A DESIGN METHODOLOGY FOR EMBEDDED SYSTEMS BASED ON MULTIPLE PROCESSORS

Luigi Carro, Flávio Wagner, Márcio Kreutz,
Marcio Oyamada

This paper presents S³E²S, a CAD environment that allows specification, co-simulation and synthesis of embedded electronic systems, which are composed of a combination of analog parts, digital hardware, and software. S³E²S is based on a distributed, object-oriented system model, where abstract objects are initially used to express complex behavior and may be later refined into digital or analog hardware and software. The target architecture is a set of processors, each one with a possibly different architecture, covering microcontrollers, digital signal processing and RISC-like architectures. S³E²S offers a semi-automatic mechanism for selecting the processor that is best suited for implementing the function of each object in the system model. Besides performance, during the synthesis step S³E²S takes into account different design goals, like power consumption and area. This paper presents the processor selection methodology, as well as results from different applications.

1. Introduction

Embedded electronic systems contain a combination of software and hardware, both analog and digital. Although simple systems can be implemented with a single, off-the-shelf microcontroller, a digital signal processor or a conventional microprocessor and associated software, more complex systems that have critical requirements regarding aspects such as area, speed, and power consumption ask for a dedicated design. Various target architectures can be considered for matching different requirements. Solutions may include dedicated processors and/or ASICs, or even multi-processor platforms, combined with dedicated analog parts.

Typical system examples are portable multimedia devices, industrial distributed controllers or vehicle supervision systems. All these systems demand digital signal processing, analog circuits to interface with the real world, radio frequency communication links and scalar processing for database lists, display and keyboard control. The current trend is to have a mix of behaviors in the same system-on-chip, requiring different design styles and processor architectures.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35409-5_23](https://doi.org/10.1007/978-0-387-35409-5_23)

The design of such complex embedded systems encompasses a suite of different technologies, tools, and design styles. A complete design environment must consider system specification, partitioning among software, digital hardware and analog parts, and synthesis of software and hardware parts and interfaces. The design should ideally proceed from an initial, abstract specification, going through a sequence of successive refinements, until a final detailed solution is achieved. Intermediate, heterogeneous descriptions generated during this stepwise refinement must be validated, usually by co-simulation.

In the S^3E^2S (Specification, Simulation, and Synthesis of Embedded Electronic Systems) environment, complex systems may be modeled not only at different abstraction levels, but also at different domains – abstract behavior (expressed by a high-level, object-oriented specification), digital hardware, analog hardware, and software. Co-simulation is supported by coupling different simulation engines, so that any heterogeneous model developed during a process of stepwise refinement is supported (Wag, 99).

The S^3E^2S environment allows an easy exploration of the design space at a multi-processor level, selecting a combination of processors which best matches the design requirements, regarding not only speed, but focusing on area and power as well. This paper covers the processor selection features of S^3E^2S .

The remainder of this paper is organized as follows: The next section presents a comparison of S^3E^2S with other design approaches. Section 3 introduces the methodology for processor selection. Section 4 discusses case studies that illustrate the capabilities of the design environment. Section 5 presents conclusions and future work.

2. Related work

The design of a simple embedded system can be solved with a single microprocessor or microcontroller and its associated software. In the case of complex designs, however, many issues regarding system specification, simulation and synthesis arise. The first hard task is to specify the desired behavior. The description of complex systems through a formal and abstract language is an open issue (Adam, 1996; Böt, 1998). Recently, a first attempt to define a benchmark to system level specification was developed, but with no clear conclusion on which specification approach would lead to the best results (Neb, 1999; Mos, 1999). Two basic approaches have been proposed: the specification with a single language or model and the specification of a heterogeneous system, by combining different languages or models.

Ptolemy (Kal, 1995), for instance, is an environment for simulation and prototyping of heterogeneous systems, using object-oriented technology. Ptolemy accomplishes multi-paradigm simulation by supporting internally different mechanisms, called domains. This way, objects in Ptolemy can be simulated using models such as Synchronous Data Flow, Dynamic Data Flow, discrete event, and analog domain (Pin, 1998). Another environment allowing the specification and simulation of heterogeneous systems is described by Jerraya and Ernst (Jer, 1999), where a backbone in the operating system implements communication among dedicated simulators that are needed for validating a heterogeneous model specified

with different languages. MCI (Hes, 1999) is also a generic mechanism for integrating different simulators for validating multi-language specifications.

The description of complex systems through a single, abstract language has been also proposed. Some approaches that follow this strategy adopt an object-oriented specification to describe both software and hardware (Böt, 1998; Wol, 1997; Woo, 1997; Aig, 1997). Most work on hardware and software co-design focus on the synthesis of a dedicated hardware or of a dedicated instruction set processor (Adam, 1996; Kal, 1995; Wol, 1997; Mrv, 1998). In the Polis system, the model of execution is a set of commercial processors (Chi, 1994). However, all processors have the same characteristics. They are microcontrollers, targeted to embedded control, and not to data intensive applications. The synthesis style is based on software synthesis and performance estimation techniques.

S³E²S combines the advantages of the multi-language and heterogeneous approach with the abstract, object-oriented specification (Wag, 1999). In S³E²S we also aim at using as much software as possible, in order to reduce system cost and design time, and the target architecture also follows a multi-processor paradigm. Instead of a fixed target architecture devoted to ASICs or ASIPs, however, S³E²S synthesis is based on a library of processors, each with different characteristics, ranging from micro-controllers to digital signal processors. Therefore, differently from Polis, our target system can combine data dominant and control dominant behaviors, and the system tries to find the best processor (according to some design criteria) for each task.

3. Object evaluation and processor selection

The synthesis step in S³E²S is not targeted at a single, specific processor architecture. Instead, it allows easy design space exploration at the multi-processor level, whereby different processor architectures are analyzed, and those best matching the desired application requirements are selected and combined for design refinement. Moreover, in S³E²S we try to use available processors as much as possible, in order to reduce hardware costs and to enhance design time. Since nowadays one can find different microprocessors with different costs, architectures, and power consumption, using them in the design cycle generally turns out to be a flexible and low cost solution. We must also consider that designs are seldom started from scratch. Most companies try to reuse previously designed boards, multi-chip modules or IP processors for which a library of software modules is available. Furthermore, small and medium companies rarely have the capital to invest in high volume, single chip solutions. This way, the use of programmable processors is a natural choice to start a new product.

After modeling the system in the object-oriented environment, the primary target to obtain a working system is to map all objects to one or more physical processors. This strategy assumes that, nowadays, there are different commercial processors available, with different cost/performance ratios, ranging from high performance DSPs to low power microcontrollers.

3.1 Object evaluation

In (Suz, 1996) the evaluation of software performance is based on a two-step procedure. At first, a high-level processor-independent representation is obtained, like a CDFG, and then the CDFG is translated into C code for the target processor. In S^3E^2S we also use an intermediate description of the code to be executed. Over a CDFG structure, an evaluation of the behavior of the object is obtained. Differently from (Suz, 1996), whose work is targeted to controllers, each software module in S^3E^2S is free from any previous template, so that each object has any possible behavior. This way, one must find out the typical behavior of the object code: a) control-dominated, as in FSMs for controllers; b) data-intensive computations, as in digital filters; or c) memory-intensive computations, as in list processing or data-base searching in a building entrance control, for example.

Each object is targeted to a processor that best implements its behavior. The criteria for choosing the best processor are based on the processor characteristics to execute the desired code. For example, a DSP processor with a deep pipeline will pay a high branch penalty and is not adapted to a control-intensive application. On the other hand, if a low cost microcontroller can be used in a slow varying process that requires digital filtering at largely spaced samples, then this solution should also be given as an option for the designer.

From the CDFG, a 3-address code for a virtual machine is generated. Actually, three different virtual machines are used, each one for a specific family of target architectures (microcontrollers, DSP processors, and RISC architectures). The purpose of this specialization is to enhance the predictability of software performance when executing on a certain class of processors. This way, the virtual machine for microcontrollers has only 2 working registers, and most operations use the internal accumulator. This way, most part of operating data must use the memory, slowing down the processor. This fact reflects actual characteristics of real microcontrollers.

In the virtual machine targeted to DSP applications, memory references are used as registers, and special instructions like multiply-accumulate (MAC) are identified in the code. This tries to mimic the fact that DSP architectures are targeted to data-dominant applications, and so memory is accessed in a pipeline operation, with small timing penalty. On the other hand, control dominated programs often break the pipeline, incurring in a timing penalty. Finally, the RISC-like virtual machine has a large register set and operations are performed register to register. We assume a limit of 32 registers. This means that the RISC-like virtual machine will favor complex computations (even filters with small number of taps), up to the limit that more data than can be stored in the internal registers is required.

The next step concerns object analysis. One tries to find which characteristic of the object is dominant: a) control-intensive – many control instructions and flow breaks; b) memory-intensive – list processing, digital filtering, much memory usage; or c) data-intensive – few memory access, most processing done with internal registers. Each of these characteristics will favor a different processor in the library.

Let M be the total number of cycles used in memory access in the internal 3-address code, P the number of cycles to execute all data transformations (add, sub, and, mult, etc), and C the total number of cycles taken to test and branch (control

instructions). These numbers are obtained from the 3-address code and are thus specific for each virtual machine. The total number of operations in the application is thus $P+M+C$. Let AP_x (Application Profile) be the relative importance of each behavior x in comparison with others, expressed as

$$APP = P / (P + M + C), \quad (1)$$

$$APM = M / (P + M + C), \quad (2)$$

$$APC = C / (P + M + C). \quad (3)$$

Equations 1 to 3 show the relative importance of improving a given architecture to obtain the maximum gain while executing the modeled object. This way, if an application has a APC of 0.7, this means that it is control-dominated, and there is no point in using a DSP processor to implement it.

A group of objects can also be mapped to a single processor so that the application may fit in a smaller number of processors. However, all actions that the design requires to run in parallel must be allocated to different processors.

3.2 Processor analysis

In order to implement the processor selection procedure, processors that are available in the library must be pre-characterized. Some of the processor characteristics that are analyzed and included in each virtual machine are: the size of binary word; types of instructions; memory operand accessing modes; number of busses to access memory; execution time of each instruction; type of memory; number of busses to access memory; number of registers; control instructions; use of pipeline and depth of eventual pipeline; and use of harvard architecture or not.

These characteristics provide a high-level abstraction of a processor from a behavioral point of view. They can also be used to classify application-specific processors, like those devoted to DSP.

We have characterized 3 different processors belonging to the three different architecture families, so that one could have an idea of different performance metrics. Processors described in the library are the 8051 microcontroller (Int, 1985), the C25 digital signal processor (Tex, 1997), and the Risco microprocessor, a 32-bit RISC-like microcontroller (Car, 1996). Table 1 shows some of the processor characteristics stored in the library.

For example, since the C25 has a DSP architecture, memory accesses and computations take the same amount of cycles. This favors data-intensive applications. On the other hand, a RISC machine with many registers favors computations with few memory accesses. At the same time, the cost of a branch is higher in the C25, due to the effect of the possible pipeline flush. The added cost of the flush is considered in the table. In the C25, internal memory is considered as a register bank, due to its small access time and special indexing registers available in the architecture.

Table 1 – Partial processor characteristics

	#of registers	Jump Cycles	Mem access cycles	# of Busses	Operation cycles	Power (mW)
8051	8	3	2	1	3	40
Risco	32	1+2	2	1	1	90
C25	544	4+2	1(internal)	2	1	210

3.3 Processor selection

For each processor in the library we must obtain its Performance Factor regarding the application. Performance Factors are given by the following equations:

$$PFP_i = P_i / (P_i + M_i + C_i), \quad (4)$$

$$PFM_i = M_i / (P_i + M_i + C_i), \quad (5)$$

$$PFC_i = C_i / (P_i + M_i + C_i), \quad (6)$$

where the index i stands for a certain processor, and P_i , M_i and C_i are the relative costs of the processor instructions to execute data transformation, memory accesses and control operations, respectively. These costs depend on the processor characteristics, as introduced in the previous section. In this analysis, each 3-address code instruction of the application is assumed to generate a single instruction in the target processor.

The simplest way to choose a processor would be to pick the one that, in the critical characteristic of the application, has the smallest Performance Factor. This would mean that, when executing, the processor would have a good performance while executing the critical part of the required code. This simplification could lead, however to a non-optimal solution. There might be applications where the difference between Performance Factors could be small, or with complementary characteristics (example: processor P1 with $PFP_1=0.6$, $PFM_1=0.1$ and $PFC_1=0.3$, against processor P2 with $PFP_2=0.6$, $PFM_2=0.3$ and $PFC_2=0.1$). Our solution thus considers a right balance of the three factors.

Consider a particular application, for which APP, APM and APC have been calculated. Consider also a processor whose Performance Factors for this application have been calculated. The Application Performance Distance (APD) for this pair {application x processor} is obtained by the following distance measure:

$$APD = \sqrt{(PFP_i - APP)^2 + (PFM_i - APM)^2 + (PFC_i - APC)^2} \quad (7),$$

where index i stands for a certain processor. For this analysis, it is assumed that each 3-address code instruction generates a single instruction in the target processor. Equation 7 shows how distant is the processor from the ideal virtual machine that can execute the code. The processor with the smallest distance will be probably best suited to execute the application, since it has a small overhead considering the three types of instructions. Moreover, to evaluate the execution time of the target processor, we take the instructions executed in the virtual machine, the clock

frequency of each processor and the number of cycles each processor takes to execute an instruction. This gives a rough estimate of the processor performance, enough to decide whether the processor is suited to work in the required timing or not.

In the design of embedded systems, however, performance is not the only issue. For certain applications, there are many processors that could achieve the required performance. Other important aspects must come into play, like power dissipation and area. Moreover, an important point regarding reuse is the ability to answer the question "can a specific board or SOC be reused in this new application?". To answer this, the CAD system must evaluate all other aspects. In case more than one processor executes the object code in the required time, other aspects like power and area of the processor families may be compared, so that the best solution regarding all system aspects is achieved. In this work, area is evaluated in terms of FPGA cells used for the design of each processor core.

4. Results

In order to illustrate the concepts presented in this paper, we have applied the processor selection methodology to various examples, as shown in Table 2. In all cases, the whole system functionality has been implemented by a single object, so that a single processor has been selected to implement the function. Biquad is the classical biquad filter, while scrambler, descrambler, coder and decoder are part of a modem system, as well as echo-canceller. OCR is a neural network devoted to character recognition. The Podos system is an integrated circuit that measures the distance a person walks or runs. It is placed on the shoe and communicates with a display on the person's wrist. The computation of the distance is based on the double integration of acceleration in two axes.

Somewhat larger examples are the Crane Control and the Translating Pen. The first one has been proposed in (Neb, 1999) as an attempt of benchmarking in the area of system-level modeling and synthesis. The physical plant is composed of a crane with a load, moving along a track. The modeling of the physical system is done by a set of differential equations, which describe the behavior of the crane with a load and external forces being applied. The control algorithm of the Crane is implemented as a discrete computation of the state-variable method (Wag, 1999).

The Translating Pen has an optical sensor that slides over characters, finding words that are translated in a dictionary. For this example, two objects were modeled, splitting the system into the optical character recognition part, which uses a neural network, and the list processing part, which uses a hash table to find the words in memory.

Results concerning the above examples can be found in Table 2. Two selection mechanisms have been applied. *Selection 1* is based on the calculation of APD, as explained in the previous section. In this mechanism, the maximum allowed values for execution time, power and area are used as requirements that must be met (*no* means no requirement). *Selection 2* verifies which processors match the maximum required execution time and selects, among those, the processor with minimum power and area requirements.

It can be noticed that, for the biquad 1 and Podos examples, the *Selection1* procedure chooses the 8051 microcontroller, although the C25 is the best processor regarding speed and has the smallest APD and the Risco is the second best choice. But both the C25 and Risco are excluded due to user-defined limitations either in power or in area. A similar case happens with the biquad 2, where C25 would be the best choice regarding the APD, but it is excluded because of area requirements and Risco is chosen.

When the *Selection 2* procedure is applied, in most cases the 8051 is chosen, because it matches the maximum execution time and has less power and area requirements than the C25 and Risco. The only exceptions occur with the filter and echo canceller examples, where the 8051 is excluded because of time limitations. In both cases Risco is then chosen, because it has less power and area requirements than the C25.

As it can be seen, S^3E^2S can not only guide the design process, but it can also help the designer in the specification phase for buying an IP or in the development of a new architecture. After processor selection, the C code for the dedicated processor is generated, and a dedicated commercial compiler is used to obtain the final object code.

Table 2 - Results

Application	APD 8051	APD Risco	APD C25	max. time	max. power	max. area	selection 1	selection 2
Biquad1(1)	0.91	0.55	0.00	1	50	no	8051	8051
Biquad2 (2)	0.91	0.55	0.00	1	no	1050	Risco	8051
Filter (3)	0.74	0.42	0.37	0,02	no	no	C25	Risco
Podos (1)	0.64	0.31	0.35	10	50	no	8051	8051
Crane	0.37	0.21	0.43	2	no	no	Risco	8051
Pen-Hash	0.67	0.08	0.42	1	no	no	Risco	8051
Pen-OCR	0.16	0.49	0.41	1	no	no	C25	8051
Echo-canc.(3)	0.76	0.12	0.46	0,02	no	no	Risco	Risco
Scrambler	0.69	0.16	0.29	0,0125	no	no	Risco	8051
Descrambler	0.78	0.14	0.26	0,0125	no	no	Risco	8051
Coder	0.77	0.26	0.23	0,02	no	no	C25	8051
Decoder	0.37	0.33	0.39	0,02	no	no	Risco	8051

Time is given in ms; power is given in mW; area is given in LE's for an FPGA.

(1) C25 and Risco do not match power.

(2) C25 does not match area.

(3) 8051 does not match time.

5. Conclusions and future work

An integrated CAD environment for embedded systems must consider important aspects such as system specification, validation, and synthesis. Various different approaches have been proposed in the literature to cope with these issues. Most

environments have a fixed target architecture, consisting of a single processor and maybe some peripheral ASICs. These synthesis approaches concentrate on the task of partitioning system functions among hardware and software. S^3E^2S , in turn, performs a synthesis that is based on a library of processors, ranging from microcontrollers to ASIPs and DSPs. Each processor is characterized by a set of parameters, and the environment tries to match each object of the application (considering the application profile) to the most adequate processor. The final architecture is therefore a multi-processor platform.

Future work includes: the expansion of the processor library; the development of larger examples that really require multi-processor platforms; the generalization of the co-simulation mechanism in order to allow the integration of other specification languages, as in (Hes, 1999); and the development of algorithms to explore grouping of objects into a single processor, considering various quality metrics, as in (Dia, 1999). Other very important topics in the context of distributed embedded systems must still be considered in the future: the synthesis of the communication between processors and the synthesis of the operating system for objects executing various functions. Following an approach similar to that used for the processor selection, we intend to develop a communication synthesis mechanism based on a library of protocols, as in (Hes, 1999).

References

- Adams, J.K.; Thomas, D.E. (1996). The Design of Mixed Hardware/Software Systems. In: ACM/IEEE Design Automation Conference, Las Vegas, 1996, **proceedings** ... ACM Press, 1996. pp. 515-521.
- Böttger, J. et al. (1998). An Object-Oriented Model for Specification, Prototyping, Implementation and Reuse. In: Design, Automation and Test in Europe, 1998, **proceedings**... IEEE Computer Society Press, Los Alamitos, 1998. pp. 303-310.
- Hessel, F. et al. (1999). MCI – Multilanguage Distributed Co-Simulation Tool. In: **Distributed and Parallel Embedded Systems**, edited by F.Rammig. Kluwer Academic Publishers, 1999, pp 191-200.
- Jerraya, A.A.; Ernst, R. (1999). Multi-language System Design. In: Design, Automation and Test in Europe, Munich, 1999, **proceedings**... IEEE Computer Society Press, Los Alamitos, 1999. pp. 696-699.
- Kalavade, A.; Lee, E. (1995). Hardware/Software Codesign Using Ptolemy. In: **Codesign**, edited by J.Rozenblit and K.Buchenrieder. IEEE Press, 1995, New York. pp. 397-413.
- Moser, E.; Nebel, W. (1999). Case Study: System Model of Crane and Embedded Control. In: Design, Automation and Test in Europe, Munich, 1999, **proceedings**.... IEEE Computer Society Press, Los Alamitos, 1999. pp. 721-723.
- Nebel, W.; Gorla, G. (1999). JAVA, VHDL-AMS, ADA or C for System Level Specification? In: Design, Automation and Test in Europe, Munich, 1999, **proceedings**... IEEE Computer Society Press, Los Alamitos, 1999. p. 720.
- Pino, J.Z.; Kalbasi, K. (1998). Cosimulating Synchronous DSP Applications with Analog RF Circuits. In: Thirty-Second Asilomar Conference on Signals, Systems and Computers, 1998.
- Wolf, W. (1999). An Architectural Co-Synthesis Algorithm for Distributed, Embedded Computing Systems. **IEEE Transactions on VLSI Systems**, vol. 5, n.2, June 1997. pp. 218-229.

- Woo, N.; Dunlop, A.; Wolf, W. (1997). Codesign from Cospecification. **IEEE Computer**, Jan. 1997. pp 42-47.
- Aiguier, M. et al. (1997). ECOS: A Generic Codesign Environment for the Prototyping of Real Time Applications. In: **Hardware/Software Co-Design and Co-Verification**. Edited by M.Bergé, O.Levia and J.Rouillard. Kluwer Academic Publishers, 1997. pp 23-58.
- Wagner, F.; Oyamada, M.; Carro, L.; Kreutz, M. (1999). Object-oriented Modeling and Co-simulation of Embedded Systems. In: IFIP X International Conference on VLSI, Lisbon, 1999, **proceedings ...** IEEE Computer Society Press, Los Alamitos, California, 1999.
- Mrva, M.; Buchenrieder, K.; Kress, R. (1998). A Scalable Architecture for Multi-Threaded JAVA Applications. In: Design, Automation and Test in Europe, 1998, **proceedings...** IEEE Computer Society Press, Los Alamitos, 1998. pp. 868-874.
- Chiodo, M. et al. (1994). Hardware Software Codesign of Embedded Systems, **IEEE Micro**, vol. 14, n.4, Aug. 1994. pp. 26-36.
- Suzuki, K; Sangiovanni-Vincentelli, A. (1996). Efficient Software Performance Estimation Methods for Hardware/Software Codesign. In: ACM Design Automation Conference, Las Vegas, 1996, **proceedings ...**, ACM Press, 1996. pp. 605-611.
- Intel. (1985). Microsystems Components Handbook, Volume I, 1985.
- Texas Instruments. (1997). TMS320Cxx User's Guide. 1997.
- Carro, L. et al. (1996). Embedded Systems Design with Frontend Compilers. In: IEEE International Conference on Computer Design: Austin, 1996, **proceedings...** IEEE Press, Los Alamitos, 1996. pp. 200-205.
- Dias, O.P; Teixeira, I.M.; Teixeira J.P; Pereira, C.E. (1999). An OO-Based Methodology for Real-Time Hw/Sw Systems Modeling. In: **Distributed and Parallel Embedded Systems**, edited by F.Rammig. Kluwer Academic Publishers, 1999, pp 213-222.