Optimizing the Communication Overheads during the Allocation of Global Memories and Busses

J. Müller, R. Kumar
Automation of Circuit Design, Forschungszentrum Informatik (FZI)
Haid-und-Neu-Str. 10-14, D 76131 Karlsruhe, Germany
Tel: ++49/721/9654412, Fax: ++49/721/9654459
email: {jmueller|kumar}@fzi.de

Abstract
One of the important aspects in synthesizing from multi-process specifications is to deal with the existence of complex data between the processes. In this paper we focus on the allocation of memory elements and busses for realizing such data exchanges. The synthesis approach minimizes the communication delays, taking the area demand for the needed communication structure into consideration. The global data are clustered according to a distance factor which is derived by an analysis of the specification. This clustering information is then used to appropriately partition the global data into subsets which can be assigned to a cost minimal memory and bus structure.

Keywords
system-level synthesis, communication synthesis

1 INTRODUCTION

Most existing commercial and academic VHDL based synthesis tools do not take a system view point of the overall behaviour and hence concentrate on the synthesis of single process specifications. Although some tools synthesize multi-process VHDL descriptions, they do not adequately handle the synchronization aspects between the various processes, thus leading to implementations with hazards, in the presence of multiple clocks. A second major problem in synthesizing multi-process descriptions is the consideration of complex data which can be more efficiently implemented using register files or RAMs, as compared to single registers or latches. Our approach towards multi-process synthesis considers these descriptions from a system point of view, i.e. the global synchronization of the processes and the global storage of the data are accounted for. However in this paper, we mainly concentrate on the memory/bus mapping aspects of the multi-process synthesis.

Two approaches which also consider the above-mentioned problems separately are worth mentioning. In (Berthet a. o., 1992) the possibility of synthesizing VHDL arrays are discussed. The} map each array onto one generic RAM primitive, which is implemented using a RAM generator, however without further exploitation of the design space. In (Narayan and Gajski, 1994) the bus assignment of communicating processes is examined. The number of communication channels (busses) and the data transfers which are to be mapped onto them is assumed. Based on this, they then determine the bitwidths of the busses and the possible protocols. Our approach takes a more global view of the problem by simultaneously considering the trade-offs between the sizes, number and type of memories and also the number and bitwidths of the necessary busses. Furthermore, the communication delays for memory/bus accesses are also incorporated.

This paper is organized as follows: after a brief overview of the overall synthesis system, the parameters influencing the optimization of the memory/bus allocation are discussed. This is followed by the partitioning concepts and finally some experimental results are presented.

2 OVERVIEW OF THE SYNTHESIS SYSTEM

A behavioural, multi-process VHDL specification is transformed into a structural VHDL description of a shared-memory system architecture (figure 1) and a set of behavioural descriptions. The structural description defines the interconnection between the memories, processors, and the global controller. In order to ensure correct communication, specialized interface components are automatically inserted for the memories, processors and busses. These components are available as a predesigned library of generic VHDL models and are instantiated appropriately. In addition, each process in the input specification is transformed into a behavioural processor description with the suitable bus ports. Furthermore, a behavioural description of the global controller is also generated. The synthesis of the behavioural descriptions can then be performed by using commercial design tools like the Synopsys Design Analyser.

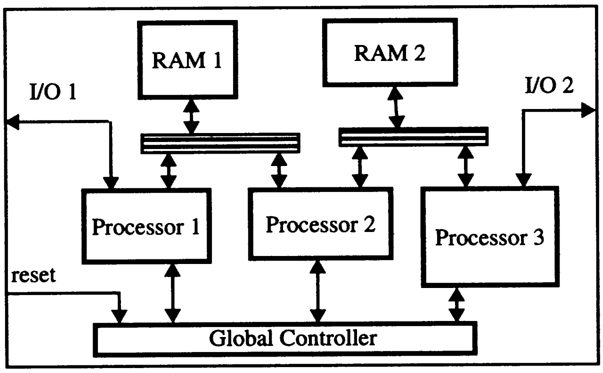

Figure 1 Example for a generated system target architecture

In our approach we distinguish between synchronization and data exchanges. The synchronization of the processors is performed by the global controller, and the data which resides in the global memories is exchanged between the processors via the busses.

In order to define our VHDL subset for the input specification we follow the recommendations given by Camposano a. o. (1991). The communication between different processes and the environment can be specified only by means of global signals. The synchronization signals that are used for the synchronization of the processes are those signals which are used at least in one *wait statement* and additionally, they must be of type BIT. The other signals in the input specification are used for the data exchange. These data signals are assigned to several RAM modules or global registers.
In this paper, we present an approach to determine a nearly optimal communication structure for the data exchange which takes the area and the temporal communication overheads into account.

3 FACTORS INFLUENCING THE GENERATION OF THE COMMUNICATION STRUCTURE

The overall problem that we are solving can be stated as follows:

given a multi-process VHDL description comprising the global signals for synchronization and data transfers, find a communication structure (i.e. the memories and the busses) such that the costs and access conflicts are minimized.

There are two aspects that are to be considered during the generation of the communication structure, namely, the costs and the occurrence of the conflicts. The costs that influence the optimization of the memory and bus mapping of the data signals are the overall area and the access times. The occurrence of the conflicts is analyzed by using a Petri-net model of the communication among the VHDL processes. A brief description of the costs and the conflict analysis is given below.

3.1 Area costs

The area costs can be divided into three parts: the area needed for the individual processors and the global controller, the storage of the global signals and the interconnection between the components of the system. In the context of our task, we need to consider only the area costs for the storage elements and the interconnections and hence the costs to implement the processors and controller are ignored. The total area costs for the storage and the interconnections can be lumped together and considered as communication costs. The area cost for a RAM module depends on the number of stored signals and the bitwidth of the module. The interconnection costs depend on the number and the size of the interface components of the memory modules, which are needed to perform read, write or read/write accesses together with the bus arbiter which is used to resolve concurrent bus accesses among the processors.

3.2 Time costs

The time cost function has to give a relative measure for the communication overhead caused by a certain system structure. The factors influencing the cost are:

- the access protocol determined by the type of memory element which stores a signal
- the number of accesses to a signal
- if a set of signals are assigned to one memory element, then a sequence of accesses is needed
- if the bitwidth of the signal is greater than the bitwidth of the memory then more than one access is required

It is to be noted that, all data accesses to data which are implemented in different memory elements can be performed in parallel. Therefore, the total cost function is determined by the maximum access delay of all RAM modules and registers.

3.3 Conflict Analysis

If more than one signal is assigned to one memory element then the probability of concurrent accesses to the set of signals is needed to evaluate the effect of serialization delays. To rate the probabilities of access conflicts the synchronization scheme of the processors must be analyzed.

In (Müller and Krämer. 1993) a Petri net based approach to examine the system behaviour was described. A Petri net model is used to detect which parts of the different processes can work concurrently. The markings of the reachability graph are regarded as the set of all possible states of the system. Data accesses take place during the transitions between the successive system states. By a Markov process the probabilities of the system states can be calculated which imply the probabilities of concurrent accesses to each pair of data signals. We have improved this approach by some features. The Petri net model was modified into a timed Petri net with fixed firing delays. Here the desired clock frequencies of the processors are taken into account.

4 SYNTHESIS OF THE COMMUNICATION STRUCTURE

The task now is to map disjoint sets of data signals onto memories which implies the overall system structure, while minimizing the costs and access conflicts. The complexity of this problem is large and hence we developed an approach which consists of two phases: first a set of cluster trees which mirrors the similarity of the data signals is created, using an abstract measure which is derived from the communicating processes in the input description. In the second phase, these cluster trees are partitioned. Each partition then corresponds to a set of disjoint subsets of the data signals, where each subset has been assigned to a specific type of memory element. These memory elements implicitly determine the kind of interface and bus components thus leading to a candidate system structure. Therefore, concrete cost functions taken from a library, can be used in this phase.

The cluster trees are created using a concept called *distances* between the signals. The distance between two signals is large if the probability of access conflicts between these signals is high, and it is small if the signals are accessed by many common processes. The distance between two signal *s* and *q* can be calculated by:

$$d_{s,q} = \omega \cdot p_{s,q} + \frac{a_s \cdot a_q}{a_{com}^2 + 1} \tag{1}$$

$p_{s,q}$ is probability of access conflicts between the signals *s* and *q*,
a_s and a_q are the number of processes accessing the signals *s* and *q*,
a_{com} is the number of processes accessing both signals,
ω is a weighting factor to normalize both terms.

The algorithm to build a cluster tree, based on the distances (eqn. 1), was described in (Krämer and Müller, 1992). The leaf nodes of such a cluster tree represent the individual data signals (figure 2). The cluster tree is constructed in such a way that signals with small distances are collected in common subtrees. The shape of a cluster tree depends on the weighting factor ω. In order to explore a large design space, all significant cluster trees resulting between a maximum and minimum ω (ω-range) are created, by using a divide-and-conquer algorithm. The significant cluster trees are created by recursively halving the ω-range, as long as the resulting cluster trees are different or a minimum ω-range is reached.

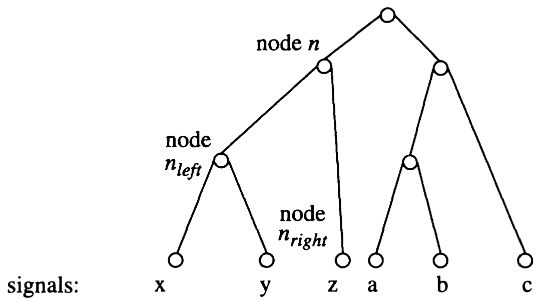

Figure 2 Example of a cluster tree

The cluster tree is recursively traversed from the leaf to the root and appropriate cuts are made in the tree, for creating the different partitions. The following objective function is used to compare two partitions:

$$Obj(\gamma, T, A) = \gamma \cdot \ln T + \ln A \tag{2}$$

T represents the costs for the communication delay and A represents the costs for the area demand. The parameter γ is a user defined factor which biasses a faster or a smaller design. By setting γ as 2 we get the well known AT^2 factor for optimization.

The cluster tree is cut at node n and declared as leaf node if the cost of implementing its subset S_n of signals in one memory module is less than the cost for the separate implementations of the signal subsets on the left and the right successor subtrees, i.e.:

$$Obj(\gamma, T_n, A_n) < Obj(\gamma, max(T_{n_{left}}, T_{n_{right}}), A_{n_{left}} + A_{n_{right}}) \tag{3}$$

It is to be noted in eqn. 3, that the parameters T and A consider the type of memory, i.e. register or RAM and the corresponding bitwidths into account. The formulas for T and A are complex and hence beyond the scope of this paper.

The cluster tree is cut as high as possible so that the overall objective function is minimized. The final cuts result in the memory and bus mapping of the signals.

5 Examples and Results

To illustrate the partitioning approach we have used a 5 process example which computes the determinant of a 3x3 matrix. One process reads the data of the input matrix row by row, three processes calculate the 3 sub-determinants, and the last process sums the sub-determinants and writes the result to the output. The synchronization of the processes was specified so that the input process can restart as soon as the calculation of the sub-determinants has finished, which results in an asynchronous two stage pipeline. The parameters for the bus cost functions were got by instantiating the generic library of interface components built using a commercial standard component library. For the RAM costs we have assumed a RAM generator.

By modifying the weighting parameter ω between 0 and 50 we have got 12 different cluster trees. To find the best partitions, for a given γ all the cluster trees were examined. In our experiments, γ was varied between 0.01 to 100 in order to obtain different realizations ranging from a single RAM solution to separate registers for each signal. Table 1, shows the best 4 results obtained with $\gamma = 2$.

Table 1 Best Partitions for the example with $\gamma = 2$

RAM 15 bit	*RAM 30 bit*	*Registers*	*T*	*A (gates)*	*Obj(γ,T,A)*
1 (9 words)	0	3 (30 bit)	0.29	10281.0	6.731
1 (9 words)	1 (4 words)	0	0.33	9205.3	6.881
0	1 (12 words)	0	0.33	9534.5	6.916
1 (15 words)	0	0	0.43	6268.5	7.035

The best solution uses one 15 bit RAM with 9 words and three 30 bit registers. In the second solution the registers are replaced by a 30 bit RAM with 4 words which is the minimum word number, dictated by the RAM generator. The last result uses only one 15 bit RAM, where each 30 bit data needs 2 words.

6 REFERENCES

Berthet, C., Rampon, J. and Sponga, L. (1992) Synthesis of VHDL Arrays on RAM Cells, Proc. of EURO-VHDL, pp. 726-731

Camposano, R., Saunders, L. F., Tabert R. M. (1991) VHDL as Input for High-Level Synthesis, IEEE Design &Test of Computers, pp. 43-49

Krämer, H., Müller, J. (1992) Assignment of Global Memory Elements for Multi-Process VHDL Specifications, Proc. of ICCAD, pp. 496-501

Müller, J. , Krämer, H. (1993) Analysis of Multi-Process VHDL Specification with a Petri Net Model, accepted at EURO-VHDL

Narayan, S. , Gajski, D. D. (1994) Synthesis of System-Level Bus Interfaces, Proc. of EDAC, pp. 395-397

Narayan, S. , Gajski, D. D. (1994) Protocol Generation for Communication Channels, Proc. of 31th DAC

7 BIOGRAPHY

Jens Müller received the diploma degree in computer science from the technical university of Dresden in 1992. Then he started to work at the computer science research center (FZI) at Karlsruhe. His fields are the high-level synthesis and system-level synthesis. His special interests are the synthesis of the communication structures of complex systems, the exploration of different target architectures and performance estimation at system-level.

Ramayya Kumar completed his Ph.D. at the Indian Institute of Technology (IIT) Delhi in 1986. He worked as a scientific staff in the Department of Electrical Engineering at IIT Delhi between 1980-1986. From 1986-1988 he worked at the corporate research centre of Siemens in Munich, where he was involved with the development of the concepts for the CALLAS System. During 1989-1993, he worked at the University of Karlsruhe where he founded the hardware verification group. Since 1993 he heads the design automation group at the Computer Research Institute (FZI) Karlsruhe. His areas of interest are formal methods in hardware, synthesis and CAD frameworks, in which he has published over 40 papers.