COMET: A Hardware-Software Codesign Methodology*

Michael J. Knieser
Department of Computer Engineering
Case Western Reserve University
Cleveland, OH 44106
knieser@alpha.ces.cwru.edu

Christos A. Papachristou
Department of Computer Engineering
Case Western Reserve University
Cleveland, OH 44106
cap@alpha.ces.cwru.edu

Abstract

COMET is a system-level C and VHDL hardware/software codesign methodology. This process is made possible through the use of a rules file which adds timing and area constraints to the C and VHDL descriptions that the languages do not support. The methodology of COMET is functional and has been tested. A neural network program has been implemented to perform automated hardware/ software partitioning.

1. Introduction

The state of industrial computer-aided engineering(CAE) tools used for hardware and software design is currently at the structural level for hardware and the compiler level for software. Industry has just begun to release behavior synthesizers for computer hardware, but many problems still exist. Much research has been done [10], but more is still needed to solve these problems. There is also a need for the next level of research involving partitioning the system-level description into hardware and software as well. There are many ways of approaching this research and many tradeoffs to consider [9] [2] [11] [5].

1.1. Motivation

Figure 1 illustrates a typical top-down design process. First, a conceptual system definition of the product is created. This definition can be graphical, textual, symbolic or any combination of these methods. Next, the definition is converted into a system representation. Once we have a system representation, the design is ready for hardware/software codesign. The codesign objective is to optimally designate some parts of the system into software components and other parts into hardware components.

Traditionally the engineer has done everything from writing the conceptual system description to partitioning the software and hardware, often using ad-hock methods such as

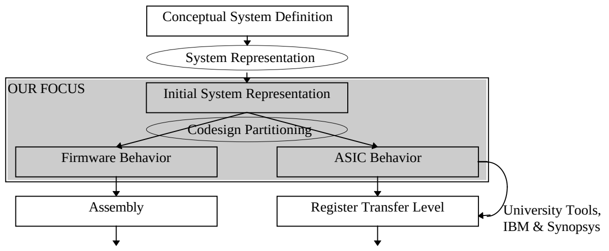

Figure 1. Design abstraction and our focus.

past experience and knowledge of current design practices. We are proposing a hardware/software COdesign METhodology(COMET) that is capable of performing hardware and software partitioning to reduce the overall design time of a system.

1.2. Contributions of our Research

Our approach to hardware/software codesign has two elements: specification and codesign partitioning methodology. The specification methodology uses current programming languages and a rules file to fully describe the system. We assume that all software components are written in standard C and we assume that all hardware components are written in standard VHDL. The rules file consists of statements about the system which neither C nor VHDL can describe and codesign constructs to aid in the process.

The codesign partitioning methodology uses the system defined in C, VHDL and the rules file to partition the system description along function boundaries into optimal hardware and software partitions. Once we have statistics on each partition, COMET will suggest some hardware/software solutions, and report all the partition information to the engineer in a spreadsheet format. To perform automated codesign partitioning, we developed a neural network partitioner that determines hardware/software splits based on desired goals.

The rest of this paper will continue with section 2 discussing related work in the hardware/software codesign field and their differences from the COMET methodology. Section 3 will discuss the basis of the COMET methodology with emphasis on the estimators and the rules file. Section 4 will de-

*This work was partially supported by Rockwell Automation, Allen-Bradley.

scribe COMET's methodology with emphasis on codesign partitioning, interfacing, and the partition evaluator. We will finish with section 5 by demonstrating the results we have generated thus far and conclude with the status of our research.

2. Related Work and Their Differences

There is currently much research being done in the area of hardware/software codesign. This section will review a few of these major research efforts, and their differences from COMET.

2.1. Related Work

The Cosyma hardware/software codesign system [2] starts with the system described in Cx. Cx is a superset of ANSI C where the extensions are timing, task concepts, and task intercommunication. The designer must describe the behavior of the hardware for possible implementation as a C function. The Cosyma tool's internal representation of the system is described by ES (Extended Syntax) graphs. Partitioning and codesign occurs in the ES graph form and the results are converted to C for software and HardwareC for hardware. How the system is partitioned remains an open question for the Cosyma system; however, their direction is towards an iterative partitioning method. As for software estimations, partitions are compiled and the object code is simulated with an RTL (register-transfer-level) simulator. The hardware estimations are performed by synthesizing the hardware partitions and also feeding the results into the RTL simulator.

The hardware/software codesign environment created by [9] uses Verilog for the hardware description and C for the software description. The input into their codesign tool consists of a complete functional description of the system, a technology description and performance goals. Partitioning the input system is done by breaking it up into nontrivial sequences of operations. Each partition considered for hardware is memory-mapped with an interrupt completion line back to the controlling software. Software and hardware estimations are done on a "cycle per input byte" comparison with the parameters of cost and performance as the metrics for codesign. The output of the tool is C software partitions, Verilog hardware partitions and standard parts descriptions.

The Codesign Finite State Machine(CFSM) [1] is a representation of a hardware/software codesign system. The intent is to convert high level behavior languages like VHDL, SpecCharts and others into a CFSM representation. Once the system is in a CFSM, partitioning is done in an iterative fashion based on design metrics [1]. After partitioning, software is mapped into S-graphs which are technology-independent representations of the software components. Then the S-graphs are mapped into portable C code for compilation. The hardware partitions are converted from CFSM's into an abstract description. This description is then logic synthesized to generate the final synchronous hardware description.

SpecCharts [3] is another system specification language for hardware/software codesign. The researchers have developed estimators for both software and hardware. Their software estimator is based on a generic processor model method of estimation [4]. Their hardware estimators can estimate based on many models such as a pin model, an area model, and a performance model [10]. Through the use of a graphical interface and graphical representations of their estimations, hardware/software codesign can be done in an interactive environment.

2.2. COMET Differences

COMET uses the C and VHDL languages directly (along with the rules file) to perform hardware/software codesign; whereas, other tools make use of intermediate languages that are not so universally know and have no commercial synthesis tools to support them. The problem with using these languages is that eventually it has to be converted to C and some hardware description language like VHDL for compilation and synthesis. C and VHDL are well known languages and commercial synthesis tools already exist for them. These factors will increase the quality of the codesign result while preserving the original C and VHDL system description.

Another important difference is that our codesign process does not require a functionally complete description to perform hardware/software codesign. Other codesign systems require functionally complete descriptions to perform the codesign because every partition has to be estimated. With COMET, a VHDL or C stub can be created for those partitions for which functional code cannot be generated, or has yet to be generated.

Also, COMET can perform substitutional operations on partitions where multiple options are available via the rules file which others cannot do. This way we believe that we can truly analyze the hardware/software tradeoffs of particular options for a system.

Next, interfaces between hardware and software are considered important to the codesign process. Thus COMET provides greater control of the interface methods via the rules file. This removes restrictions to the possible communication methods between partitions. Some other systems have not fully addressed the importance of interfacing hardware and software partitions.

The codesign of a system can be an iterative "design-codesign-simulate-improve" loop where the original system is refined until a quality system is achieved. By describing the system in an intermediate language it incurs an iterative "design-codesign-simulate-improve" loop in that language, but then subsequent iterations must be done in the synthesis languages. This is undesirable for two reasons: the design cycle of the product increases and in the end there are two system descriptions to manage. The most important difference is that all of these aspects are not found in any one other tool.

3. The Basis of COMET

The approach one takes to hardware/software codesign is important to its quality. Figure 2 illustrates our approach.

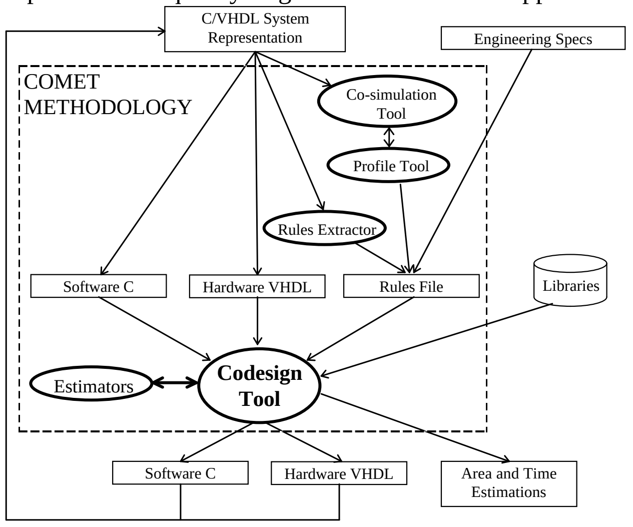

Figure 2. Road-Map of the COMET tool.

Our codesign process starts with C, VHDL and a rules file. The rules file will contain the engineering specifications for the system including the design specifications and any engineering scenarios that are applicable. Hardware timing information can be embedded in the VHDL code or in the rules file. The results of the codesign process are the original C/VHDL source, the time and area estimations in spreadsheet format and any C/VHDL code that is generated.

3.1. COMET Rules File

The rules file is the cohesive element between the software C and the hardware VHDL. This section will discuss some of the important features of the rules file.

One feature is the substitutional constructs that it provides. These constructs allow functions to have substitute time and area values for hardware and software partitions without affecting the simulation. For instance, in an answering machine application, it may not be known whether the recording media is an audio cassette or EEPROM. There could be a function called ”write_message_to_media()” which prints a message that a notice has been written. If we already knew the time and area estimations available for the audio cassette and the EEPROM versions, the estimations would be entered as shown in Figure 3.

To take advantage of reusable pre-existing components, libraries are a necessity. There are rules allowing for the inclusion of such a library. The syntax is similar to C where ”#include ”/home/library/old_useful.rules”” will refer to a physical location for an included library component and ”#include <previous_design.rules>” looks at an environment variable for the included library path for the library component.

```
.write_message_to_media = {EEPROM,audio_cassette};
.write_message_to_media.:hardware_only;
.EEPROM.:hardware.default.area = 100000 transistor;
.EEPROM.:hardware.default.timing = 10 msec;
.audio_cassette.:hardware.default.area = 10000 transistor;
.audio_cassette.:hardware.default.timing = 1 sec;
```

Figure 3. Answering Machine Example.

```
.fft:hardware_to_software_interface = {FIFO_type,shared_mem};
.fft:software_to_hardware_interface = {simple_handshake};
                                (a)
.printf.:software_only;
all_hardware.queue.:hardware_only;
stimulus..:simulation_only;
                                (b)
```

Figure 4. Interface and exclusion constructs.

Interfaces between software and hardware are specified via the rules file (Figure 4a). The engineer can select a few methods for an interface and the COMET tool will choose the best of those methods. There are also rules which force partitions to be software only, hardware only, or simulation only (Figure 4b). The simulation only rule will exclude any partitions from being considered in the codesign process.

3.2. The Estimators

3.2.1. COMET Profiling

The main objective of profiling is to provide information about the branching and iterative constructs of the C and VHDL system to the estimators. Since the engineer will be dealing with C and VHDL source, the profile information should relate back to that source. The ideal profiler for COMET is one that would just record the number of times a basic block of C or VHDL was executed. It would also report its findings in a way that fits into the rules file format. We have examined public domain profilers and determined that what our tool needs is not available. So, we have developed our own C/VHDL profiler. This profiler adds the necessary code to VHDL and C such that when the system is re-simulated the number of iterations of each profiled line of VHDL and C is extracted. By knowing how many times a line of C or VHDL is executed, we can compute the average probability for branches and the number of iterations for loops.

3.2.2. The software estimator

The method of software estimation we are proposing is a hybrid of both the established processor specific estimator and the generic processor estimator. [4] Figure 5 illustrates our method of software estimation.

We propose the use of processor-specific compilers with

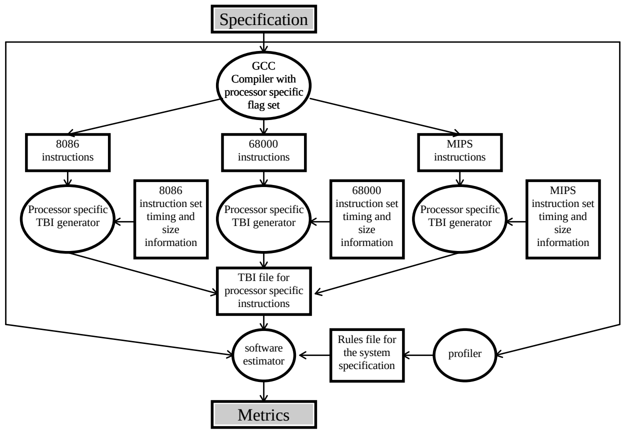
Specification
GCC Compiler with processor specific flag set
8086 instructions
68000 instructions
MIPS instructions
Processor specific TBI generator
8086 instruction set timing and size information
Processor specific TBI generator
68000 instruction set timing and size information
Processor specific TBI generator
MIPS instruction set timing and size information
TBI file for processor specific instructions
software estimator
Rules file for the system specification
profiler
Metrics

Figure 5. COMET software estimation.

a generic estimator. It turns out that the GCC compiler is re-targetable and supports a host of target processors. GCC also provides facilities of attaching debugging information to the assembly that the compiler produces. This debugging information relates the single line of C code to the number of instructions required to accomplish that code. We use this debugging option to create processor-specific assembly with relationships to the C code from which it was generated. Thus, we have created an automated tool to abstract the assembly instructions. This tool will convert the processor-specific instructions into Time, Bytes and Instruction information for each line of C. Thus, we call it the TBI generator. This effectively removes the specificity of the processor, allowing for the use of a generic software estimator.

Instead of using the assembly instructions for estimation, we use the C source code and the output of the TBI generator to do the same thing. The benefit of using C over assembly is that we do not lose any information about the C constructs that are lost when compiled. When profiling is used to enhance the estimators, the information about the looping constructs is more visible in C versis the assembly counterpart. Since the engineer is familiar with C and VHDL looping and branching constructs, there is an understanding benifit to estimation in this fashion.

3.2.3. The hardware estimator

COMET uses an estimator based on a data flow graph (DFG) approach of hardware modeling. This is analogous to high-level or register-transfer-level(RTL) synthesis except COMET stops short of the synthesis process. Our approach to hardware estimation is to convert the VHDL into a DFG representation while considering the profiled looping and branching constructs. Afterwards, the DFG representation is scheduled and functional units are allocated to compute an estimate.

The VHDL-to-DFG program converts VHDL processes, functions, and procedures into a DFG format similar to that used in SYNTEST. [7] Our DFG format differs from SYNTEST's in that ours requires the computation of the complete execution time for comparison to its software counterpart.

Scheduling is done in the same manner as SYNTEST. [7] After scheduling and allocation, the data path is examined to determine the average execution time of the original VHDL partition. We are planning to use the layout estimation techniques found in [6] to produce area estimates for codesign partitions.

4. Codesign Methodology of COMET

With the basics of the rules file and estimations already explained , we can define the method by which COMET will perform hardware/software codesign. The following are the main steps that COMET will follow to achieve the codesign process.

a) Unclustering C/VHDL partitions

b) Interfacing codesign partitions

c) Estimating hardware and software partitions

d) Automatically evaluating hardware/software partitions

The codesign process starts with unclustering the C and VHDL system. Section 4.1 will discuss the method COMET uses to uncluster C and VHDL into codesign partitions. After unclustering, the interfaces for each function are gathered or generated. Interfaces are discussed in Section 4.2. COMET will then estimate each partition in both hardware and software. The generated results are gathered for evaluation by COMET and the engineer. COMET sends the gathered data to the codesign partition evaluator which then suggests a hardware/software codesign partition. The codesign partition evaluator is discussed in Section 4.3. The codesign process can be iterative by going through steps a) through d) again with either new rules information or a more robust system definition.

4.1. COMET Pre-Partitioning

The COMET methodology uses a function-level pre-partitioning model for unclustering codesign partitions. In essence, the pre-partitioner will uncluster the functions within the C and VHDL descriptions into unique C and VHDL partition files. Each resulting pre-partition file contains only C or VHDL code. If no rules information exists, each pre-partitioned function is then passed to the estimators. For VHDL, the hardware estimator will only estimate sequential statements like functions, procedures, and processes. For C, the entire language can be estimated.

Due to function-level pre-partitioning, there is always an initial partition: the initial description of the system. This provides a basis point for measuring the quality of the codesigning system. This also allows for the rules file to easily substitute or remove partitions from the codesign process.

4.2. COMET Partition Interfacing

Dealing with the interfaces between hardware and software partitions is an important issue in hardware/software codesign. There are two basic types of interfaces:

- The interface between a hardware partition that is controlling a software partition.
- The interface between a software partition that is controlling a hardware partition.

The reason we do not consider hardware/hardware and software/software interfaces is that they are inherent within C and VHDL already. For VHDL, communication between signals, variables, and functions are based on their connectivity. If two signals are connected, they can communicate as long as the target signal can accept values while the source signal can supply a value. For C, the software/software interface uses the processor's stack to push variables before calling a function and to pop the variables after the called function completes. Therefore, we need only to consider the hardware/software and software/hardware interfaces.

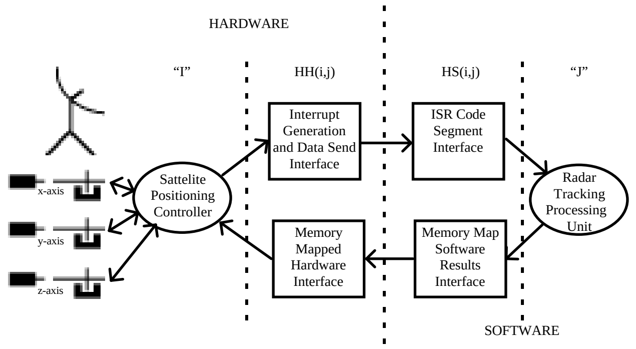
HARDWARE
"I"
HH(i,j)
HS(i,j)
"J"
Interrupt Generation and Data Send Interface
ISR Code Segment Interface
Sattelite Positioning Controller
Radar Tracking Processing Unit
x-axis
y-axis
z-axis
Memory Mapped Hardware Interface
Memory Map Software Results Interface
SOFTWARE

Figure 6. Hardware controlled interfacing.

In Figure 6 we introduce notation for the representation of the interface between two partitions. We start by labeling the current partition for interface consideration as "I." The partition that "I" is interfacing to is partition "J." The interfaces between partitions "I" and "J" consist of software and hardware interfaces. For a hardware partition "I" which is controlling a software partition "J" (Figure 6), the interfaces are HH(i,j) which stands for Hardware controlled Hardware Interface from partition "I" to partition "J"; and HS(i,j) which stands for Hardware controlled Software Interface from partition "I" to partition "J." For a software partition "I" which is controlling a hardware partition "J", the interfaces are SH(i,j) which stands for Software controlled Hardware Interface from partition "I" to partition "J", and SS(i,j) which stands for Software controlled Software Interface from partition "I" to partition "J."

Any type of interface is allowed wheither synchronous or asynchronous. All that COMET requires is either a cost estimates passed within the rules file or a C or VHDL representing the interface desired. At present, if no interface information is passed in the rules file, COMET will supply memory-mapped interfaces between the hardware and software partitions.

4.3. Hardware-Software Partition Evaluator

Once the estimators compute the time and area metrics for the partitions and the interfaces, we can begin to decide which partitions are better implemented in software or hardware. The evaluator tries to maximize the flexibility of a design by placing only the necessary partitions in hardware. The method we are investigating now is a neural network solution. The neural network uses McCulloch-Pitts binary neuron model [8] for the placement of each codesign partition: 0 for hardware, 1 for software.

Due to the functional partitioning of the system we can represent the connections of a system in what we are defining as a call graph. A call graph node represents a codesign partition.(Figure 7) The directed arcs show a function call from one codesign partition to another. The number of times a function calls a partition is the weighted value for that directed arc. Each node has an associated average execution time excluding the time from its called function.

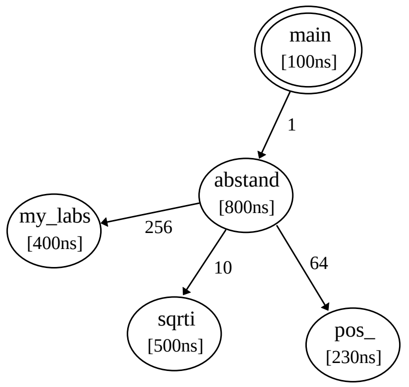

Figure 7. Decorated COMET call graph.

To compute the average execution of a given call graph we can use the following equations.

$$AE_{total} = \sum_{i=0}^{P} TNE_i \tag{1}$$

$$\begin{aligned} TNE_i \quad = \quad & \sum_{j=0;j!=i}^{P} (Iter_{ji} \cdot (NE_i + \\ & X_j^{'} \cdot X_i \cdot (THH_{ji} + THS_{ji} \cdot clk) + \\ & X_j \cdot X_i^{'} \cdot (TSH_{ji} + TSS_{ji} \cdot clk))) \end{aligned} \tag{2}$$

$$NE_i = X_i \cdot (clk \cdot S_i) + X_i^{'} \cdot H_i \quad (3)$$

At present, the partition evaluator will try to meet the goal of a specific execution time and the number of memory bytes required by the system. Our reasoning is based on the micron size of transistors. As the transistor size decreases to below .4 microns, the custom ASIC becomes pad limiting leaving ample space for custom hardware, an embedded processor and memory. Now the issue comes down to how much memory can be placed in the ASIC versus the custom hardware. Therefore, knowing the size of the embedded program become a major issue. To compute the average bytes for a functional node, we can extract the number of bytes for a given partition from the execution time equations. This produces the following equations.

$$AB_{total} = \sum_{i=0}^{P} TNB_i \quad (4)$$

$$TNB_i = \sum_{j=0;j!=i}^{P} (Callee_{ji} \cdot (X_j^{'} \cdot X_i \cdot BHS_{ji} + X_j \cdot X_i^{'} \cdot BSS_{ji})) \quad (5)$$

$$NB_i = X_i \cdot B_i \quad (6)$$

With these equations (1-6), we can implement a neural network hardware/software partitioner to meet the timing and memory byte goals. The neural network operates in an iterative fashion. Each iteration computes a new energy difference for that neuron and adds that to the current binary neuron input. Based on the updated input the output is set accordingly. Consecutive iterations occur in the same fashion until the system stabilizes a solution.

5. Results and Status

5.1. Chroma-key Algorithm

The Chroma-key algorithm is used by [2] to benchmark their Cosyma codesign system. The algorithm is used in high-definition television studio equipment. The first of the four implementation columns is the initial partition implementation that is completely implemented in software. This implementation resulted in a 93.51505 second execution with 51753 bytes. The second column of implementations has a goal of 1 second for execution and a goal of eight kilobytes of memory. The result was 11.214297 seconds and 10083 bytes. The third column of implementations has a goal of 20 seconds for execution and a goal of sixteen kilobytes of memory. The result was 27.155764 seconds and 15994 bytes. The forth column of implementations has a goal of 70 seconds for execution and a goal of thirty-two kilobytes of memory. The result was 78.253448 seconds and 31968 bytes. The results were generated assuming a 20Mhz 68HC11 processor. The interfaces were assumed memory mapped for software-controlled hardware and ISR memory mapped for hardware-controlled software.

5.2. Status

The main goal of the chroma-key experiment was to show that the methodology of COMET is operational. This goal was achieved with approximately 95% automation. All the components of COMET, including the neural network partition evaluator, have been implemented in a preliminary form. Given a profiled system, COMET can pre-partition the system, generate hardware/software interfaces, estimate the hardware and software costs, gather the estimated results in an integrated form and suggest a hardware/software partition. The COMET tools are implemented in both a command-line form and a common graphical user interface.

Also, work is underway to verify the quality of the estimators as well as adding area estimation as a codesign parameter. We also expect complete automation of the COMET codesign methodology soon. Also, in the workings, is an industrial system design to verify the functionality and usefulness of the COMET methodology in a market-driven industrial atmosphere.

References

[1] M. Chiodo, P. Giusto, A. Jurecska, H. C. Hsieh, A. S. Vincentelli, and L. Lavagno. Hardware-software codesign of embedded systems. *IEEE Micro*, pages 26–36, August 1994.

[2] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test of Computers*, pages 64–75, December 1993.

[3] D. D. Gajski and F. Vahid. Specification and design of embedded hardware-software systems. *IEEE Design and Test of Computers*, pages 53–67, Spring 1995.

[4] J. Gong, D. Gajski, and S. Narayan. Software estimation using a generic-processor model. *European Design & Test Conference*, pages 498–502, March 1995.

[5] T. B. Ismail and A. A. Jerraya. Synthesis steps and design models for codesign. *IEEE Computer*, pages 44–52, February 1995.

[6] M. Nourani and C. Papachristou. A layout estimation algorithm for rtl datapaths. *Design Automation Conf. (DAC 93)*, pages 285–291, June 1993.

[7] C. Papachristou, H. Harmanani, S. Chiu, and M. Nourani. Syntest: An environment for system-level design for test. *First European Design Automation Conf. (EURO-DAC 92)*, pages 402–407, September 1992.

[8] Y. Takefuji. *NEURAL NETWORK PARALLEL COMPUTING*. Kluwer Academic Publishers, 1993.

[9] D. E. Thomas, J. K. Adams, and H. Schmit. A model and methodology for hardware-software codesign. *IEEE Design and Test of Computers*, pages 6–15, September 1993.

[10] F. Vahid and D. D. Gajski. Specification partitioning for system design. *29th ACM/IEEE Design Automation Conference*, pages 219–224, June 1992.

[11] C. A. Valderrama, A. Changuel, P. V. Raghavan, M. Abid, T. B. Ismail, and A. A. Jerraya. A unified model for co-simulation and co-synthesis of mixed hardware/software systems. *European Design & Test Conference*, pages 180–184, March 1995.