The ITB Journal

Volume 4 | Issue 1 Article 5

2003

Hardware/Software Codesign

Richard Gallery

Hardware/Software Codesign

Richard Gallery & Deepesh M. Shakya
School of Informatics and Engineering, ITB
{Richard.Gallery,Deepesh.Shakya@itb.ie}

Introduction

The current state of the art technology in integrated circuits allows the incorporation of multiple processor cores and memory arrays, in addition to application specific hardware, on a single substrate. As silicon technology has become more advanced, allowing the implementation of more complex designs, systems have begun to incorporate considerable amounts of embedded software [3]. Thus it becomes increasingly necessary for the system designers to have knowledge on both hardware and software to make efficient design trade-offs. This is where hardware/software codesign comes into existence.

Hardware/software codesign is the concurrent design of both hardware and software of the system by taking into consideration the cost, energy, performance, speed and other parameters of the system. During the design, trade-offs are made between the implementation of functionality in hardware and/or software depending upon both cost considerations and technical feasibility.

Since the concept of hardware/software codesign surfaced in 1990s [1], different methodologies have been proposed for hardware/software codesign. This article gives an overview of hardware/software codesign. In section 2, a generic hardware/software codesign methodology is described, section 3 describes the taxonomy of hardware/software codesign where different aspects of hardware/software codesign is discussed along with some works performed in these arena to date, section 4 gives an introduction of different codesign methodologies widely accepted in the literature.

Generic Hardware/Software Codesign Methodology

In this section a generic methodology for hardware/software codesign (
Figure ***1***) is discussed. The initial step in hardware/software codesign is the high level specification of the system behaviour to include the functionality, performance, cost, power and other constraints of the expected design. The specification step includes modelling of the system in order to capture the entire characteristics of the system.

After the system specification is ready, it is divided into a number of blocks to allow a costing, through the application of cost metrics[2] for each of these blocks. This is performed in the Cost Estimation step where the estimation is done for both hardware and software implementation. This is actually the step for analysis and estimation. The system is analysed from different aspects of its cost metrics. This step provides valuable information for the hardware/software partitioning.

The next stage is to partition the system functionality between hardware and software. The partitioning phase takes information collected from the Cost Estimation phase to allow decisions to be taken as to which block is to be mapped on hardware and which block to be mapped on software. The quality of such mapping depends on how much the design constraints have been

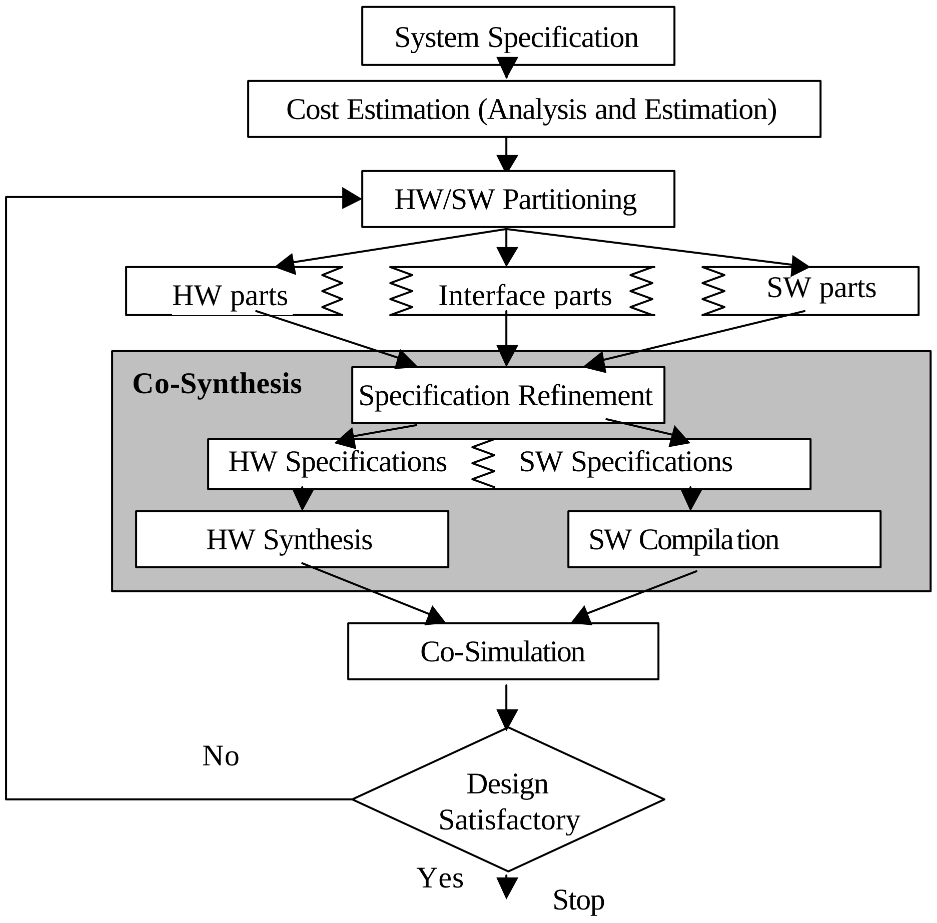

Figure 1 Generic Hardware/Software Codesign Methodology [2]

[2] Cost metrics can be calculated for both hardware and software. Hardware cost metrics can be, for e.g., execution time, chip area, power consumption or testability. Software cost metrics can be, for e.g., execution time, program and data memory.

achieved and how much the design cost is minimized [3]. If the design constraints[3] are not met then the expected performance of the system cannot be met and if the design cost is not minimized, the design cannot compete in the market.

The system is then designed within the context of the heterogeneous target architecture[4]. This requires the specification of the interfaces (communication and synchronization) between hardware represented by ASICs and software represented by the processors.

Once hardware/software blocks and the interfaces between them have been decided, the next step is the co-synthesis. In co-synthesis, specification refinement is done, where the implementation independent system specification is transferred into hardware and software specifications (and the specific ations for the interfaces).

Once the separate specification and the necessary refinement[5] in the hardware design are carried out, hardware is synthesised that gives a set of ASICs and software is compiled for the target processor.

Once the synthesis step is over, the next step in the design flow validates the system design simulating both the ASIC and the processor together. This is called the co-simulation. The co-simulation checks whether the design goal has been achieved or not. If the design is acceptable, the codesign flow stops. If the design is not acceptable the design is traced back to the hardware/software partitioning step and the design cycle is repeated until the satisfactory design output is obtained [2].

Taxonomy of Hardware/Software Codesign

Hardware/Software has the following important aspects [3] which must be considered for an effective system design.

- Modelling
- Analysis and Estimation
- System Level Partitioning, Synthesis and Interfacing
- Implementation Generation
- Co-Simulation and Emulation

[3] There can be different design constraints for e.g. time, area, power, memory etc. Timing constraints specifies timeline for the execution of the system task.
[4] System architecture consisting of both hardware (ASICs) and software (general processor).

Modelling

Modelling can be regarded as the science of capturing the system characteristics [3]. Models should capture all the information which is necessary for the designers.

Edwards et al.[4] explores the various computational models in the embedded system design, in which they stress that the formal model should include:

- Formal specification (relation between input and output and internal states),
- Set of properties[6] (a set of relation between input and output and internal states. This is explicitly mentioned to verify it against the functional specification. The properties are for assertion of the behavior rather than description of the behavior.)
- Performance indices (e.g. cost, reliability, speed, size etc.)
- Design constraints (on performance indices).

The functional specification fully characterizes the system while satisfying the set of properties.
The design process starts by modelling the system at a high level of abstraction. The designer checks whether a set of properties have been verified, performance indices are satisfactory and the design constraints are met.

Edwards et al.[4] goes on to recognize four different types of Models of Computation[7]:

- Discrete Event,
- Communication Finite State Machines (FSM)
- Synchronous/Reactive
- Dataflow Process Networks Models.

A discrete event (DE) model is characterized by events which are time-stamped (i.e. the time at which event occurs is timestamped). A DE simulator requires a global event queue which keeps track of the time-stamped events and orders them to be executed according to their time-stamps. The DE approach is used to simulate the digital hardware [5].

---

[5] An act of adding design details or converting from abstract representation to Register Transfer Level (RTL) ready to be fed into the hardware synthesis tool.
[6] It can be property of determinate behavior i.e. the output of the system depends entirely on its input and not on some internal hidden factors.
[7] A system can be thought of as composing of simpler subsystems, or pieces. The method or the rules for composing and capturing these pieces to create a system functionality is called models of computation.

FSMs are good for modelling sequential behaviour but are not suitable for modelling concurrent behaviour of a group of machines, as it may reach the point state explosion[8]. This is because the number of states will be the product of the number of states in each machine.

The synchronous/reactive model consists of events which are synchronous i.e. all signal have events at clock tick. The simulators that use the synchronous models are called cycle-based or cycle-driver simulators.

In the dataflow model, there is a directed graph where the nodes represent computations and ordered sequences of events which is represented by arcs [6].

The above discussed modelling techniques may be deployed through the use of appropriate modelling or specification languages. For example designers using SDL[9] (a state oriented language) [7] can describe multiple concurrent processes[10] which communicate with signals. StateCharts [8] has the ability to decompose a sequential model into hierarchical structure thus facilitating the representation of FSMs in Statecharts. This hierarchical decomposition can solve the problem of state explosion [11][3]. The Esterel [9] language is a synchronous language that supports concurrent behaviour and thus makes it suitable for modelling in FSMs [9]. SpecCharts [73] exploits the advantage of hierarchical and concurrent state diagrams and the hardware description language VHDL [61].

Specification languages

Specification language describes the overall goal of the desired functionality of a system. A good specification language is able to address different aspects of a system which includes following [2] [52] [54]:

- Concurrency
- State-transitions
- Hierarchy

[8] The number of state grows exponential that makes the design complex enough to be handled.
[9] Specification and Description Language
[10] Process is a program codes consisting of sequence of statements.
[11] States contained within a state are called sub-states of this surrounding state. The surrounding state is higher in the hierarchy. If there are two machines with 4 states each then if these machines are combined to form a single machine then the total number of states of the combined machine will be the permutation of the number of states of each machine. If in any case, all the states (both machines combined) can be arranged in an hierarchical manner for e.g. all four states of one machine can be considered as substates of one of the state of another machine then these substates will have nothing to do with other states of the machine hence the total number of possible number of states is reduced.

- Programming constructs
- Behavioral completion
- Communication
- Synchronization
- Exception Handling
- Non-determinism
- Timing

Concurrency: Parts of an embedded system work in parallel. Parts may be a process and threads of the process. A specification language should be able to capture the concurrency[12] behavior of the system.

State-transitions: Systems are often conceptualized as having various modes or states, of behavior. In a system with multiple states, the transition between states occurs in undefined or unstructured manner. The specification language must be able to model such arbitrary transitions.

Hierarchy: A system can be conceptualized as a set of smaller subsystems if modelled as hierarchical models. Such conceptualization helps system designer to simplify the development of a conceptual view of a system, since parts of the system can be treated as a separate unit paving way for scoping objects, such as declaration types, variables and subprogram names. Lack of hierarchy will make all such objects global and it becomes increasingly difficult for a designer, as the system will become more complex. There are two types of hierarchy: structural hierarchy and behavioral hierarchy. *Structural hierarchy* enables designer to design a system with interconnected components. Each component is themselves a composition of sub-components. *Behavioral hierarchy* decomposes a system behavior into distinct sub behaviors for e.g. procedures or functions.

Programming Constructs: Specification language should have programming constructs, for e.g. constructs like functions, procedures, loops, branches (if, case) and assignments simplifies the sequential description of the system behavior.

Behavioral Completion: A specification language should be able to model the behavioral completion to indicate that the behavior has completed the execution of all computations. An advantage of behavioral completion is that it allows designer to conceptualize the behavior as an independent module. The designer may start next behavior in sequence once the preceding behavior has finished without worrying if any unfinished work remained in that behavior.

Communication/Synchronization: A system has several processes working concurrently. These processes need to communicate with each other. A specification language should have an ability to model the communication between concurrent behaviors or processes and at the same time should ensure the synchronization of two behaviors or processes. Such communication can be conceptualized as a shared memory[13] or message passing[14] paradigms. In shared memory, the sending process writes to a medium which is also accessible to the receiving process. Such a medium could be global variables or ports. In message-passing communication model, the communication is accomplished with an abstract medium called channels with send/receive primitives.

Exception handling: A specification language should be able to model the exception handling mechanism, for e.g. when an exception occurs in the form of interrupts or resets, the current state of the system should be terminated and the transition to the new state is required. Such reactive behavior is quite common in the embedded system.

Non-determinism: Non-determinism is helpful when the designer doesn't want to take decision during the specification, for e.g. if two events occurs simultaneously then the designer can leave the decision of which event to be executed first at the time of implementation. This is only possible if the specification language has an ability to model the non-determinism.

Timing: Specification language should have an ability to model the timing aspects of the embedded system which are the functional timing and the timing constraints. *Functional timing* represents a time required to execute a behavior. *Timing constraints* indicate a range of time within which a behavior has to be executed.

The specification languages have been categorized into the following categories as presented in [2].

1. Formal Description Technique (for e.g. LOTOS [56], SDL [55], Estelle [57])
2. Real Time System Languages (for e.g. Esterel [58], Statecharts [59], High-Level Time Petri Nets [60])

---

[12] The act of two processes running concurrently.
[13] Memory in a parallel computer, usually RAM, which can be accessed by more than one processor, usually via a shared bus or network. It usually takes longer for a processor to access shared memory than to access its own private memory because of contention for the processor-to-memory connections and because of other overheads associated with ensuring synchronised access.
[14] A message passing system has primitives (for e.g. send ( ) and receive ( )) for sending and receiving messages. These primitives can be either synchronous or asynchronous. In synchronous message passing, sending and receiving of message is not complete unless receiving end acknowledges the receipt of the message. In asynchronous message passing, the message sending process in complete once the message is sent irrespective of whether the message has been received by the receiving end or not.

3. Hardware Description Languages (for e.g. SpecCharts [73] [54], VHDL [61][61], Verilog [62], HardwareC [44], Handel-C[70])
4. Programming Languages (for e.g. SpecC [63] [64] [65] ,C^x [19])
5. Parallel Programming Languages (for e.g. CSP [66], Occam [67])
6. Data Flow Languages (for e.g. Silage [68] [2])
7. System Design Language (for e.g. SystemC [69])

Analysis and Estimation

It becomes necessary for designers to take crucial decisions during the codesign process and in order to take these decisions a designer requires:

- Application domain knowledge (ideally the designer understands the application domain in which the technology will be deployed)
- Knowledge of the technology options that are available
- The ability to analyse proposed design solutions (and as a result access to, training and knowledge of the capabilities and limitations of design tools)

The analysis and estimation of the design become more crucial when the design constraints require fast timing and high power consumption [3]. Correct procedures in the design process can avoid non-competitive and costly designs.

There are different analysis types that need to be made in the design [3], including, amongst others:

- Process path analysis
- Architecture modelling and analysis
- Power analysis
- Multiprocessor analysis rate analysis
- memory system analysis

A process path is a sequence of process statements that is executed for given input data [3]. *Process-path analysis* corresponds to determining a set of possible process paths. By examining the possible process paths, it is possible to find out the worst case execution time (WCET) by first tracing the process paths during worse case and then calculating the WCET. Li et al. extensively discusses the process path analysis in [10].

Power analysis consists in determining the power cost of the system. Tiwari et al. in [11] describes a power analysis technique to generate the power cost model for the embedded software. Fornaciari et al. in [12] introduces power metrics included in a hardware/software codesign environment to guide the system level partitioning. Yanbing Li et al. in [13] explores a framework for optimizing the system parameters to minimize energy dissipation of the overall system i.e. both hardware and software. The paper also explores the trade-off between system performance and the energy dissipation.

Rate analysis includes the analysis of execution rate of the processes. The rate constraints, imposed by the designer in order to assure proper working of the system to its environment, is one form of the timing constraints [14]. Mathur et al. in [14] proposes an interactive rate analysis framework to make sure all the rate constraints are satisfied in the design of the system.

Memory system analysis is also an important factor in the embedded system design. Specially, in the areas of image and video processing systems, 50-80% of area cost of the ASICs for real-time multidimensional signal processing is due to the data storage and transfer of array signals [15]. So it becomes increasingly necessary to estimate the memory usage and optimize them well before any decision on hardware software partitioning is made.

Multiprocessor analysis deals with the estimation and analysis of parallel process execution. Process scheduling decides the order of process execution.

System-Level Partitioning, Synthesis, and Interfacing

Partitioning

Hardware/software partitioning takes place after the necessary information on cost metrics is generated from the analysis and estimation of the design. Based upon this information, the system is divided into hardware and software according to whichever gives the best overall performance result.

Various algorithms have been developed for the hardware/software partitioning. Gupta and DeMicheli [16] [17] created an algorithm to automate a search of the design space for the hardware/software partitioning. The algorithm starts by implementing all functionalities in hardware and the operations to be moved into software are selected based on the cost criterion of communication overheads. Movement into software is only done if there is any improvement in the cost of the current system partition. The algorithm iterates the process of movement until no cost-improving move could be found. The main defect in this algorithm is that the algorithm frequently created very costly hardware that consumes many resources, since the initial partition starts with the hardware solution [18]. Authors in [17] depict the use of their algorithm for describing the implementation of a network coprocessor communication via an ethernet[15] link. This co-processor is used to take load off the CPU to handle the communication activities.

Ernst and Henkel [19] start with the initial partition in software and gradually transfer the software part into hardware. Ernst and Henkel used a hill-climbing[16] partitioning heuristic, an example of which is the simulated annealing [53]. This algorithm uses a cost function to minimize the amount of hardware used with the performance constraints remaining intact. Simulated annealing in [19] starts with an infeasible solution with a high cost penalty for run time exceeding timing constraints. Then the algorithm searches for an improved timing and a steep decrease in the cost. Ernst and Henkel in [19] uses their algorithm for the hardware/software partitioning of the digital control of a turbocharged diesel engine and a filter algorithm for a digital image in which they got a speed up of *1.4* and *1.3* respectively in reference to the implementation in software alone.

Synthesis and Interface

Once the partitioning specification is ready, the next step is the synthesis of hardware, software and their respective interfaces. In other words, the co-synthesis step follows next after the hardware software partitioning. Co-synthesis is defined as the synthesis of hardware, software and the interface between hardware and software. Once the synthesis is complete then the design is subjected to co-simulation.

The final synthesized system architecture generally comprises of: a programmable processor, one or more hardware modules all of which are connected through a system bus, and the appropriate software modules and interfaces. Hardware modules consist of a datapath, a controller and I/O interface between hardware and the processor. The processor runs the

[15] Ethernet is a physical and data link layer technology for LAN networking.
[16] Hill-climbing algorithms are neighborhood search algorithms that subsequently select the neighbor with the highest quality and continue from there. The search terminates when no neighbor exists that represents an improvement over the current solution. Hill-climbing algorithms belong to the class of greedy algorithms i.e. the algorithm never goes back to a solution with lower quality. In other words, the climber never goes downhill to finally reach a higher peak [55].

software component of the architecture and also includes the device drivers to establish communic ation between software and hardware (the hardware/software interfaces).

In [20], an environment is described for the specification and the synthesis of a heterogeneous system using Cosmos[17]. Design starts with an SDL[18] [47] specification and produces a heterogeneous architecture comprising hardware in VHDL and software in C. Codesign steps in Cosmos includes: partitioning, communication synthesis and architecture generation. Communication synthesis consists in transferring the process that communicates with high-level primitives[19] through channels into signals. Architecture generation here is actually an implementation generation step discussed in the next section. Architecture generation includes two major tasks i.e. virtual prototyping and architecture mapping. Virtual prototyping consists of hardware (in VHDL), software (in C) and communication (in VHDL or C) which can be simulated. Architecture mapping consists of synthesizing VHDL descriptions into the ASICs, conversion of software parts into assembly code resulting in the final architecture that consists of software, hardware and the communication components.

The software design and the software synthesis are also an important aspect of the hardware/software codesign since a significant part of the system (i.e. the system that consists of both hardware and software) are implemented in software. Software synthesis focuses on the support of embedded systems without the use of operating systems[20] [21].

Implementation Generation

Implementation generation for hardware refers to generating hardware for a set of functions. Hardware typically consists of [24]:

- Control-unit/datapath
- Storage unit (for e.g. registers, register files and memories)
- Multiplexer

[17]Cosmos is a co-design methodology and tools aimed at the design and synthesis of complex mixed hardware-software systems.
[18] Specification Description Language.
[19] Primitives are the basic operations. High level primitives for the communication between two processes can be taken as the communication that occurs by calling functions.
[20] The main drawback of using the support of operating system is that most kernels tend to use a fixed priority preemptive scheduling mechanism, where the timing constraint is realized from the process priorities. In some cases the timing constraint is realized by scheduling the process with information of process period, release time and deadline. But in embedded system, the timing constraints should be realized more on the occurrence of the events. Since, the operating system scheduling mechanism doesn't have idea on the time stamp; it doesn't know when the events are generated. [21]

- State-register
- Control-logic

While generating hardware, the size of hardware should be as small as possible while maintaining the system constraints intact. An implementation, which is silicon area efficient, is thus a sign of quality design. Implementation generation for software refers to producing an assembly code for software. An efficient software implementation can only be realized if the compilers can take full advantage of the architectural features of the processor. Some approaches for exploiting architectural features are described below.

Sudarsanam et al. in [23] presents a retargetable[21] methodology in an effort to generate high quality code for a wide range of DSPs[22]. The paper describes a solution for the problems arising in those compiler technologies which are unable to generate dense, high-performance code for DSPs as they do not provide adequate support for the specialized features of DSPs. Also, the paper describes the solution for the problem where it is necessary to build a compiler from scratch, due to the unavailability of a suitable compiler (a time consuming process). The solution presented is a methodology for developing retargetable DSP compilation.

Vahid et al. in [24] describes an algorithm for estimating the hardware size. The paper describes an algorithm for the hardware estimator, whic h is based on incrementally updating the design model to acquire accuracy and iterative improvement algorithms to explore the different design possibilities. Hence, the algorithm maintains both speed and accuracy in estimating hardware size. The algorithm takes advantage of the fact that between two iterations of partitioning design there is only an incremental change. For this incremental change, a data structure (representing an incrementally modifiable design model) and an algorithm that can quickly provide the basic design parameters needed by the hardware-size estimator are developed. Therefore, whenever there is any incremental change in the design model, the corresponding hardware size is estimated.

[21] Retargetable means the reuse without little or no modification for e.g. retargetable compiler is able to generate code (maintaining the same quality) for the new processor after minor modifications without need of creating entirely new compiler from the scratch.

[22] Digital Signal Processor

Co-Simulation and Emulation

Co-simulation

Co-simulation of hardware and software refers to the simultaneous verification of hardware and software functions correctly [25]. The conventional co-simulation approach waits until the real hardware has been delivered and then performs verification by using in-circuit emulators[23]. Due to the increased complexity of the designs and the importance of verifying the system design as much as possible before committing to the (expensive) transfer of the hardware aspects of the system to silicon, it has become necessary to perform co-simulation before the real hardware is produced. This saves time-to-market as well as the cost required in debugging and re-building the hardware. Rowson in [25] gives an overview of the techniques available for hardware/software co-simulation.

Ghosh et al. in [32] describes a hardware-software co-simulator that can be used in the design, debugging and verification of embedded systems. This tool consists of simulators for different parts of the system (for e.g. Clock Simulator, Parallel Port Simulator, UART simulator, CPU Simulator, Timer Simulator, Memory Simulator etc.) and a backplane[24] which is responsible for integrating all the simulators. The back plane is represented by Simulation Manager which manages communication between the co-simulators (e.g. CPU simulator, Memory Simulator etc.) and the virtual instruments. Virtual instruments are used to provide stimulus and to observe response. The paper describes a tool that provides an environment for joint debugging of software and hardware and is also capable of evaluating system performance, selection of algorithms and implementations. The tool also addresses the possibility of exploring hardware-software tradeoffs.

In [32], the performance of the tool for the applications (which was taken as an example) like engine control unit has been evaluated. The co-simulation of the engine control unit showed a slowdown by a factor of 400 which is quite suitable for debugging.

Valderrama et al. in [33] describes a unified co-synthesis and co-simulation methodology i.e. both the steps are performed using the same descriptions (in C and VHDL). The

[23] In-circuit emulators are used to replace the processor or microcontroller of the target hardware. It is a valuable software developers tool in embedded design. The developer loads the program into the emulator and can then run, step and trace into it.
[24] Simulation backplane controls all simulators coupled to it. If a simulator needs to communicate with partner-simulators, it does this through the simulation back plane.

communication between hardware and software is through a communication unit, which is an entity able to execute a communication scheme invoked through a procedure call mechanism [74]. The VHDL entity[25] is used to connect a hardware module with that of software. The use of procedure call mechanism hides the implementation details related to the communication unit. The access to the interface of the communication is done through the procedures. By employing this method, the two communicating modules become quite independent of each other and changes in one module need not change in other module unless the communication unit interface is being accessed using the same procedure before and after the change. The level of abstraction obtained by using procedures help in using the same module descriptions with different architectures (i.e. the architectures which varies depending upon the communication protocols used).

Emulation

Co-simulation uses an abstract model to form a virtual prototype (of hardware) while co-emulation provides a real prototype by implementing functions in hardware (for e.g. FPGA[26]).

Luis et al. in [34] describes the co-emulation process observed in the co-design methodology-LOTOS [56]. Once all the construction of the interface between hardware and software is completed, the execution of software (in C) and the simulation of hardware (in VHDL simulator) is performed on SUN workstation. The things that software requires to write or read into or from the FPGA (i.e. the hardware) is written into the files (through C functions) and the hardware simulator reads the files via an extra VHDL component in the interface, which is a wrapper[27] for a set of C functions that perform reading and writing operation on the files. This is to perform a test for errors before emulating hardware with the FPGA. The next step performed is the co-emulation where the hardware part is replaced by the FPGA.

Cedric et al. in [71] describes the co-simulation between SystemC and an emulator called ZeBu. The paper depicts how SystemC can be co-simulated with ZeBu at different level of abstraction i.e. at signal-level and at transaction level[28]. ZeBu is a hardware verification product built on a PCI card with Xilinx Virtex-II FPGA [72] devices. ZeBu consists of a technology called Reconfigurable Test Bench (RTB) that interfaces a design under test

[25] A modular representation of deign in VHDL is called an entity.
[26] Field Programmable Gate Array
[27] Wrapper is a piece of code which is combined with another code to determine how the latter code is executed. Wrapper actually acts as an interface between its caller and the wrapped code.
[28] The communication that takes place with function call.

(DUT). DUT is emulated by one or more Virtex-II FPGA devices. The main function of the RTB is to stimulate and monitor each individual I/O data pin of the DUT emulated by the FPGA. ZeBu also consists of C/C++ API which works in concert with the RTB providing direct interaction with the test benches modelled at higher level of abstraction via SystemC. In the paper, a test case is presented in which the co-simulation for a graphics design is conducted for three different cases: SystemC model and HDL[29] (Verilog), SystemC model and ZeBu at the signal level and SystemC model and ZeBu at the transaction level. SystemC models consist of test bench that interacts with the emulated hardware. The result shows that the co-simulation execution time for SystemC/HDL is *3* days (for that particular test case considered), SystemC/ZeBu at signal level is *330* seconds and SystemC/Zebu at the transaction level is *5* seconds. This co-simulation process with emulated hardware is one of the latest technologies in the literature. The main benefit of this technique is its ability to co-simulate at transaction level that gives significant speed-ups.

Co-design Systems

In section 0, a generic hardware/software codesign methodology was presented. In this section different codesign approaches will be introduced. An interested reader on particular codesign system may refer to the references given against a methodology name introduced here.

Ptolemy [35] is codesign methodology that allows heterogeneous specification[30] to develop a unified environment for creating heterogeneous systems. Castle [36] [37] is a codesign platform which puts more emphasis on processor synthesis i.e. starting from an application it ends up with synthesis of suitable processor design on which the application considered can be executed efficiently. Cosyma[31] [38] is a codesign methodology which starts the system solution all in software and during the partitioning step gradually ports software portion into hardware to achieve practically feasible design. Lycos[32] [39][40] is a codesign tool based on a target architecture with the single processor and a single ASIC. Lycos stresses design space exploration[33] with automatic hardware/software partitioning. Tosca[34] [22][41] is a codesign

[29] Hardware Description Language to simulate the hardware.
[30] System specification with more than one specification language.
[31] Co-synthesis for Embedded Architectures
[32] Lyngby Cosynthesis
[33] Choosing one suitable design out of many.
[34] Tools for System Codesign Automation

methodology that is mainly targeted for control flow dominated[35] reactive real-time systems [2]. The target architecture in Tosca consists of off-the-shelf processors and a set of co-processors on a single chip. The design description is specified using C, VHDL or Occam [67]. Vulcan [42][43] is a hardware/software codesign tool focusing on the co-synthesis. The input specification to this codesign tool is the hardware description language, HardwareC [44]. The partitioning in Vulcan starts with a complete solution in hardware i.e. describing the entire solution in HardwareC [44]. Chinook [45] is a co-synthesis tool for embedded real time systems. Chinook focuses on the synthesis of hardware and software interface and communication. Cosmos [46] is a codesign environment in which the system description is specified in SDL[36] [47] and ends up by producing a heterogeneous architecture with the hardware descriptions in VHDL and the software descriptions in C. CoWare [48][2] is a codesign environment of a system supporting co-specification (heterogeneous specification), co-simulation and co-synthesis (heterogeneous implementation). Polis [49] is a framework for hardware/software codesign targeted for the reactive embedded systems[37]. The system is specified in the specification language called Esterel [50]. SpecSyn [51] is a codesign environment which supports a specify-explore-refine design paradigm i.e. the design starts with the specification of the system functionality and then the rapid exploration of numerous system level design options is performed. Once the feasible most option is selected then refinement is carried out for the chosen option

Conclusion

Hardware/software codesign is relatively a new topic but since its inception, its literature has grown to a wide range of arena and many researches have been conducted in this field. There is no standard co-design methodology which can be regarded as the most useful for every system design. All the methodologies that are available in the literature till date has its own advantages and disadvantages. In some cases, it only suits specific applications. Competitive product with low cost and less time to market is the manifestation of an efficient design methodology. However, methodology alone is not sufficient; it needs equally strong specification language, suitable model of computation, efficient compiler, efficient synthesis tool and the efficient co-simulation environment.

[35] System which is determined at run time by the input data and by the control structured (e.g. "if" statements) used in the program.
[36] Specification and Description Language
[37] Reactive systems typically respond to incoming stimuli from the environment by changing its internal state and producing output results [2].

References

[1]. Wayne Wolf, *A Decade of Hardware/Software Codesign,* Article from Computer, pp. 38-43, April 2003.

[2]. Ralf Niemann, *Hardware/Software Co-design for Data Flow Dominated Embedded Systems,* Kluwer Academic Publishers, 1998.

[3]. Jorgen Staunstrup and Wayne Wolf, *Hardware/Software Co-Design: Principles and Practice,* Kluwer Academic Publishers, 1997.

[4]. S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli, *Design of Embedded Systems: Formal Models, Validation, and Synthesis,* Proc. IEEE, vol. 85, pp. 366-390, Mar. 1997.

[5]. V. Lazarov and R. Iliev, *Discrete Event Simulation of Parallel Machines,* 2nd AIZU International Symposium on Parallel Algorithms / Architecture Synthesis, pp. 300, March 1997.

[6]. Edward A. Lee and Thomas M. Parks, *Dataflow Process Networks,* Proceedings of the IEEE, vol. 83, no. 5, pp. 773-801, May, 1995.

[7]. M. Daveau, G. Marchioro and A. Jerraya, *VHDL generationfrom SDL specification,* In: CHDL. pp. 182-201, 1997.

[8]. Harel, D., *Statecharts: A Visual Formalisms for Complex Systems,* Communications of the ACM Vol.31 No.5, 1988.

[9]. G. Berry and G. Gonthier, *The Esterel synchronous programming language: Design,semantics, implementation*, Science Of Computer Programming, 19(2):87–152, 1992.

[10]. Yau-Tsun Steven Li and Sharad Malik, *Performance analysis of embedded software using implicit path enumeration*, Proceedings of the 32nd ACM/IEEE conference on Design automation conference, USA, 1995.

[11]. Vivek Tiwari, Sharad Malik and Andrew Wolfe, *Power analysis of embedded software: a first step towards software power minimization*, IEEE Transactions on VLSI Systems, December 1994.

[12]. William Fornaciari , Paolo Gubian , Donatella Sciuto and Cristina Silvano, *Power estimation of embedded systems: a hardware/software codesign approach*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v.6 n.2, p.266-275, June 1998.

[13]. Yanbing Li , Jörg Henkel, *A framework for estimation and minimizing energy dissipation of embedded HW/SW systems*, Proceedings of the 35th annual conference on Design automation conference, p.188-193, California, United States, June 15-19, 1998.

[14]. A. Dasdan, A. Mathur, and R. K. Gupta. *RATAN: A tool for rate analysis and rate constraint debugging for embedded systems*. In Proceedings ED&TC '97, 1997.

[15]. Koen Danckaert, Francky Catthoor and Hugo de Man, *System level memory optimization for hardware-software co-design*, 5th International Workshop on Hardware/Software Co-Design (Codes/CASHE '97) Braunschweig, GERMANY, March 24 - 26, 1997.

[16]. R. Gupta and G. De Micheli, *Hardware-software cosynthesis for digital systems*, IEEE Design and Test of Computers, vol. 10, no.3, pp.29-41, Sept. 1993.

[17]. R.K. Gupta and G.D. Micheli, *System-level Synthesis using Re-programmable Components*, 1EEE/ACM Proc. of EDAC'92, 1EEE Comp. Soc. Press, pp. 2 -7, 1992.

[18]. Adam Kaplan, Majid Sarrafzadeh and Ryan Kastne, *A Survey of Hardware/Software System Partitioning*, (Details not available)

[19]. Rolf Ernst, Jorg Henkel and Thomas Benner, *Hardware-Software Cosynthesis for Microcontrollers*, Design and Test of Computers, pp. 64-75 ,Vol. 10, No. 4, October/December 1993.

[20]. Tarek Ben Ismail, Ahmed Amine Jerraya, *Synthesis Steps and Design Models for Codesign* ,Computer, Vol. 28, No. 2, pp44-52, February 1995.

[21]. Filip Thoen, Marco Cornero, Gert Goossens and Hugo De Man, *Real Time Multi-Tasking in Software Synthesis for Information Processing Systems*, Eighth International Symposium on System-Level Synthesis, Los Alamitos, 1995.

[22]. A. Balboni, W. Fornaciari, and D. Sciuto, *Co-synthesis and cosimulation of control dominated embedded systems*, in International Journal Design Automation for Embedded Systems, vol. 1, no. 3, July 1996.

[23]. Ashok Sudarsanam, Sharad Malik and Masahiro Fujita, *A retargetable Compilation Methodology for embedded Digital Signal Processors using a Machine-Dependent Code Optimization Library*, Design Automation for Embedded Systems, Kluwer Academic Publishers, pp. 187-206, 1999.

[24]. Frank Vahid and Daniel D. Gajski, *Incremental hardware estimation during hardware/software functional partitioning*, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.3 No.3, pp.459-464, Sept. 1995.

[25]. J. Rowson, *Hardware/software co-simulation*, In Proc. of the Design Automation Conference, pp. 439--440, 1994.

[26]. Heiko Hubert, *A Survey of HW/SW Cosimulation Techniques and Tools*, Thesis work, Royal Institute of Technology, Sweden, 1998.

[27]. Triscend, *Bus Functional Model*, Application Note,(AN_32),v1.4, July 2001.

[28]. Virtutech White Papers, *Introduction to Simics: Full System Simulator Without Equal*.

[29]. Texas Instruments, *TMS320C28x DSP CPU and Instruction Set-Reference Guide* , Literature Number: SPRU430B, August 2001 – Revised May 2002.

[30]. Vojin Živojnovic and Heinrich Meyr, *Compiled HW/SW co-simulation*, Proceedings of the 33rd annual conference on Design automation conference, pp.690-695, USA, June 03-07, 1996.

[31]. Chris Schlager, Joachim Fitzner and Vojin Zivojnovic, *Using Supersim Compiled Processor Models For Hardware, Software And System Design*, (Details not available)

[32]. A. Ghosh, M. Bershteyn, R.Casley, C. Chien,A. Jain, M. LIpsie, D. Tarrodaychik, and O. Yamamoto, *A Hardware-Software Co-simulator for Embedded System Design and Debugging*, In proceedings of ASP-DAC'95.

[33]. C. A. Valderrama, A. Changuel, P. V. Raghavan, M. Abid, T. B. Ismai and A. A. Jerraya, *Unified Model for Co-simulation and Co-synthesis of mixed hardware/software systems,* Proc. EDAC'95, Paris, France, February- March 1995.

[34]. Luis Sanchez Fernandez,Gernot Koch, Natividad Martinez Mardrid Maria Luisa Lopez Vallejo, Carlos Delgado Kloos and Wolfgang Rosenstiel, *Hardware-Software Prototyping from LOTOS*, Design Automation for Embedded Systems, Kluwer Academic Publishers, 1998.

[35]. Edward A. Lee, *Overview of the Ptolemy Project*, Technical Memorandum UCB/ERL M01/11 March 6, 2001.

[36]. P.G. Plöger and J. Wilberg, *A Design Example using CASTLE Tools*, Workshop on Design Methodologies for Microelectronics, Institute of Computer Systems Slovak Academy of Sciences, Bratislava, Slovakia, pp. 160-167, Sep., 1995.

[37]. J. Wilberg, A. Kuth, R. Camposano, W. Rosenstiel and H. T. Vierhaus, *Design Space Exploration in CASTLE*, Workshop on High-Level Synthesis Algorithms, Tolls and Design (HILES), Stanford University, Nov. 1995, in: GMD-Studie Nr. 276, Dec. 1995.

[38]. Achim Osterling, Thomas Benner, Rolf Ernst, Dirk Herrmann, Thomas Scholz and Wei Ye, *The Cosyma System*, a chapter from Hardware/Software Codesign: Principles and Practice, pp 263-282, Kluwer Academic Publishers, 1997.

[39]. J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen and A. Haxthausen, *LYCOS: the Lyngby Co-Synthesis System. Design Automation of Embedded Systems*, Vol. 2, No. 2, March 1997.

[40]. Achim Osterling, Thomas Benner, Rolf Ernst, Dirk Herrmann, Thomas Scholz and Wei Ye, *The Lycos System*, a chapter from Hardware/Software Codesign: Principles and Practice, pp 283-305, Kluwer Academic Publishers,1997.

[41]. W. Fornaciari, D. Sciuto and A. Balboni, *Partitioning and Exploration Strategies in the TOSCA Co-Design Flow*,4th International Workshop on Hardware/Software Co-Design (Codes/CASHE'96),Pittsburgh, Pennsylvania , 1996.

[42]. R. K. Gupta and G. De Micheli, *A Co-Synthesis Approach to Embedded System Design Automation. Design Automation for Embedded Systems*, January 1996.

[43]. Rajesh Kumar Gupta,*Co-Synthesis Of Hardware And Software For Digital Embedded Systems*, Phd. Thesis, Dept. of Electrical Engineering, Stanford University, 1993.

[44]. *HardwareC-A language for Hardware Design,* (Details not available)

[45]. Pai H. Chou, Ross B. Ortega and Gaetano Borriello, *The chinook Hardware/Software Co-Synthesis System,* Proceedings of the eighth international symposium on System synthesis,France, 1995.

[46]. Tarek Ben Ismail and Ahmed Amine Jerraya, *Synthesis Steps and Design Models for Codesign*, Computer, Vol. 28, No. 2,pp. 44-52, February 1995.

[47]. Telelogic, *Specification and Description Language (SDL),* (Details not available)

[48]. Verkest, K. Van Rompaey, Ivo Bolsens and Hugo De Man, *CoWare-A Design Environment for Heterogeneous Hardware/Software Systems*, Design Automations for Embedded Systems, 1(4), 357-386, 1996.

[49]. L. Lavagno, M. Chiodo, P. Giusto, H. Hsieh, S.Yee, A. Jurecska, and A. Sangiovanni-Vincentelli , *A Case Study in Computer-Aided Co-Design of Embedded Controllers*, In Proceedings of the International Workshop on Hardware-Software Co-design, pp. 220-224. 1994.

[50]. Gerard Berry, *The Esterel V5 Language Primer, Version 5.21 release 2.0*, April 6, 1999.

[51]. D. D. Gajski, F. Vahid, S. Narayan, and J. Gong,*SpecSyn: An Environment Supporting the Specify-Explore-Refine Paradigm for Hardware/Software System Design*, IEEE Transactions on VLSI Systems 6, no 1, pp. 84-100 1998.

[52]. D.D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems. Englewood Cliffs*, NJ: Prentice Hall, 1994.

[53]. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing. Science*, 1983.

[54]. D.D. Gajski et al., *Specification Languages*, presentation slides, September 2000.

[55]. Andreas Mitschele-Thiele, *Systems Engineering with SDL*, Wiley, 2001.

[56]. B. Bolognesi and E. Brinksma, *Introduction to the ISO Specification Language LOTOS*, Computer Networks and ISDN Systems 14, pp. 684-707, 1987.

[57]. Stanislaw Budkowski, *Estelle: ISO-Formal Description Technique*, National Institute of Telecommunications, France, 1989.

[58]. G. Berry, *Hardware implementation of pure Esterel*, In Proceedings of the ACM Workshop on Formal Methods in VLSI Design, January 1991.

[59]. David Harel, *Statecharts: A visual formalism for complex systems*, Science of Computer Programming, 8:231-274, 1987.

[60]. Robert Esser, *An object oriented Petri net language for embedded system design*, In: Proceedings of the 8th International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering, London, 1997.

[61]. D. Smit, *VHDL & Verilog Compared & Contrasted*, Proc. 33rd Design Automation Conference, 1996.

[62]. Peter J. Ashenden, *Verliog and Other Standards*, IEEE Design & Test of Computers, pp. 84-85, January 2002.

[63]. Rainer Dömer, *The SpecC Language*, A Tutorial Presentation, Centre for Embedded Computer Systems, University of California, Irvine. (Date not available)

[64]. Andreas Gerstlauer, *The SpecC Methodology*, A Tutorial Presentation, Centre for Embedded Computer Systems, University of California, Irvine. (Date not available)

[65]. Rainer Dömer, *System-Level Modeling and Design with the SpecC Language*, PhD Thesis, University of Dortmund, 2000.

[66]. C.A.R Hoare, *Communicating Sequential Processes*, Prentice Hall International, First Publication 1985, March 2003.

[67]. Daniel C. Hyde, *Introduction to the Programming Language Occam*, Department of Computer Science, Bucknell University, Lewsiburg, Updated March 20, 1995.

[68]. P. Hilfinger, *A High-Level Language and Silicon Compiler for Digital Signal Processing*, In proceedings of the Custom Integrated Circuits Conference, -NA, 1985.

[69]. Stan Y. Liao, *Towards a New Standard for System-Level Design*, CODES'00, 2000.

[70]. Celoxica Limited, *Handel-C Reference Manual*, 2001.

[71]. Cedric Alquier, *Stephane Guerinneau, Lauro Rizzatti and Luc Burgun, Co-Simulation Between SystemC and a New Generation Emulator*, DesignCon 2003.

[72]. Xilinx Website, *www.xilinx.com*

[73]. Sanjiv Narayan, Frank Vahid, Daniel D. Gajski, *System Specification with the SpecCharts Language,Design & Test of computers*, pp 6-13 (Vol. 9, No. 4), October/December 1992.

[74]. A. Birrell and B. Nelson, *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, 1984.