Framework For Hardware/Software Partitioning Utilizing Bayesian Belief Networks

John T. Olson and Jerzy W. Rozenblit
Dept. of Electrical and Computer Engr.
University of Arizona
Tucson, AZ 85721
{olson | jr}@ece.arizona.edu

ABSTRACT

In heterogeneous systems design, partitioning of the functional specifications into hardware and software components is an important procedure. Often, a hardware platform is chosen and the software is mapped onto the existing partial solution, or the actual partitioning is performed in an ad hoc manner. The partitioning approach presented here is novel in that it uses Bayesian Belief Networks (BBNs) to categorize functional components into hardware and software classifications. First, the motivation and background material are presented. Then, a case study of a programmable thermostat is developed to illustrate the BBN approach. The outcomes of the partitioning process are discussed and placed in a larger design context, called model-based codesign.

1.0 INTRODUCTION

In this article, we propose a new approach to the hardware/software partitioning problem [3][4][9] by utilizing Bayesian Belief Networks for functional component classification into hardware and software. Design of heterogeneous systems entails choosing which functional components should be implemented in hardware and which should be implemented in software. Classically, a hardware platform is chosen and the software is written to make the hardware meet the specified requirements. The problem with this approach, however, is that during system integration, interface and incompatibility problems arise. Hardware/software partitioning is used to push the implementation decisions back so that the decision of whether to use hardware or software is not made in isolation for each functional component.

In our previous work, we have established a systematic approach to design of heterogeneous systems. Called model-based codesign [7][8], this approach uses simulatable system descriptions as the basis for the generation of design descriptions from which the real system is built. Simulation is used as a primary means of verifying functional requirements of the design. Thus, in parallel to the simulation, classifications to the system model components into hardware or software must be made.

The partitioning approach presented here uses the Bayesian Belief Network (BBN) concept [2][5][6][12] for classification of functional elements within the system model. The reasons for using the BBN framework are: (1) its aptitude to represent the causal nature of a functional description (e.g., a function, *A*, calling another function, *B*, is a causal influence from *A* to *B*) and (2) the ability to distribute local evidence throughout the entire network and thus, make the effects of a local partitioning decision affect partitioning decisions throughout the entire model. This in conjunction with the other benefit of having probabilistic measurements as to the correctness of classification decisions makes the use of BBNs appropriate.

In the ensuing sections, we first describe the principles of BBNs and briefly summarize the hardware/software partitioning work. Then, we present the BBN-based partitioning methodology. An illustrative partitioning problem is introduced in Section 3 to demonstrate our technique.

1.1 Background

A Bayesian Belief Network (BBN) is a directed acyclic graph, representing the causal nature of a problem domain [5]. A BBN is composed of two parts: (1) the graphical representation showing the causal relationship between nodes (the qualitative part), and (2) the conditional matrices associated with each link and the equations that govern the propagation of evidence (the quantitative part).

In the qualitative portion of a BBN, a directed arc from any node *A* to another node *B* (denoted $A \rightarrow B$) denotes the causal influence of *A* over *B*. The use of the qualitative portion of a BBN lies in the graphical nature in which it is represented. Someone with little experience in the area of probablistic reasoning can

0-7803-4778-1/98 $10.00 © 1998 IEEE

easily understand the causal relationships among thenodes. Each node within a BBN represents a statistical random variable, which may comprise of several hypotheses.

When distributing probablistic evidence throughout a BBN, two types of messages are used: (1) evidence messages carry the effects of newly introduced evidence, and (2) causal messages carry the effects of causal influences. The quantitative portion of a BBN uses the qualitative part by determining in which direction the evidence and causal messages travel throughout the network. Evidence messages travel against the direction of the arc in the form of λ messages. The causal messages travel with the direction of the arc in the form of π messages. The combination of these two types of messages, along with the prior probabilities and link matrices are used to determine the beliefs associated with each node of the graph. The prior probabilities give the hypothetical beliefs for each node before any evidences have been introduced (usually set to equal probability), and the link matrices represent conditional probabilities of choosing a hypothesis given that the values of the hypotheses of a node acting as a causal influence are already known. For a more detailed description of BBNs we refer the reader to [2][5][6][12].

1.2 Previous Work

There is a great body of partitioning work that is well documented in the literature. [1] provides an excellent review of the major classes of partitioning algorithms that not only can be used for VLSI circuit design, but for any system in which components are grouped and whose inter-group communication must be kept to a minimum. Among these classes of partitioning algorithms are the following: (1) move-based approaches such as greedy and iteritive exchange algorithms, (2) geometric approaches such as vector partitioning, (3) combinatorial approaches such as max-flow min-cut, and (4) clustering-based approaches [1]. In addition, other researchers have either augmented the general algorithms (for example, Vahid [11] modified the min-cut algorithm for functional partitioning), or introduced new types of algorithms (such as Wolf who employed an object-oriented approach [13]).

2.0 THE PARTITIONING PROBLEM

In the design of heterogeneous systems, the choice of how to implement the system architecture can make significant differences in performance and reliability. In the past, a hardware platform was often chosen and then software was written for correcting the inadequacies of the hardware. Currently, however, research has progressed from the idea of partitioning hardware elements, to that of partitioning a high level functional model of a system. Figure 1 shows an example partitioning into hardware and software, with the system model containing four functional components (A, B, C, D) that are partitioned into hardware (A, C, D) and software (B).

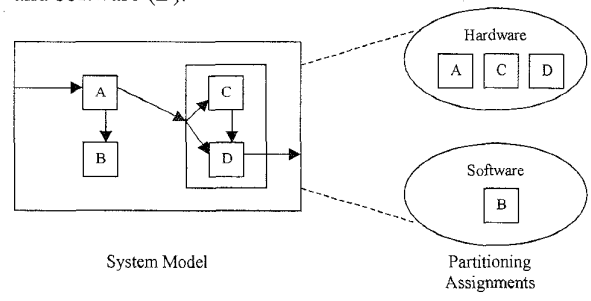

Figure 1: An illustration of the partitioning problem.

The hardware/software partitioning methodology we present here is part of a larger design context called model-based codesign [7][8]. In model-based codesign, a set of requirements and specifications are obtained for the system to be modeled. The system is then described as an abstract model that is a combination of its structural and behavioral specifications. Model components are specified at a high level of abstraction to remain technology independent. The modeling process includes a stepwise refinement of specifications to a desired level of granularity. Then simulation studies are carried out to gain introspection into how well the model-based specifications meet the system's requirements. At the end of the simulation process, a virtual system's prototype is obtained.

The BBN framework uses the results from the simulation experiment as evidence. The hardware and software functional classifications chosen by the BBN framework are mapped into specific hardware and software components. At this point, the abstract model is considered to be mapped onto a collection of interconnected, real world components.

2.1 BBN Functional Classifications

There are several requirements of a functional classification system as described above. Evidence produced from simulation results is propagated throughout the BBN. Also, coupling between functional components determines the values in the conditional matrices. These two facts combined with the hierarchical nature of the system model requires that simulation experimental specification and interfaces between levels of abstraction be well defined. This ensures that correct evidence is introduced and that the

values in the link matrices allow accurate evidence propagation. The introduction of evidence along with the causal structure of the belief network can be combined to calculate the beliefs of component classifications (e.g., the analysis of a simulation result may introduce evidence in support of implementing a given component in hardware or software). The classification algorithm and its description follow.

CLASSIFICATION ALGORITHM

```
functional_model := generate_functional_model(requirements);
BBN := generate_BBN(functional_model);
while (not(synthesizable) ) do
        results := simulate(system_model);
        evidence := convert_to_evidence(results);
        BBN := propagate_evidence(evidence);
        system_model := add_classified_components(BBN);
        synthesizable := check_synthesis(system_model);
end while
```

In the classification algorithm, given an initial system model, a functional description of the model is created (in the format similar to the Specification Level Intermediate Format (SLIF) model [10]). Next, the BBN is generated with nodes representing functional components, and causal links corresponding to component couplings, function accesses, and functional independence of components. The choice of which values to place inside the conditional matrices associated with each link depend on the communication needs between the given pair of elements, and how tightly their performance is coupled. Once the BBN is created, it can be used to evaluate the current design by incorporating the simulation results as evidences.

During each iteration of the design loop, results are obtained from simulation and converted into evidence that is propagated throughout the BBN. The beliefs for each available type of classification are calculated at each component node and the system model (now possibly with some classified components) is altered to reflect the new classifications. Simulation is performed again, and the process is repeated until the components of the system model reach a level that can be synthesized into a prototype capable of being built and tested with tangible hardware and software. In determining if a model can be synthesized into actual hardware and software components, we look at the strengths of the beliefs associated with each functional classification, and check to see if they meet a required threshold value.

3.0 AN ILLUSTRATIVE EXAMPLE

In this section, we present a programmable thermostat design example. The thermostat must meet the following requirements:

1) The user must be able to set the temperature in the range 55 degrees F to 100 degrees F.
2) If cooling, the unit must turn on at desired_temp + 1, and cool to desired_temp –1
3) If heating, the unit must turn on at desired_temp – 1, and heat to desired_temp + 1
4) The unit must be able to keep track of day of the week and time of day.
5) The unit must be able to store 5 temperature zones (a zone consists of a starting and ending time, along with a desired temperature) for every day of the week, which will be active while in the program mode:
   a) The temperature will return to the default desired temperature if no temperature zone is specified for a particular time, and the unit is in program mode.
   b) The default desired temperature would also be stored where the temperature zone programs are stored.
6) The unit must be able to be switched from program mode to manual mode and back again without malfunctioning.
7) There must exist a method to keep the stored program data in the event of a power failure.
8) The temperature must be checked at least 10 times every second.

In meeting these requirements, we adopted a functional representation similar to SLIF and divided the problem into variables and functions needed. The table below gives the variables and a description of whether or not they are fixed size and an approximate size to each.

Variables:	**Fixed-size?**	**Size (order of magnitude)**
Current-Temp	Yes	1 byte
Desired-Temp	Yes	1 byte
Program-Array	Yes	100 bytes

Table 1: Variables needed for programmable thermostat.

The next table gives the names of the needed functions along with the type of algorithm they employ (standard, loop, or switch) and a rating as to how critical the speed in which the function is performed (on a scale from 1 to 10).

Functions:	Type	Speed-Critical-Rating
Get-Current-Temp	Stand.	5
Control-Temp	Loop	1
Get-Input	Stand.	1
Write-Output	Stand.	1
Check-Program	Switch	3
Determine-Desired-Temp	Switch	2
Accumulate-Elapsed-Time	Loop	10
Determine-Current-Time	Loop	8

Table 2: Set of functions required to implement the programmable thermostat.

From Table 2, we select to model only those functions deemed "important" for the function of the programmable thermostat (excluding input and output) and generate the BBN shown in Figure 2. Depending on how a BBN is constructed, and the interpretation of requirements, the final BBN structure can appear in many different topologies. Therefore, the BBN shown in Figure 2 is only one possible interpretation. In this case, it is easy to see that the causal links point in the direction of one function calling or accessing another.

The conditional matrices were created using the information from Table 2, along with communication assumptions. To interpret the entries to the conditional matrices, take the entry for row 1, column 1 of the matrix associated with the link from Control-Temp to Determine-Desired-Temp (shown in bold). This particular entry states that the probability that Determine-Desired-Temp should be implemented in hardware given that Control-Temp is already implemented in hardware is 75%. Row 2, column 1 indicates the probability that Determine-Desired-Temp should be implemented in software given that Control-Temp is implemented in hardware is 25% (shown in bold). Similarly for the link from Determine-Current-Time to Accumulate-Elapsed-Time, it can be seen from the matrix that if Determine-Current-Time is implemented in hardware, then the probability that Accumulate-Elapsed-Time should also be implemented in hardware is 90% (shown in bold). In general for any link matrix from $A \rightarrow B$, column 1 represents A is in hardware, column 2 represents A is in software, row 1 represents B is in hardware, and row 2 represents B is in software.

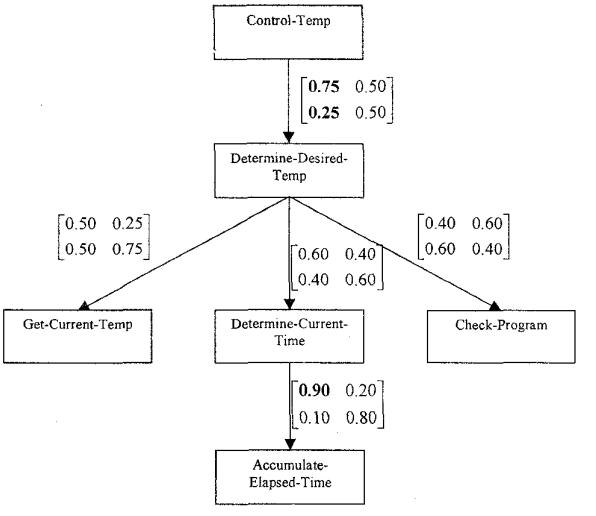

Figure 2: Bayesian belief network for the major functional components.

With the BBN given, it is now possible to begin the 'while' loop of the algorithm. At this point, the system model would be simulated and the results from simulation converted into evidences that can be propagated throughout the BBN. Because our methodology is currently a stand-alone system, the evidences introduced here are estimates of reasonable values that one would expect to obtain from simulation. Figure 3 shows one such piece of evidence added to the node for Determine-Current-Time. Determine-Current-Time is deemed as a computationally intensive loop that has to execute often. Because of this, a reasonable estimate for the evidence that this function should be implemented in hardware has an 85% probability and a probability of 15% to be implemented in software. Figure 3 shows the beliefs associated with each function, after the introduced evidence has propagated throughout the BBN. Note that because of the causal relationship (and strong coupling shown by the conditional matrix) between Determine-Current-Time and Accumulate-Elapsed-Time, the belief that Accumulate-Elapsed-Time should also be implemented in hardware is now approximately 80%.

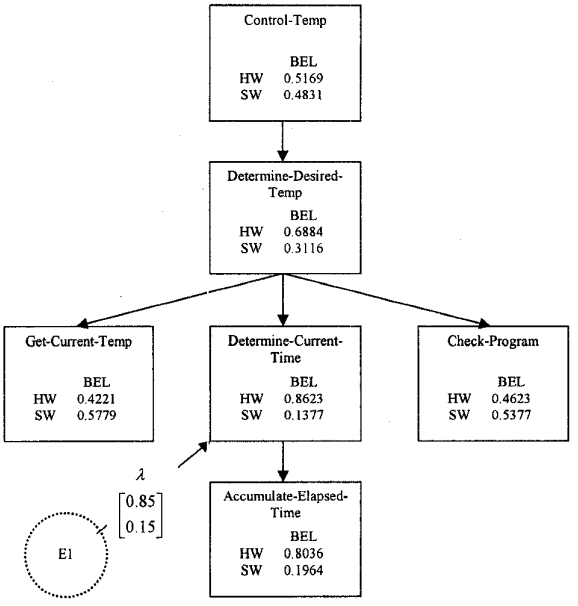

Figure 3: Introduction of the first evidence node.

Figure 4 shows another piece of evidence being added to the BBN; in this case for Control-Temp. Because Control-Temp calls other functions and does not need to be performed at critical speeds, a reasonable performance estimate would have a 25% probability of being implemented in hardware, and a 75% probability of being implemented in software. This estimate assumes that the cost of software is significantly cheaper than hardware and should be used when performance permits; an interpretation that would normally be used in the conversion of simulation results to evidence. The beliefs shown for each function in Figure 4 are those valid after the evidence has been propagated throughout the BBN. It can easily be seen that this piece of evidence has a profound influence over the Control-Temp function by changing its probability of being implemented in software from 48.31% to 73.71%. Also note that the effects of this piece of evidence are felt all the way down to the Accumulate-Elapsed-Time function by lowering the probability it should be implemented in hardware from 80.36% to 79.94%. A small change, but a change none the less.

The process of propagating evidences throughout the BBN continues until no new data can be introduced. At this point, a decision is made as to whether or not each function should be implemented in hardware or software. This decision is based upon the amount of belief associated with each type of implementation (i.e., if the belief is greater than some threshold, e.g., 75%, then that type of implementation will be chosen). The classification of a function into hardware or software is reflected in the system model, and the process continues until all functions can be classified into either hardware or software.

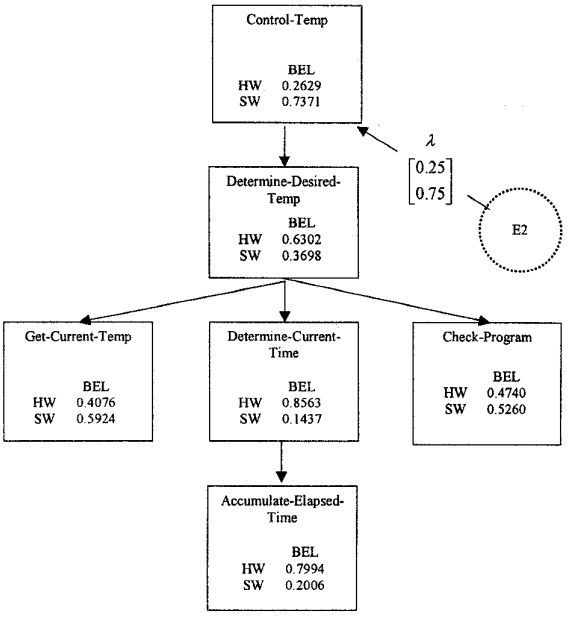

Figure 4: Introduction of second evidence node.

4.0 CONCLUSIONS

In this paper, we have introduced a new methodology for functional partitioning into hardware and software classifications. Through an example system, we have shown how Bayesian Belief Networks can be used to propagate evidence regarding classification of functions into hardware or software realizations. This propagation permits the effects of a classification decision made about one function to be felt throughout the entire network. In addition, because BBNs have a belief of hypotheses as their core, we know how well a given classification fits into either hardware or software. Knowing that a function with a 75% hardware belief, should be implemented 75% of the time in hardware allows the user to have a measure of the appropriateness of their solution.

The work presented in this paper makes several assumptions. The first assumption is that functions would be only classified into hardware or software realizations. Future work will expand the scope of classifications to include several types of hardware and mixed types of hardware and software such as application specific integrated circuits, field programmable gate arrays, among other intermediate

types. This expansion will be implemented by utilizing multi-valued hypotheses, where several classifications of hardware and software are possible for each functional component.

The second major assumption is that we have in place a method to generate both the causal structure of the BBN, and the conditional matrices. Currently, this construction is performed manually. Future research will include automatic generation of the causal structure of the BBN from the input requirements and independence assumptions that can be made about the relationship between sets of functional components. In addition, we plan to automatically generate the conditional matrices associated with each causal link based on the coupling and communications between functional components.

Acknowledgments

This work has been supported by the National Science Foundation under grant No. 9554561 "Hardware/ Software Codesign for High Performance Systems."

References

[1] Charles J. Alpert and Andrew B. Kahng, "Recent Directions In Netlist Partitioning: a Survey," Integration, the VLSI Journal, Vol. 19, No. 1-2, August 1995, pp. 1-81.

[2] E. Charniak., "Bayesian networks without tears," AI Magazine, Vol. 12, No. 4, (winter 1991) pp. 50-63.

[3] M. Chiodo, P. Giusto, A. Jurecska, H.C. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno, "Hardware-Software Codesign of Embedded Systems", *IEEE Micro*, 1994, Vol. 14, No. 4, pp. 26-36,

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

[5] Q. Ji and M.M. Marefat, "Bayesian approach for extracting and identifying features," *Computer Aided Design*, Vol. 27, No. 6, 1995, pp. 435-54.

[6] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[7] J. W. Rozenblit and K. Buchenrieder (Eds.), *Codesign: Computer-Aided Software/Hardware Engineering*, IEEE Press, 1994.

[8] S. Schulz, J. W. Rozenblit, M. Mrva, and K. Buchenrieder, "Model-Based Codesign: the Framework and its Application, *IEEE Computer,* August 1998.

[9] Donald E. Thomas, Jay K. Adams, and Herman Schmit, "A Model and Methodology for Hardware-Software Codesign," *IEEE Design and Test of Computers*, Vol. 10, No. 3, Sept., 1993, pp. 6-15.

[10] Frank Vahid and Daniel D. Gajski, "SLIF: A Specification-Level Intermediate Format for System Design," Proceedings. The European Design and Test Conference. ED&TC 1995, pp. 185-189.

[11] Frank Vahid, "Modifying Min-Cut for Hardware and Software Functional Partitioning," Proceedings of the Fifth International Workshop on Hardware/Software Codesign. CODES/CASHE '97, pp. 43-48.

[12] L. C. Van der Gaag, "Bayesian Belief Networks: Odds and Ends," *Computer Journal*, Vol. 39, No. 2, (1996) pp. 97-113.

[13] Wayne Wolf, "Object-Oriented Cosynthesis of Distributed Embedded Systems, ACM Transactions on Design Automation of Electronic Systems, Vol. 1, No. 3, July 1996, pp. 301-31.