Synthesis of VLSI Systems with the CAMAD Design Aid

Zebo Peng

Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract

CAMAD is a high level design tool which helps designers to model, analyze, and design VLSI systems. This design aid system is based on a unified design representation model derived from timed petri nets and consisting of separate but related models of control and data parts. The present paper describes the automatic synthesis package of the CAMAD system which takes a high level behavioral description as its input and synthesizes it into an implementation structure. This implementation structure may then be partitioned into several quasi-independent modules with well-defined interfaces, which allows potentially asynchronous operation of the designed systems as well as physical distribution of the modules.

1. Introduction

CAMAD (Computer Aided Modelling, Analysis, and Design of VLSI Systems), a system level design tool, is currently under development at Linköping University. CAMAD is built around a unified design representation model described in [9], which consists of separate but related models of control structures and data parts. The control part is modelled as a timed Petri net with restricted transition firing rules. The data part, on the other hand, is represented as a digraph. This extended timed Petri net (ETPN) model differs from other Petri net models used mainly for descriptive and analysis purposes [10] in that it addresses the issue of design directly and allows graphical representation of the behavior as well as structure of a system.

This paper describes the synthesis of VLSI systems specified by their high level behavioral descriptions into implementation structures with the CAMAD synthesis package. Other DA systems which start with a high level behavioral description are CMUDA[2], ADAM[3], and MIMOLA[8]. Most of these systems have concentrated on the synthesis of centralized processing units. The present approach, however, attempts to address the design of decentralized systems where it is important to partition a system into a set of loosely coupled modules so as to allow asynchronous operation of the designed systems as well as physical distribution of the modules.

The synthesis task is accomplished by a series of simple ETPN transformations each of which guarantees the preservation of the system semantics. One important characteristics of the present approach is that the ETPN model allows us to view a behavioral description as a *primitive* structural description. This structural description is, of course, very crude, i.e., if we implement it directly, we get a very expensive design. But once we have a structural description, we can make improvement on it to produce a better one; moreover this improvement can be done step by step until a satisfactory result is reached. This improvement process is guided by an optimization strategy that decides which transformation algorithm to use in each step. The basic optimization trade-off here is silicon area vs. time which can be made by trading data paths for control structure, or *vice versa*. Unlike most other control/data path allocations that only attempted to allocate one dimension of the control/data path problem at a time, we integrate them together in the synthesis process. Therefore, the advantages from compacting the data part, for example, can be compared immediately with the possible expansion of the control logic to justify the compaction.

This work is part of the ASAP project (An Architectural Strategy for Asynchronous Processing) at Linköping University. We are aiming at developing CAD tools and architecture support for a design environment in which asynchronous concurrent systems can be first specified without regard to the detailed implementation or packaging of their modules and then either allow the designer to explore the performance and cost implications of different implementations or have the system attempt to automatically provide an optimized solution based on a library of functional implementations [6], [7], [9].

The ETPN model and its basic transformations have been discussed in [9]; the present paper will concentrate on the problem of how to utilize this model in the synthesis process of VLSI system and to partition an ETPN description into ASAP based modules. In section 2, we describe briefly the major characteristics of the ETPN design representation which serves as the intermediate specification of the designed systems. The synthesis process based on this model is then discussed in section 3 together with a description of the CAMAD synthesis package. In section 4, we describe the module partitioning problem and its solutions. Finally we summarize the discussions in section 5.

2. The ETPN Design Representation

The ETPN model consists of a data part represented as a digraph where the nodes are used to model data manipulation units (arithmetic operators, storages, etc.) and the arcs are used to model the communication paths between them. These nodes and arcs are *abstract* models which may have different levels of granularity. For example, a node may be used to model a register with one input port and one output port. In other cases, a node may represent a large VLSI component, such as a microprocessor. The combination of such high level function modules (other examples are a CPU or a memory module) with low level elements like gates in the same

0738-100X/86/0000/0278$01.00 ©1986 IEEE

specification results in a highly flexible and efficient cell-based strategy.

Further, the ETPN model utilizes a hierarchical strategy to deal with different levels of details of the designed systems; each node of the data part, for example, can be itself another very complicated digraph represented in ETPN forms. In this way, the ETPN design representation is capable of capturing abstract information so as to give designers as well as design tools freedom in the implementation phase to make trade offs to reach optimal solutions. It also provides, on the other hand, possibilities for a designer to predefine lower level details when he feels it is necessary to freeze some implementation decision in the higher level in order to cut down the design search space or to make use of some standard components stored in a cell library, for example.

The ETPN data part is controlled by a control engine that produces a sequence of control signals to evoke the data part in an order consistent with the behavior of the designed system. This control part is modelled as a timed Petri net which requests a token to reside in a place for some period of time before it can be used to enable a transition [9], [11]. This time interval represents the time required to finish the associated operations in the data part.

The use of Petri nets provides the ability to directly describe concurrency and parallelism. Petri nets also allow explicit asynchronous description of control (some of them may later be converted to synchronous control structures). In the ETPN model, no assumption about the existence of a centralized clock or a clock hierarchy is made; though a local clock mechanism will later provide necessary clock signals to synchronize operations within each isochronous region.

The behavior of a Petri net is also non-deterministic, which makes it very difficult to analyze it and to use it as design representations. In the present approach, we have excluded the non-deterministic aspect of Petri nets by introducing a restricted transition firing rule; a transition will fire *immediately* when it is enabled and the guard condition is true (condition signals are produced by the data part as the result of some operations). This restriction results in the reduction of complexity of the "reachability tree" by a significant order, thus reducing the complexity of analysis of the control Petri nets in our model [9].

3. The CAMAD Synthesizer

The overall structure of the CAMAD synthesizer is depicted in Fig.1, where a high level behavioral description is first transformed into a data flow representation and then into the intermediate representation of the ETPN model which consists of a data part and a control structure. The ETPN representation is then manipulated by a set of semantics-preserving transformation algorithms, which collapse the possible data elements or control elements to reflect the decision to share hardware resources. The module partitioner is responsible for dividing the design into submodules and designing communication protocols of the interfaces between them. Both the transformation and partitioning processes are guided by heuristic search strategies. In addition to the automatic mode, however, all of the algorithms in the CAMAD synthesizer also provide user interactions to allow designers to override the decision made by the system, or to make their own design decisions.

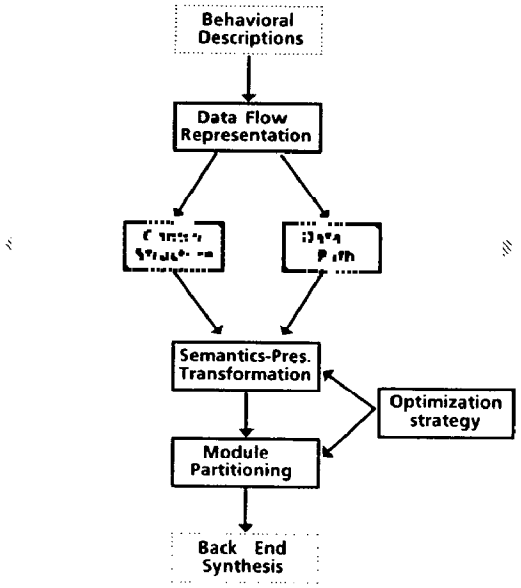

Fig.1 Overview of the CAMAD synthesizer

CAMAD is driven by a generic menu handling system, which allows the reconstruction of the structure of CAMAD as implementation proceeds. This menu handling system is used to interface the design algorithms and designers in an interactive way via a terminal. The structure of the menu is separately stored in a form of dimensional flowchart outside the system. An interactive tool for creating and manipulating dimensional flowcharts, called DIMsystem, developed at Linköping University [4], is used to create such a menu structure. If the menu structure is not satisfied or new menu item is to be added into the system because of the creation of new algorithms, we can use the DIMsystem to change the menu structure and run the menu constructor program to build a new structure of the whole system.

In this way, the designers can also reconstruct the menu to suit their particular design requirements or personal tastes. This is considered as one way to resolve the expert/layman conflict in expectation for man-machine interface design. An experienced user of the system would like to go directly into the appropriate procedure as quickly as possible; a new user, on the other hand, would like the system to provide as much help information as possible to guide him to the desired place.

3.1 Construction of the ETPN Representation

Unlike intermediate design representations such as the ETPN model, a product specification, i.e., the input to the design aid system, is more desirable to be a *high level* behavioral description language, e.g., ISPS [1]. A high level behavioral description specifies only the functions the hardware must be able to perform without prescribing the physical structure of the implementation. This will free the designers from the burden of selecting a good implementation structure and allow them to concentrate on the functionality of the system. The synthesis algorithm must then be able to transform such a high level behavioral description into a set of function modules or blocks that as a whole will implement the specified semantics.

The first step of synthesizing VLSI systems into their implementation structures in the CAMAD environment is then to transform a behavioral description into its ETPN representation. This transformation is divided into two phases, as illustrated in Fig.1. First the high level behavioral description is transformed into its equivalent data flow representation with each node representing one instruction and an arc from one instruction node to another meaning basically that the second instruction may not be executed until the first has been completed.

The second step is to transform this data flow representation into the ETPN representation, which is quite straightforward. Basically each node of the data flow representation has its corresponding place in the control part. An arc which represents a data dependency between two nodes is modelled as a transition. If the dependency is conditional, the corresponding transition is also conditional, i.e., guarded by a condition signal coming from the data part (we assume each of the condition signals is created by some dedicated operation). A node which depends on two or more nodes will be mapped into a synchronization transition.

The image of the data flow representation on the data part digraph is formed by the convention that each variable will have only one place of resource, i.e., each variable is assumed at this point to be implemented by one register. All of the operations, on the other hand, will have their own copy of the resource, i.e., even if 10 instances of the + operation appear in the data flow representation and all are going to be implemented by a single ADD unit, they will be present as 10 different operators at this point.

The arcs that represent the connections of these data manipulation units will be used to model the data communication between them. If there is some data dependency between a set of data units, e.g., $A := B + C$, then there will be arcs connecting the output port of B and of C to the input ports of the adder, and an arc connecting the output port of the adder to the input port of A. These arcs will be guarded by the corresponding control signal for this data flow graph node in the control net. In this way, it is clear that the addition of B and C is performed and the sum is fed into A only when the corresponding control signal is on (a token residing in a place represents a control signal being sent to all of its guarding arcs), as illustrated in Fig.2.

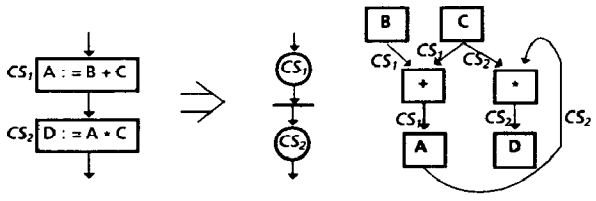

Fig.2 An ETPN example

During the above transformations, it is assumed that a potentially unbounded number of abstract function units (operators) are available; therefore, the ETPN representation at this point is still more or less a data flow representation rather than a physical implementation. The important property of this ETPN representation, however, provides the possibility of applying a set of simple synthesis algorithms to reduce the required number of function units as well as data paths to a level where each node of the data part can be implemented by a physical hardware component and each arc by a physical connection. The control part, on the other hand, can be implemented by microprograms, PLAs, or dedicated circuits.

Further the produced ETPN representation has a set of important properties which help to avoid some traditional difficulties of utilizing Petri net models due to the complexity of analysis [10]. Examples of such properties are safeness and conflict-freeness. Safeness is an essential property for hardware design using Petri net models, because it is erroneous to have two operations going on in the same operator simultaneously. On the other hand, conflicts are excluded from our model due to the way we interpret the places of the nets; a token coming into a place in our model indicates that some operations have been started in the data part. When the token leaves the place, the associated operations are supposed to be finished; consequently, new operations can be started. If a token in a place can make, for example, two transitions firable at the same time, it means that the two sets of operations related to these two transitions can be performed after the associated operations of this place finish. To fire one of the transitions, however, will disable the other, resulting in a contradiction.

To solve the above problem, we can either allow one token to fire more than one transition, or exclude the case where one token can make more than one transition firable. The latter solution is chosen in the present approach. Therefore, if the completion of an operation will enable more than one operation in the data flow representation, we will duplicate its associated place to form a set of *linked places*, as illustrated in Fig.3 (CS_{11}, ..., CS_{1k}). All of the linked places of the same data flow node will have the same amount of execution time; but only the first place, CS_{11} in the example illustrated in Fig.3, is associated with the corresponding data part. The other places are used here only as auxiliary places to make the produced net conflict free and at the same time keep the semantics of the designed system unchanged.

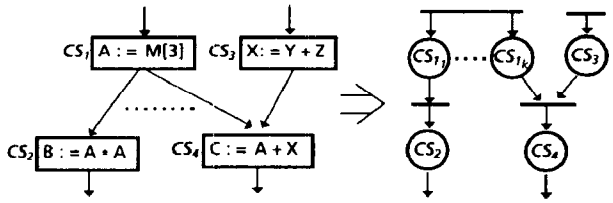

Fig.3 The linked places

3.2 Basic Transformations

After the input behavioral description which may consist of functions at different levels and in different forms is transformed into the ETPN representation which consists of only well defined functional primitives, it will be analyzed and reconstructed until a satisfactory result is produced. The reconstruction process is carried out by a set of basic transformation algorithms discussed in [9]. The basic principle here is to share components as much as possible, which can be done by trading performance for resources. The major resources to be considered here is the operators represented by the nodes and the connections denoted by the arcs.

One example to share operators is to do an *Operator Merger* as illustrated in Fig.4, which folds two operator nodes into one. It is easy to see that if *CS1* and *CS2* will not have tokens appearing at the same time, such a merger will not change the semantics of the designed system. Other examples of mergers are *Connection Merger* to share physical bus by logic connections and *Constant Merger* to collapse the data part and possibly control part where the computation can be executed in the design time. The later is similar to the constant folding technique for code improvement in optimizing compiler.

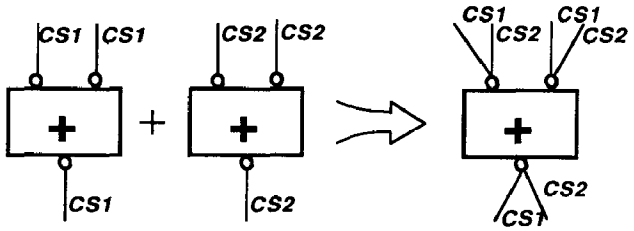

Fig.4 An example of operator merger

Other transformations may also involve the control part of the designed system. For example, we can stretch those control places that are in the form of concurrency into a sequence of places in order to facilitate an operator merger or to reduce the complexity of the control structure during the implementation.

3.3 Optimization Strategy

The choice of a sequence of transformation algorithms to apply to a ETPN representation affects the final result of the design; one transformation may, for example, prevent another one with more effective gains. A general design space search strategy is needed to achieve an optimal solution. The goal of our optimization is the overall system performance and the estimated layout area requirement. More specifically, optimization is measured by a set of cost matrices, each of which represents an aspect of the design.

Some of these matrices associated to the nodes of the data part are used to store the cost of the function of each node in terms of implementation (e.g., area requirement); estimated time of delay at the nodes; and the frequency of the using of this function units. Cost matrices associated with the arcs of the data part are used to store the cost of the communications in terms of implementation; the estimated time of delay at this connection; and the data bandwidth required by this connection.

Given such a set of matrices and a vector of scales each of which represents the weight of a matrix (hence an aspect of the system), an estimated design complexity measurement can be calculated. The optimization algorithm will then try to minimize this complexity measurement, thus creating an optimal design, by applying a sequence of transformation algorithms. The weight vector allows the designers to stress some aspects of the design over others so as to suit different design tastes. It can also be used to produce a series of designs based on the same specification but running at different speeds or occupying different areas of silicon space.

A systematic way to reach the optimal solution is to search for all possible sequences and compare the complexity measurements of their final results; the best will then be selected, which is a *NP* complete problem. However, the present approach provides a basis for heuristic solutions to be developed. The general strategy we use is an iterative improvement strategy which starts with the system in a known configuration and applies an operation to each part of the system in turn until a rearranged configuration that improves the design is discovered. The criteria used for the choice of subparts to be improved are the critical path and critical signal. In each synthesis step, a critical path/signal is calculated, and the related part is improved by an appropriate transformation until a better design is achieved. This strategy will not always create an optimal design, but in most of the design cases, it assures a near optimal solution.

In the ETPN model, a set of operations in the data part is supposed to finish within the period of time when the associated control signal is on. The length of the time for a control signal to keep "high" must then be decided by the time taken by the most time consuming operation. The associated path is the critical path for this control signal. The time between two control signals, i.e., two set of operations, is also defined by the worst case of paths between this two places. For example, a transition may have two input places, one associated with a very long operation, the other associated with a quick one. The later must then always wait for the former to finish before the transition can be fired. The former place together with its preceding places if any also forms a critical path. Critical signals, on the other hand, are the signals whose path lengths and associated delays are critical to the *overall performance* of the designed system. The property of critical signal analysis differs from that of critical path analysis in that the dynamic feature of the system is taken into account in the former case, but not in the latter one.

4. Partitioning of Systems

In this section, we will discuss the problem of partitioning a system, represented in ETPN forms, into a set of ASAP based modules which can operate at different rates of speeds. Formally a partition of the ETPN representation of a system is a set of arcs (in the data part) and transitions (in the control part) which separate the data part into a set of data sub-parts and the control into a set of control sub-parts. A control subpart must match a data subpart, thus forming an ASAP module. The major criteria for the partitioning algorithm are the communication required between the partitioned modules. A partition must, of course, also satisfy constraints of size, number of unique assemblies, and pin-out.

The proposed partitioning algorithm consists of two parts. The first part is a stepwise abstraction phase in which the ETPN representation is transformed step by step towards a representation with a higher level of granularity. That is, in each step of the transformation, some possible reconstruction of the data part and the control structure is done in order to build blocks of control part (data part) such that each block can be replaced by a single place (data node). The second part utilizes a divide and conquer algorithm for the final partitioning.

4.1 Stepwise Abstraction

One of the major weakness of using Petri nets as modelling tools is that as the number of places and transitions increases, the analysis of all possible interactions becomes almost impossible. One way to solve this problem is to abstract

subnets into places (transitions) before analysis is carried out. This will significantly reduce the number of the nodes in the net so as to reduce the time required for the final partitioning where a through analysis of the net is necessary to produce an optimal solution.

Techniques for making abstraction of Petri nets while preserving important properties like safeness, conflict-freeness and liveness have been discussed in [12], [13]. One technique we employ here is to incrementally search the control Petri net for subnets that are *well-formed* blocks [13]. However, when deciding whether a well-formed subnet is going to be abstracted into a single place in the present approach, we must also take into account whether its corresponding data sub-parts are tightly connected to each other. The analysis of this connectivity is, nevertheless, quite straightforward because each time only a small part of the design is considered.

Another factor that should be considered when replacing a subnet by a single place is whether the abstraction will result in possible abstraction of the data sub-parts. A data sub-part is said to be *well-formed* under a well-formed control block if its internal arcs are guarded only by control signals corresponding to those places in the control block. If a well-formed control block is abstracted into a single place, all of its associated well-formed data blocks can be also abstracted into data nodes. One of such examples is illustrated in Fig.5.

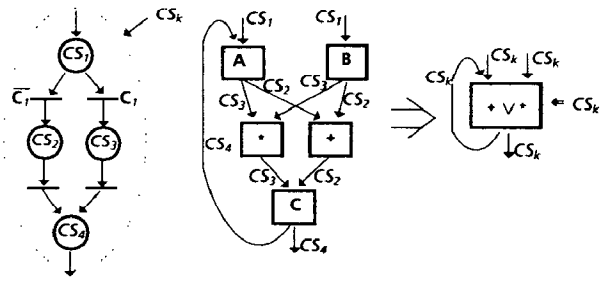

Fig.5 An example of control and data block abstraction

4.2 Construction of the Partitioning Graph

The final partitioning of systems into modules can be formulated as a graph partitioning problem where a system is formalized as a graph with the components being the vertices and the interconnection the edges. Associated with the vertices and the edges are some forms of cost. The partitioning algorithm decomposes the graph into a set of subgraphs so as to minimize the sum of the "cost" on all cut edges under a set of constraints. What we are going to describe here is a way to transform the abstracted ETPN model into a single graph representation with cost measurement on the edges so that some existing effective algorithms for graph partitioning, for example, those discussed in [5], can be unitized.

The separation of the data part and the control part makes the partitioning problem very difficult because of the references between two graphs. One way to solve this problem is to take one graph as the basis and reduce the references to the other as much as possible. We have chosen to take the control Petri net as the basis of the partitioning graph, i.e., the partitioning graph is similar to the topological structure of the control Petri net. In the partitioning graph, however, both places and transitions of the Petri net are represented as vertices. Connections between places and transitions, on the other hand, are represented as edges that are called C-edges.

To reduce references to the data part digraph during the partitioning process, we must have a way to represent the structure of the data part in the partitioning graph. For this purpose, those vertices representing originally Petri net places are connected by D-edges which capture the connectivity information of the data part. If the corresponding data sub-parts of two Petri net places share the same data manipulation units, the "cost" assigned to the D-edge which connects their associated vertices in the partitioning graph will be equal to Wd, a weight indicating the importance of the data part in the partitioning algorithm. If their corresponding data sub-parts are only connected by data paths without sharing data manipulation units, the "cost" on the D-edge will be between zero and Wd depending on the bandwidth of the data paths. In the case where the corresponding data sub-parts do not have any connections, the "cost" of the D-edge equals zero, and this D-edge can then be taken away to reduce the complexity of the graph.

The "costs" of the C-edges, on the other hand, capture the effect of partitioning upon system performance and will be calculated by an incremental algorithm. Starting from the vertices representing the initially marked places, the algorithm assigns the "cost" of all their output edges (i.e., the edges representing the output arcs of the places) to be Wc, a weight indicating the importance of the control part in the partitioning process. The algorithm will then visit and assign "cost" to each C-edge of the graph in turn by following the C-edges in the direction of the corresponding Petri net arcs. The rules for calculating new "costs" based on the previous ones are given in Fig.6, where f_p is the probability of a token in the place being used to fire its pth alternative output transition. This dynamic information of the Petri net execution can be collected from simulation or by the analysis of the application algorithms. Note also that $f_1 + f_2 + \ldots + f_k$ should equal 1 for every place (k is the number of output transitions of the place).

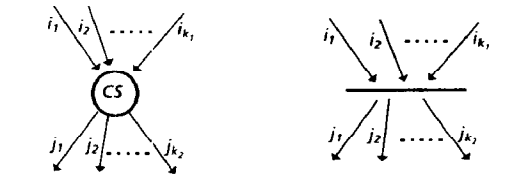

Fig.6 C-edge cost calculation rules

In the case of a loop, an estimation of average number the loop body is repeated each time it is entered should be provided to the algorithm. This average repeating number can be again collected from simulation or by the analysis of the application algorithms.

After the ETPN model of the designed system is transformed into a graph with cost measurement on all edges, a graph partitioning algorithm can be utilized to do the final partitioning. We use a divide and conquer method discussed

in [5], where the system graph is first divided into two roughly equal size submodules. Each submodule is then divided into two sub-submodules. This process is continued until sufficiently small modules are achieved. In each step of this process, the algorithm consists of two parts, a constructive part for preliminary partitioning, and an optimization part for iterative improvement of the solution. The designers can change W_d and/or W_c to emphasize the data part (mainly silicon area concerned) or the control structure (mainly performance concerned) in the partitioning process to suit particular requirements of a design instance.

4.3 Introduction of Clocks

As previously mentioned, the ETPN model makes no assumption about the existence of a global clock or a clock hierarchy in the designed system. As the design proceeds, however, those operations that take too much time will be expanded into sequences of primitive activities each of which will finish approximately within a prescribed unit of time in order to facilitate the implementation. Nevertheless, it is only after the system is partitioned into a set of modules that a clock mechanism is introduced to each module forming an isochronous region. At that moment, we can assume that each transition within a module will be synchronized by a clock signal and each place will hold a token for normally one clock cycle or a number of clock cycles of time (when it waits for synchronization, or for some conditions to become true, etc.).

4.4 Interface Protocols

The partitioning algorithm divides a system into a set of modules connected to each other by arcs (in the data part) and transitions (in the control part). Note that there are also implicit connections formed by the splitting of control signals and conditional signals from their guarding arcs and transitions respectively. The arcs and transitions between two modules together with the possible control signals and conditional signals travelling between them form the abstraction of their interface. Details of the interface protocol, however, must be designed.

We have chosen to use an embedded approach, namely, a three hand shaking protocol to interface different modules of a system, which allows message to be sent asynchronously from one module to another. The module which initiates the data transmission first makes sure that the other side is ready before it can start the data exchange process by sending a control signal. After the data has been sent, it makes sure that the data transmission is successfully accomplished by testing a condition signal (acknowledgement) from the other side. The cooperating module, on the other hand, must contain mechanisms for receiving the control signal and producing the condition signals according to the real time situation. This mechanism is designed as part of the data manipulation unit which participates in the communication.

5. Conclusions

We have described a VLSI system level design tool and the design methodology it supports. This design environment is based on a unified design representation, the ETPN model. One of the features of this approach is its ability to design asynchronous concurrent systems from high level behavioral descriptions which make no assumptions about modules structure or clocking strategies of the implementations.

The ETPN model is designed to be a multi-level design representation, which facilitates the implementation of the CAMAD design aid system built on top of it. The CAMAD synthesizer consists of a set of algorithms most of which take an ETPN representation as their input and produce as output another ETPN representation. Consequently, most of the algorithms are quite simple but still powerful because they can be iteratively utilized until satisfactory results have been achieved. This iterative improvement strategy coupled with the unified design representation results in an integrated design environment.

Further, the use of Petri net model for the control part and digraphs for the data part allows a direct mapping of the ETPN model into graphs. Therefore, it is possible to utilize graphic means to interface the designers and the CAMAD design aid. Graphic representations are also particularly useful for the display of dynamic behavior of the designed systems, which is considered very important for helping the designers to grasp the dynamic aspects of the implementation. A graphic simulator for vividly displaying the flowing of data and control signals in the designed systems is very desirable for this purpose. This part of the work, however, has not been implemented yet.

6. Acknowledgements

I am very grateful to Dr. Bryan Lyles and Prof. Harold W. Lawson Jr. for their helpful guidance and suggestions in the present work. I would also like to thank Tony Larsson, Mikael Patel, Johan Fagerstrom, and Krzysztof Kuchcinski for their comments on parts of this paper. This research project is partially supported by grants from STU — the Swedish Board for Technical Development.

7. References

[1] Barbacci, M.R., Barnes, G.E., Cattell, R.G., and Siewiorek, D.P., *The ISPS Computer Description Language*, Tech. Report, Dept. of Computer Science, Carnegie-Mellon Univ., 1977

[2] Director, S.W., Parker, A.C., Siemiorek, D.P., and Thomas, D.E. Jr., *A Design Methodology and Computer Aids for Digital VLSI Systems*, IEEE Trans. Circuits and Systems, Vol.28, No.7, July 1981

[3] Granacki, J., Knapp, D., and Parker, A., *The ADAM Advanced Design AutoMation System: Overview, Planner, and Natural Language Interface*, Proc. 22nd Design Automation Conf., 1985

[4] Jönsson, A., and Patel, M., *An Interactive Flowcharting Technique for Communicating and Realizing Algorithms*, Proc. 19th Hawaii Int. Conf. on System Sciences, Jan. 1986

[5] Kerninghan, B. W., and Lin, S., *An Efficient Heuristic Procedure for Partitioning Graphs*, Bell System Tech. J., Vol.49, Feb. 1970

[6] Lawson, H.W. Jr., and Lyles, J.B., *An Architectural Strategy for Asynchronous Processing*, Proc. IFIP 10.3 Workshop on Hardware Supported Implementation of Concurrent Languages in Distributed Systems, IFIP 1984

[7] Lyles, J.B., *CAD Approaches for an Asynchronous Architecture*, Proc. Nordic Symp. VLSI in Computers and Communications, Tampere, Finland, 1984

[8] Marwedel, P., *The MIMOLA Design System: Tools for*

the Design of Digital Processors, Proc. 21st Design Automation Conf., 1984

[9] Peng, Z., *A Formal Approach to the Synthesis of VLSI Systems From Their Behavioral Descriptions*, Proc. 19th Hawaii Int. Conf. on System Sciences, Jan. 1986

[10] Peterson, J.L., *Petri Net Theory and the Modeling of Systems*, Reading, Prentice-hall, 1981

[11] Sifakis, J., *Petri Nets for Performance Evaluation*, in *Mersuring, Modelling, and Evaluating Computer Systems*, Proc. 3rd Int. Symp. IFIP Working Group 7.3, H. Beilner and E. Gelenbe, Eds., 1977

[12] Suzuki, I., and Murata, T., *A Method for Stepwise Refinement and Abstraction of Petri Net*, J. of Computer and System Sciences, Vol.27, 1983

[13] Valette, R., *Analysis of Petri Nets by Stepwise Refinements*, J. of Computer and System Sciences, Vol.18, 1979