cores. *European Design & Test Conference*, pages 244–249, 1995.

[47] A. Sudarsanam and S. Malik. Memory bank and register allocation in software synthesis for ASIPs. *Intern. Conf. on Computer-Aided Design (ICCAD)*, pages 388–392, 1995.

[48] S.M. Thatte and J.A. Abraham. Test generation for microprocessors. *IEEE Trans. on Computers*, pages 429–441, 1980.

[49] E. Timmer. Conflict modelling and instruction scheduling in code generation for in-house DSP cores. *32th Design Automation Conference*, 1995.

[50] C.A. Valderrama and et al. A unified model for co-simulation and co-synthesis of mixed hardware/software systems. *European Design & Test Conference*, 1995.

[51] R. Woudsma. EPICS, a flexible approach to embedded DSP cores. *Int. Conf. on Signal Processing and Applications and Technology*, 1994.

[52] V. Zivojnovic and et al. DSPstone: A DSP-oriented benchmarking methodology. *Proc. of the Intern. Conf. on Signal Processing and Technology*, 1994.