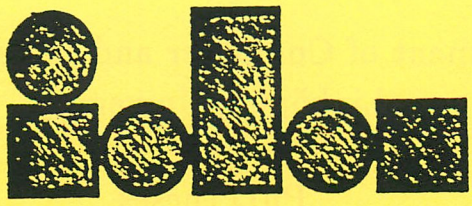
KB-RA
E 2

LiTH-IDA-R-88-39

ISSN-0281-4250



INSTITUTIONEN FÖR DATAVETENSKAP
UNIVERSITETET OCH TEKNISKA HÖGSKOLAN
LINKÖPING

CAMAD: A Unified Data Path/Control Synthesis Environment

by

Zebo Peng
Krzysztof Kuchcinski
Bryan Lyles

UNIVERSITETSBIBLIOTEKET
88 -10- 17
LINKÖPING

This paper will be published as a part of the Proceedings of the IFIP TC-10 Conference on Design Methodologies for VLSI and Computer Architecture, Pisa, Italy, Sept. 19-23, 1988

RESEARCH REPORT
CADLAB, September 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

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

The Department of Computer and Information Science
Linköping University

PhD theses:

* Available at: University of Microfilms Intl., 300 N. Zeeb Road, Ann Arbor, MI 48106, USA.

(*Linköping Studies in Science and Technology. Dissertations.*)

*No 77	**Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982.
No 94	**Hans Lunell:** Code Generator Writing Systems, 1983.
No 97	**Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983.
*No 109	**Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984.
*No 111	**Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984.
*No 155	**Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons. 1987.
No 165	**James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning. 1987.
No 170	**Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems. 1987.
No 174	**Johan Fagerström:** A Paradigm and System for Design of Distributed Systems. 1988.

Licentiate of engineering Theses:

(*Linköping Studies in Science and Technology. Theses.*)

No 48	**Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching. 1985.
No 52	**Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems. 1985.
No 60	**Johan Fagerström:** Simulation and Evaluation of an Architecture based on Asynchronous Processes. 1985.
No 71	**Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System. 1987.
No 72	**Tony Larsson:** On the Specification and Verification of VLSI Systems. 1986.
No 73	**Ola Strömfors:** A Structure Editor for Documents and Programs. 1986.
No 74	**Christos Levcopoulos:** New Results about the Approximation Behaviour of the Greedy Triangulation. 1986.
No 104	**Shamsul I. Chowdhury:** Statistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology. 1987.
No 108	**Rober Bilos:** Incremental Scanning and Token-based Editing. 1987
No 111	**Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments. 1987.
No 113	**Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge. 1987.
No 118	**Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs. 1987
No 126	**Dan Strömberg:** Transfer and Distribution of Application Programs. 1987
No 127	**Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems. 1987
No 139	**Christer Bäckström:** Reasoning about Interdependent Actions. 1988.
No 140	**Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing. 1988.
No 146	**Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System. 1988.

LiTH-IDA-R-88-39 ISSN-0281-4250

CAMAD: A Unified Data Path/Control Synthesis Environment

by

Zebo Peng
E-mail: zpe@ida.liu.se

Krzysztof Kuchcinski
E-mail: kku@ida.liu.se

Bryan Lyles
Xerox PARC

Abstract: This paper describes a set of design tools built at Linköping University to support automated synthesis of integrated circuits. They consists of algorithms for 1) automatic generation of unified design representations from high level behavioral descriptions, 2) synthesis of optimal data path and control, and 3) partitioning of VLSI systems into asynchronous modules. The set of design tools is integrated to form a unified design environment based on a design representation model derived from timed Petri net notation. Several ongoing projects are being carried out to enrich the design environment and will also be discussed in this paper.

This research has been supported by the Swedish National Board for Technical Development (STU).

This paper will be published as a part of the Proceedings of the IFIP TC-10 Conference on Design Methodologies for VLSI and Computer Architecture, Pisa, Italy, Sept. 19-23, 1988

Postadress:
Institutionen för datavetenskap
Universitetet i Linköping och
Tekniska Högskolan
581 83 Linköping

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

1. Introduction

For the past five years the Computer Aided Design Laboratory at Linköping University, CADLAB, has been developing a design methodology for integrated circuits. We start with a high level behavioral description and then compile an implementation which may be either totally synchronous or may contain a mixture of synchronous and asynchronous systems. The fundamental goal of the CAD system is to be able to represent, and make trade-offs between the three elements of a VLSI system: control, communication and computational elements. This paper summarizes the progress to date and discusses ongoing work.

The design methodology we employ is based on the extended timed Petri net (ETPN) notation [15]. ETPN notation unifies the representation of both the data and control parts of the design throughout the design cycle. When the behavioral description is translated to the initial ETPN model it can be viewed as a primitive structural description; correctness-preserving transformations then successively transform this initial model to a final implementation. The initial ETPN model usually starts as a totally asynchronous system, the transformations introduce increasing degrees of synchronous timing. The integrated representation of control and data allow the optimization process to consider the space/time trade-offs between data paths and control, synchronous and asynchronous timing.

CAMAD is the name given to the set of tools developed at Linköping. It includes tools for 1) creating ETPNs from high-level behavioral descriptions 2) transforming and optimizing ETPN models with respect to a given implementation technology 3) partitioning the ETPN into loosely coupled modules so as to allow physical partitioning of the final design and 4) ultimately outputting netlists to place and route programs.

The remainder of this paper is divided into three parts: background, the ETPN design representation and its associated transformations and optimization, and a section describing ongoing work.

2. Background and Motivation

Behavioral level specifications should be applicable to a wide range of physical realizations. To the extent that issues associated with physical realization, such as system timing, become part of the behavioral specification, multiple realizations and reuse are inhibited. Asynchronous behavioral specification allows deferring clocking issues until silicon compile time, thus increasing the specification's generality and the probability of finding the optimal design.

The need for asynchronism between portions of compiled silicon is driven by three factors: physical issues which limit chip-wide distribution of synchronizing signals, the desire to build systems from functions compiled by different silicon compilers, and

packaging constraints which prevent full knowledge of components internal state. These issues force designs towards a structure which consists of multiple communicating machines, each of which contains datapaths and associated control. Internal clocking for these machines may then be made independent of the timing associated with communication between them.

Shrinking VLSI geometries promise denser chips and decreased switching times at the gate level. Unfortunately, while gate delays scale linearly, the RC-line delay for communications between gates does not scale, thus leading to a situation where the chip's speed is limited by the interconnections rather than gate delays. As circuit speeds increase, the costs of communication become dominate and synchronous operation is revealed to have only been an approximation. Furthermore, when systems are split over multiple packages, pin-out limitations force the multiplexing of connections, thus dramatically increasing the complexity of global control.

When subsystems are designed by different designers or silicon compilers, expecting globally consistent timing is unreasonable. We do not believe that there will be a single chip description language and its associated silicon compiler for all applications. Nor will system designers wish to pay for the fabrication of new chips if existing silicon supplies the necessary functionality. This being the case, we must be prepared to connect modules designed by different compilers (including by hand layout) with different internal timing conventions. Silicon design methodology must be object oriented: each subsystem must be highly modular and should be viewed as a black box by other modules. An asynchronous module is naturally bounded; internally it has its own internal speed, critical paths, and storage.

In order to meet the constraints given above, CADLAB embarked on a project called ASAP, A Strategy for Asynchronous Processing [7]. The fundamental idea of ASAP was to define a strategy for interconnecting asynchronous functional units, a strategy that was very general even if by itself impractical. A CAD system would then compile away as much generality as was practical for specific systems.

The basic communication unit in ASAP is the *ports*. A port is to be realized as a physical register where a process can receive inputs or output results. Any given port may be in one of three configurations: 1) not currently used 2) input register or 3) output pointer to an input register. In the abstract machine, all ports may be potentially interconnected with one another. Having the output ports contain a pointer to an input port provided virtual circuits for the transmission of data between portions of the machine. The CAD system for ASAP then had to answer fundamental questions about the trade-offs between the advantages of decentralized routing of data (data paths) vs the overhead of establishing and destroying the virtual circuits (control).

The original vision of ASAP has not been fully implemented. Rather the research has focused on the problems of making trade-offs between centralized vs distributed control,

dedicated datapaths/functional units vs the control required to multiplex shared units, and synchronous vs asynchronous communication.

3. The CAMAD Design Aid Environment

This section introduces the set of synthesis algorithms in the CAMAD design aid system [13]. CAMAD is built around the ETPN formalism with its unified representation of data and control which is central to our ability to reason about design trade-offs. The overview of the CAMAD design aid system is illustrated in Figure 1.

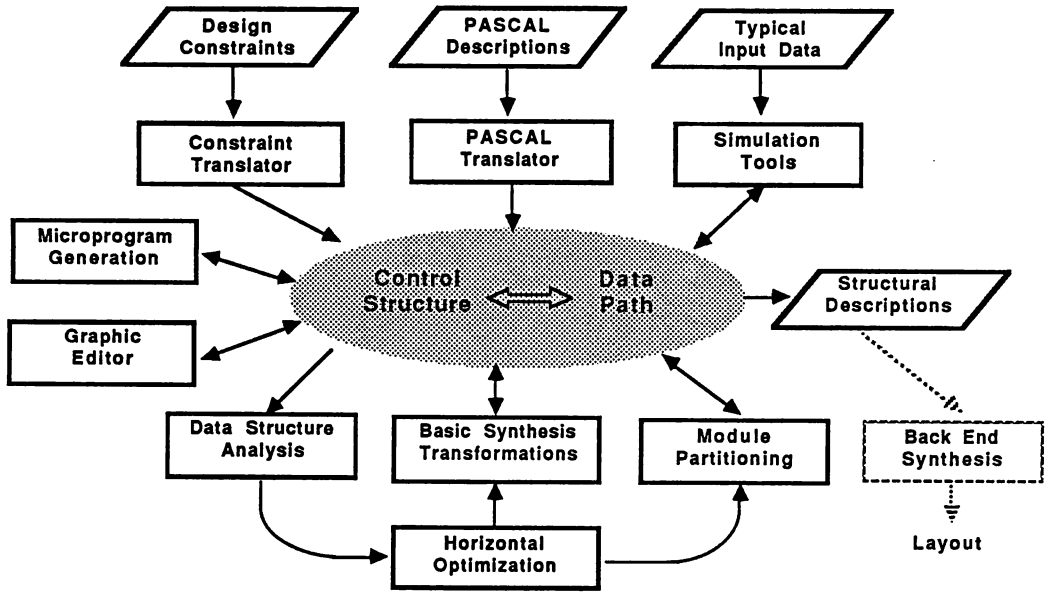


Fig. 1 Overview of the CAMAD design aid system

CAMAD consists of the synthesis algorithms as well as other supporting facilities for the design purpose. All of the algorithms interact with the ETPN design representation, which allows each algorithm to communicate freely with the others.

3.1 The ETPN Design Representation Model

The ETPN design representation consists of separate but related models of control and data path. The data path of the design representation is represented as a directed graph with nodes and arcs. The nodes are used to capture data manipulation units, which prescribe behaviors of the data manipulation units as mappings from a set of inputs to a set of outputs without giving the details of how the system performs the mappings. The arcs represent the connections of the nodes. The control part of the design representation, on the other hand, is captured as a timed Petri net with restricted transition firing rules [15]. These two parts are related by the control signals coming from the control part to

the data path and the conditional signal traveling in the opposite direction. One small example of the ETPN design representation is illustrated in Figure 2.

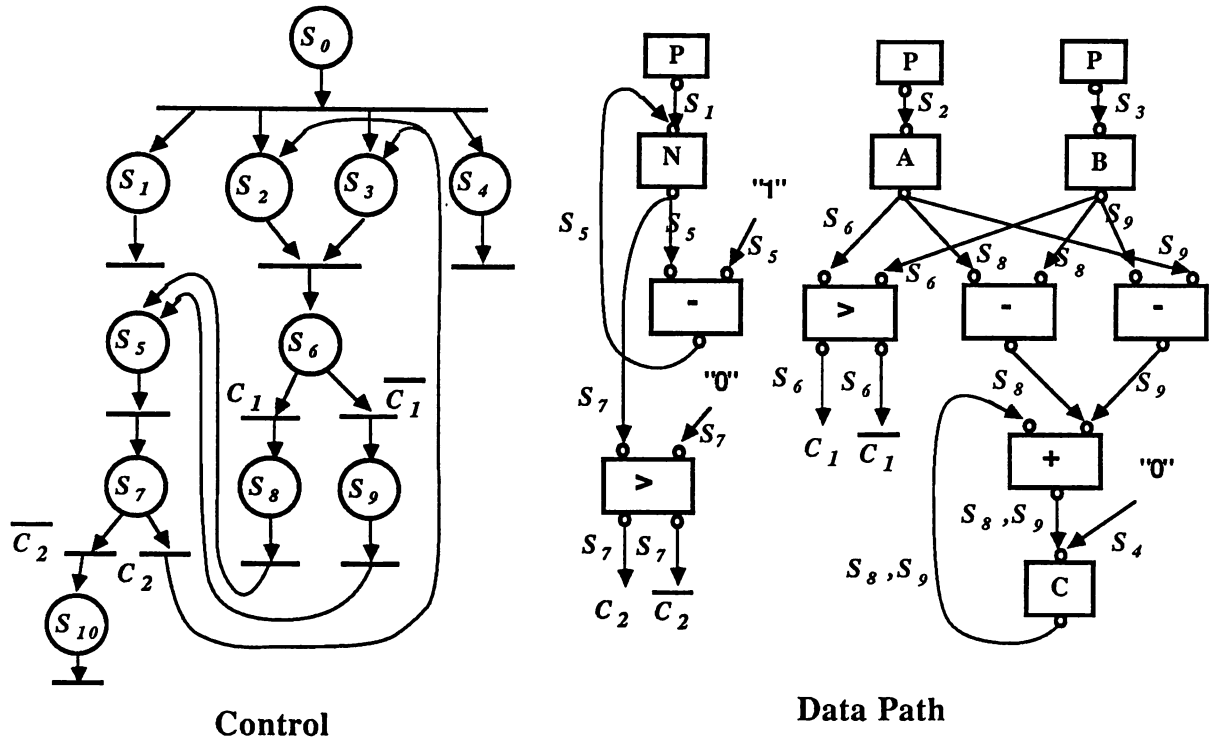


Fig. 2 An example of the ETPN representation

The right hand side of Figure 2 shows the data path of the design example where data path nodes are depicted as rectangles with labels indicating the functions of the nodes. Such labels are drawn from a set of predefined symbols which represents all of the functions supported by the synthesis algorithms. The arcs of the data path represent the data flow of the function nodes. Communication of data from one node to another is controlled by the control signals coming from the control part. The control relation is indicated by using control state labels to guard the arcs. The left hand side of Figure 2 depicts the Petri net which is used to represent the control flow of the designed system. A control state is represented as a marking of the Petri net, i.e., the possession of tokens in a subset of the places of the Petri net which are depicted as circles in Figure 2. The transitions of control states are represented as firings of one or several transitions of the Petri net which are depicted as bars in the figure.

As Petri nets are inherently concurrent [18], the ETPN design representation can deal with the design of distributed control for multiple processes executing concurrently. In a Petri net model, there can exist more than one event at a time as indicated by several places holding tokens simultaneously. More importantly, the progression of these events may occur independently; there is no need to synchronize these events unless it is required. When synchronization is needed, it is also very easy to model it. Another

advantage of a Petri net is its asynchronous property. There is no inherent implication of a clock mechanism for the firing operations. Only the partial ordering of the occurrence of events is specified.

The main feature of the ETPN design representation is its ability to capture the intermediate result of a design explicitly so as to allow the design algorithm to make accurate design decisions. For example, given an intermediate result represented in the ETPN form, an algorithm can be used to calculate its implementation cost, check whether that satisfies the design constraints, and automatically choose a transformation to apply to the design which produces another intermediate result with improved cost.

With the formulation of such a formal design representation, it is also possible to carry out verification of digital hardware at different levels, i.e., to prove that two designs which may be at different levels are equivalent to each other in respects to some axioms. For example, it is very important to verify that the behavioral specification of the design and its implementation structure are semantically equivalent to each other. The semantics of the ETPN design representation is defined in terms of its interactions with its environment [17]. That is, two designs are considered to be semantically equivalent if they interact with an environment in the same way.

The interaction of the hardware and its environment is defined by the *external events* [17]. To characterize the external events of the ETPN design representation for the purpose of synthesis and verification, they are defined in such a way that there is as much freedom as possible to alter parts of the data path and the control Petri net structure without affecting the interaction of the system to the environment. The minimal requirement is of course that the input to output functional relation should be the same, which is the most common criteria for equivalence studies in the software environment. However, this definition is usually too weak for most of the VLSI applications, because hardware components often have to interact with their environment in a predefined way. The temporal relationship among the set of input and output operations is also important. Therefore a semantic equivalence definition must also be able to express this aspect of design if needed.

The temporal relationship can be expressed either as a partial ordering of input/output or as absolute timing constraints. In our model, we use the partial order relations to represent the temporal relations of the external events of the ETPN design representation. Two systems are considered to be equivalent to each other only if the *specified* partial ordering of the input/output are the same in addition to the functional equivalence requirement.

In respect to the semantics so defined, the preservation of correct design is contained in the synthesis process. That is, proofs of the correctness are applied to the ETPN transformations, not to the individual design instances. Therefore each proof is only done once for each type of transformation. This approach results in a synthesis algorithm

which generates correct design from correct specifications.

3.2 Creating ETPNs from Behavioral Descriptions

With the formulation of the ETPN design representation model, the synthesis process is carried out by a sequence of transformations. A high level behavioral description is first transformed into a data flow representation and then into the intermediate representation of the ETPN model. The ETPN representation is then manipulated by a set of correctness-preserving transformation algorithms, which collapse the possible data elements or control elements to reflect the decision to share hardware resources, for example.

In the current implementation, we use PASCAL as a high level behavioral description language. More precisely, PASCAL programs are used to represent the functional behaviors of digital hardware, which will be used as input to our design system to generate the hardware implementation at register transfer level.

With the use of a conventional programming language as the input hardware specification language, we can make use of the existing software development environment built around it. For example, we can use a PASCAL compiler to compile a hardware description so that it can be run with test data to see whether it is functionally correct before the design process starts.

Before we discuss how a PASCAL program can be transformed into a ETPN design representation, we have to describe how it is interpreted as a behavioral specification of hardwares. In our approach, a hardware implementation is said to behave semantically the same as a PASCAL program if the external events of the two are the same. The external events of a piece of hardware are defined as the input and output operations associated with some ports which are connected to the outside world. For example, in the integrated circuit case, the pins of a chip are the only externally accessible components. In the PASCAL program, the *read* and *write* statements are considered as external events, which represents the interaction of the designed hardware to its environment.

To reduce the complexity, we have also introduced some restrictions on the PASCAL program structure at the current implementation. We can accept only PASCAL programs without recursive procedures. No dynamic data structure is allowed. For example, you are not allowed to use a pointer to create data storage during the execution of an algorithm. Further, only a subset of data types are allowed: integer, boolean, char, subranges and arrays of these.

The transformation of a PASCAL behavioral description into its ETPN representation is divided into two phases. First the 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. During this process, parallelism extraction is carried out so as to increase the performance of the design.

We have developed a stepwise refinement approach to perform the parallelism extraction operation. That is, a sequential data flow representation of a PASCAL program is reconstructed step by step toward the maximal parallel form. The parallelism extraction transformations are based mainly on the analysis of the data dependency relation between different statements. The data dependency relation is defined as the relationships between the statements or operations which contribute data to each other; in other words, two operations are data dependent if they must be executed in the predefined order to keep the semantics of the given sequential algorithm. Only if two sets of operations are data independent can they be performed in parallel.

The data dependency analysis is carried out iteratively using interval partitioning and place composition techniques so as to reduce the complexity of the extraction algorithm. The algorithm first partitions a graph into a set of disjoint intervals each of which consists usually of two parts — a natural loop plus an acyclic structure that dangles from nodes of the loop. It then computes the data dependence relation of the loop part of every interval. Each loop is then merged into a single node. The same procedure is then repeated for the new graph until finally there is only one single node left. The computationally expensive part of the algorithm is in the step where the data dependence relation is computed. However, as in each iteration there are only a small subset of the original flow graph nodes that have to be checked, the complexity of the algorithm is very much reduced. (The detailed description of the algorithm can be found in [6].)

It is important to note that the algorithm is independent of the language used since it performs parallelism extraction from a data flow notation rather than the source language program. Therefore, the algorithm can be used to extract parallelism from a set of different languages.

The second step of the ETPN generation is to transform the 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 graph 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. That is, 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.

3.3 Design Transformation and Optimization

The transformation of the initial ETPN design representation so as to reach the register transfer level structures consists of making design decisions concerning partitioning the systems into clusters of operations, scheduling these operations, allocating data path to perform the operations, generating their control circuits, etc, which are a set of highly interdependent problems. A systematic way to make such decisions is to search for all possible transformation sequences, compare their final results, and select the best one, which is an *NP*-complete problem. Therefore, it is impossible to provide a globally optimal solution to all of these problems for a practical VLSI design.

To address the complexity problem, the synthesis process is usually partitioned into a sequence of subprocesses, namely, global optimization, operation scheduling, data path allocation, control allocation, module binding, etc. These subprocesses are then performed one by one in some order with possible iterations [3, 8, 19]. One of the problems with such a vertical approach is that optimization trade-offs between different subprocesses can not be made systematically.

For example, if the data path allocation is done first and during the control allocation process it is found that some design constraints cannot be satisfied unless the data path be reconfigured, either a very expansive back-tracking facility (due to the complex design data structure) must be provided or the synthesis process has to be restarted from the beginning [11]. Most of the behavior entry synthesis systems which use a vertical approach have concentrated on the synthesis of general-purpose computer architectures where the data path allocation problem dominates the synthesis process. Consequently, the integration of the control allocation problem into the data path design phase is not critical.

We have attempted to address the design of systems for a wide class of applications from CPU based architectures to special hardwares where the control allocation is at least as important as the data path design. We propose a horizontal manner to design such systems; instead of partitioning the synthesis process, we integrate them together [16]. That is, our synthesis algorithm performs operation scheduling, data path allocation, and control allocation at the same time. To reduce the complexity, we partition the designed system into a set of small parts. The synthesis algorithm works iteratively on one small part of the designed system at a time. To guarantee global optimization, we need a

systematic way to select parts to work on at each synthesis step. This is accomplished by a critical path analysis procedure.

The horizontal synthesis algorithm works as follows: The algorithm inputs the initial ETPN representation which is the directly translated version of the behavioral description. It will then repeat the following process. First find the critical path in the control Petri net which determines the overall performance of the design. Design decisions will then be made about the scheduling of the operations associated with the critical path as well as whether these operations will share the same data path units/control circuits. The operations not belong to the current critical path will then be considered in the order of similarity to the operations associated with the critical path. The algorithm then repeats the above process, trying to find the next critical path with the new ETPN representation. This process continues until the given design constraints have been met and/or no more optimization can be done.

In the above synthesis process, late design decisions could override the effects of earlier ones. For example, a decision to share one data path unit by two operations can be followed by another decision to split the data path unit into two duplicated ones in order to increase the degree of concurrency to meet performance requirement. Therefore, back-tracking is not necessary. When the synthesis has been concluded, the algorithm will translate the ETPN description into a structural description consisting of registers, operators, their connection, their control circuits, and a control signal generation mechanism.

3.4 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 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 and data part such that each block can be replaced by a single place or data node; in this way, certain sequences of actions are considered as indivisible or instantaneous actions. The stepwise abstraction phase reduces the number of the nodes in the net so as to reduce the time required for the final partitioning algorithm which is computationally expensive.

The second part of the partitioning procedure is based on a construction algorithm which transforms the ETPN representation into a single graph notation with cost measurement on edges and vertices [14]. A divide and conquer method is then used to perform the partitioning operation. The graph is first divided into two submodules. Each submodule is then divided into two sub-submodules using the same two-way partitioning algorithm. This process is continued until sufficiently small modules are achieved.

The two-way partitioning algorithm consists of two parts; a constructive part for preliminary partitioning, and an optimization part for iterative improvement of the solution. The preliminary clustering is done by function-similarity grouping. That is, the vertices are divided into the sets of function types: CPU-oriented, memory-oriented, register files, communication devices, etc. Each function type will then be placed in the same cluster.

The improvement process employs a heuristic method which identifies one pair of vertices from the two clusters at a time such that exchange of the pair will benefit the partition in a most promising way. When a sequence of such vertex pairs is found to be able to improve on the current partitioning, exchange will be carried out and then a new sequence of vertex pairs will be found. This process is continued until no more exchange of the vertices can improve on the cost of the partitioning.

3.5 Synthesis of Control Structures

The control part of the ETPN design representation is expressed by the timed Petri net notation. Such a control structure can be implemented in different ways. For example, it can be implemented by an asynchronous logic arrays proposed by [12] or by dedicated circuits proposed by [1]. Both of these approaches implement the parallelism inherent in the Petri net using asynchronous structures (very often based on request/acknowledge sequences).

In our approach we do not assume that the whole control structure should be implemented using asynchronous logic. Within each modules of a system, for example, a synchronous control strategy can be used, even though communication between modules will be carried out asynchronously. Based on this assumption, we will first discuss the synchronous implementation of a Petri net control structure represented by the ETPN notation. In the current implementation, the timed Petri nets are implemented as microprograms which gives us more flexibility and ability to deal with the complexity of VLSI systems.

The proposed method for microprogram generation takes as an input a normalized Petri net. A normalized Petri net is one which has the time associated with every place (describing the period of time during which the token will stay in the place) equal to one time unit. If the Petri net is not normalized it is easy to normalize it by splitting places [5]. The output from the algorithm is a state transition diagram which represent the

synchronous control flow. Each state of the state transition notation consists of two parts: the encoded control signals (the set of places forming the state) and the set of next states with encoded conditions for their selection. In this form the algorithm output can be directly used for generating a microprogram or a finite state machine implementation.

The algorithm for microprogram generation is divided into three steps. First, the normalized timed Petri net is transformed into a microprogram graph. This microprogram graph is then analyzed and reconstructed in order to reduce the complexity of the microprogram that will be generated in the last step of the algorithm.

The microprogram generation problem is similar to the problem of generating the reachability tree or coverability graph from a Petri net [18]. For the ETPN Petri net representation, however, the problem is simpler since we consider only safe Petri nets, i.e, the Petri nets which can not posses more than one token in a place during its execution sequence. The microprogram graph is a directed graph which serves as an intermediate representation between the normalized timed Petri net and the microprogram. The node of a microprogram graph represents a micro-instruction and the arcs model possible control flows from one micro-instruction to another. Each arc is guarded by a condition and a node (called a state) is a set of places which can possess tokens simultaneously, or in another words, whose execution order is not enforced by the Petri net.

The microprogram graph obtained from the algorithm usually contains nodes which will never be executed because the arcs leading to them will never become true. This is due to the generation of all possible combinations of enabled transitions. It is obvious that a direct implementation of such a graph into a microprogram can be very inefficient and complicated; sometimes even impossible. A set of reduction and optimization steps must, therefore, be used to reduce the complexity of a generated graph before it is coded into a microprogram.

The main goal of the reduction and optimization phase is to remove all nodes which can not be executed due to its execution condition remaining false all the time. This step makes use of the semantics of the Petri net description, particularly of the knowledge of the new created guarded conditions. The next aspect of this phase is to make an optimization which takes into account the knowledge of the whole design (data part and control part). We can change some parts of the whole circuit so as to simplify its microprogram without changing its semantics. An example of such an operation is a force synchronization of two parallel paths to reduce the number of control places thus the number of states in the microprogram.

3.6 Related Work

Other examples of behavior entry synthesis systems are CMU-DA [2], USC-ADAM [11], and MIMOLA [9]. The CMU-DA project has focussed almost entirely on the synthesis of data paths for general-purpose computer architectures. It is actually a group of

sub-projects, each of which focussed on one small subproblem of the synthesis process. For example, [20] proposed a clique-partitioning based allocation algorithm for registers, operators and buses in data path generation. The reported performance of the individual algorithms is quite good for the special set of test cases. However, it is not clear how well these algorithms work together.

USC-ADAM (University of Southern California Advanced Design AutoMation) is based on a representation model derived from a data flow representation that expresses behavior in terms of operations and values augmented by the explicit representation of the control flow [4]. This synthesis model contributes to a framework into which one can embed a number of heuristic, algorithmic and mathematical programming techniques for synthesis purposes. In particular, hardware synthesis is formulated as the problem of solving a giant sets of constraints, and a mixed integer linear programming approach is used.

The basic idea of MIMOLA is to partition the design process into a sequence of subprocesses and to delay design decisions until they can not be postponed any longer. These synthesis subprocesses include statement decomposition, statement scheduling, register assignment, module selection, and optimization of interconnections and instruction word length. They are executed step by step in a vertical way as most of other synthesis systems. Nevertheless, MIMOLA allows some design trade-offs to be made across subprocess boundaries. This is done by submitting several possible solutions concerning one design aspect of a subprocess to its succeeding subprocesses when the consequences of selecting one of them depend on the decisions made in the later subprocesses.

One common feature of all these methodologies is that they have concentrated on the synthesis of centralized control architecture. Our 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. Most of previous systems also partition the synthesis process into a sequence of distinct sub-processes. Therefore, design trade-offs cannot be made across different sub-processes. CAMAD integrates the operation scheduling, control allocation and data path allocation processes. This makes it possible to make systematic design trade-offs between control structure and data path. As such, in the synthesis of systems where the control structure is tightly connected to the data path organization, our approach shows a definite advantage over previous one [16].

4. Ongoing Projects

Based on the experiences gained from the development of the design algorithms presented above, CADLAB is now working on several related research topics to extend the CAMAD design aid system. These topics deals with the problems of design specification languages, organization of parts library, and overall system integration. The main

objective of the current work is to establish a VLSI design automation environment where experiments and evaluations of different design algorithms and methodologies can be effectively carried out.

4.1 Design Specification

In the current implementation, we are using PASCAL as a design specification language. PASCAL is, however, not very powerful for hardware specification purpose and we intend to extend it with several important properties which a behavioral hardware specification language should possess. One of the properties is the possibility to specify inherent parallelism and allow parallel modules (processes) to communicate by passing messages.

The problem here is that PASCAL is a sequential language while hardware is inherently parallel. During the parallelism extraction phase we can extract some parallelism. However, as the uses are not allowed to explicitly express parallelism in their programs, the degree of parallelism extracted could be quite limited in some design cases. Our experiments have shown that the degree of parallelism, i.e., the average number of operations done at any given time, extracted from PASCAL descriptions is quite small for most of the design examples. One way to increase the parallelism in the design is for the users to specify a system as a set of concurrent processes exchanging messages via their ports. In this approach, every hardware module can be described as a process with well defined interface (set of ports) which are used by the other modules to communicate with it.

The proposed approach has much in common with the object-oriented programming style. Therefore, it also simplifies the specification task and is more natural for hardware designers than pure PASCAL style. This approach will also enforce a modular structure on the design specifications.

Further, the proposed approach gives us the opportunity to create a graphical input based on the concepts of module, port and interconnection. Such an entry to the system can be easily accepted by designers who are used to schematic entry systems.

4.2 Component Representation and Selection

A parts catalog is a collection of silicon primitives specific to a given fabrication technology. Examples include any parts suitable for a printed circuit board, e.g., TTL parts catalog or AMD's catalog of bit-slice parts, macros for gate array design and standard cells for custom silicon. Previous work has used either a procedural representation of the parts catalog, for example, the CMU-DA system where the parts catalog is implicit in the productions, or a very simple declarative structure of a single technology such as the parts file used in the first version of CAMAD. This sub-project has as its goal the enablement of declarative representation of multiple parts catalogs and their integration into the design process.

Multiple parts catalogs are very much a part of ordinary design practice. Engineers routinely mix custom or semi-custom logic (ASICs - Application Specific Integrated Circuits) with off-the-shelf parts for reasons of economics, time to market or performance. Even if the ultimate silicon compiler reduced the design cost of 100% ASIC based design to zero, fabrication costs and time would strongly argue for use of off-the-shelf parts to the extent that they were consistent with power/space/speed criteria. Furthermore, some designs may simply not be possible in a single fabrication technology, e.g., a design may need both the speed of ECL and the density of CMOS.

While the research is still in an early stage, we believe that the final system will be a hierarchy of transformation rules, some of which are technology dependent and some technology independent. For example, consider bus structures: transforming the connection between several sources and sinks of data as a bus will always imply that only one source can transmit data per time period; this is independent of implementation technology. There are then transformation rules which describe buses implemented with multiplexers, open-collector gates or tri-state devices. Finally there are rules which describe the parts in any given technology. Absence of a rule transforming a technology independent concept into a physical part should turn off those transformations for that parts catalog. This work shares many concepts with the work in expert systems on modularity and reuse of knowledge. See, for example, [10].

Multiple parts catalogs are expected to strongly affect the future evolution of CAMAD. For example, the current partitioning algorithm is not part of the horizontal optimization of the control and data parts. While this works for a single implementation technology, it fails with multiple parts catalogs. Likewise the existence of hierarchies of transformation rules will require changes to the optimization algorithms. We expect major changes to the CAMAD system as results of this.

4.3 System Integration

The CAMAD design aid system consist currently of a set of design tools which make design decisions about different aspects of the design space. We need an efficient way to coordinate different design algorithms so to to move the design towards a globally optimal solution. That is, we must development a systematic way to make design decisions in terms of a combination of the cost/performance, partitioning and testability criteria.

Most existing VLSI design environments consist of a set of design tools which have their own type of data structures. Therefore, it is very difficult to provide cooperation between different tools during the design process. Our use of an unified design representation is one means of solving some of these problems. However, we have not yet addressed the problem of efficiently storing and managing different ETPN design representations so that different designs can share their result effectively (for example, using modules of other systems already designed as parts of a new design). At the moment, the ETPN

design representation is stored externally as text files. It would be very convenient to make use of a database to hold all design data in a well-defined data format and provide a common interface to the different design tools so that they can exchange information freely via the database. Further, the database management system can also monitor the exchange of information and maintain the integrity of the stored data so as to avoid illegal operations on the design data.

In our opinion the core of the integrated design automation environment should be the database keeping information about different versions and aspects of a design description. The database's responsibility is to keep this information consistent. Different techniques can be used to fulfill this requirement. One possibility is to attach special routines to the database objects. The routines can perform two types of operations. First, there are routines which will send messages informing other routines about necessary updating. A second type of routines will receive update information and perform the appropriate actions.

5. Conclusions

We have described a VLSI design aid environment built on a unified design representation, the ETPN model. One of the features of our 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.

It is shown that automated synthesis of VLSI systems from the behavioral level can be carried out efficiently with a heuristic approach based on a formal design representation which functions as a unified design data structure. The formalization of the design representation makes it possible to make accurate design decisions and to ensure correct designs from correct specifications.

Another conclusion drawn from the current study is that synthesis of digital systems which are inherently concurrent and asynchronous requires a precise notation of concurrency and asynchrony. Partial ordering is a good notation for this purpose. Petri nets are a concise and powerful model for the expression of partial ordering amongst a set of operations, thus constituting a useful representation of VLSI hardware structures.

Finally, we have also shown that a horizontal optimization approach which integrates the operation scheduling, data path allocation, and control allocation processes produces better results than a vertical approach which separates them into three distinct optimization steps, especially with systems where the control structure is closely associated with the data path design.

References

[1] Bruck, R. and Rammig, F.J., *Synthesis of Modular Controllers from CAP/DSDL*

(DACAPO) Descriptions, Proc. Computer Hardware Description Languages and their Applications, IFIP 1985.

[2] Director, S., Parker, A., Siewiorek, D. and Thomas, D. Jr., *A Design Methodology and Computer Aids for Digital VLSI Systems*, IEEE Trans. Circuits and Systems, Vol.28, No.7, 1981, pp.634-645

[3] Hafer, L. and Parker, A., *Automated Synthesis of Digital Hardware*, IEEE Transactions on Computers, Vol.31, No.2, Feb. 1982

[4] Knapp, D. and Parker, A., *A Design Utility Manager: the ADAM Planning Engine*, Proc. 23rd IEEE/ACM Design Automation Conf., 1986, pp.48-54

[5] Kuchcinski, K. and Peng, Z., *Microprogramming Implementation of Timed Petri Nets*, Integration, the VLSI Journal, Vol.5, No.2, 1987, pp.133-144

[6] Kuchcinski, K. and Peng, Z., *Parallelism Extraction from Sequential Programs for VLSI Applications*, Microprocessing and Microprogramming, the Euromicro Journal, Vol. 23, 1988, pp.87-92

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

[8] Marwedel, P., *The MIMOLA Design System: Tools for the Design of Digital Processors*, Proc. 21st Design Automation Conf., 1984

[9] Marwedel, P., *A New Synthesis Algorithm for the MIMOLA Software System*, Proc. 23rd Design Automation Conference, 1986, pp.271-277

[10] Nordin, H., *Reuse and Maintenance Techniques in Knowledge Based Systems*, Research Report, LITH-IDA-R-87-18, Dept. of Computer and Information Science, Linköping University, 1987

[11] Parker, A., Pizarro, J. and Mlinar, M., *MAHA: A Program for Datapath Synthesis*, Proc. 23rd IEEE/ACM Design Automation Conf., 1986, pp.461-466

[12] Patil, S., *Micro-Control for Parallel Asynchronous Computers*, Proc. Euromicro Workshop, Nice, June 1975

[13] Peng, Z., *Synthesis of VLSI Systems with the CAMAD Design Aid*, Proc. 23rd ACM/IEEE Design Automation Conf., Las Vegas, Nevada, 1986, pp.278-284

[14] Peng, Z., *Construction of Asynchronous Concurrent Systems From Their Behavioral Specifications*, Proc. 10th World Computer Congress, Dublin, Ireland, 1986, pp.859-864

[15] Peng, Z., *A Formal Methodology for Automated Synthesis of VLSI Systems*, Ph.D. Dissertation, No. 170, Dept. of Computer and Information Science, Linköping University, 1987

[16] Peng, Z., *A Horizontal Optimization Algorithm for Data Path/Control Synthesis*,

Proc. IEEE International Symposium on Circuits and Systems, 1988, pp.239-242

[17] Peng, Z., *Semantics of a Parallel Computation Model and its Applications in Digital Hardware Design*, Proc. International Conference on Parallel Processing, 1988, pp.69-73

[18] Peterson, J. L., *Petri Net Theory and the Modeling of Systems*, Prentice-hall, Englewood Cliffs, New Jersey, 1981

[19] Thomas, D., Hitchcock, C., Kowalski, T., Rajan, J. and Walker, R., *Automatic Data Path Synthesis*, Computer, IEEE, 1983, pp.59-70

[20] Tseng, C. and Siewiorek, D. P., *Automated Synthesis of Data Paths in Digital Systems*, IEEE Trans. on Computer-Aided Design, Vol.5, No.3, July 1986, pp.379-395

Avdelning, institution, fakultet
Division, department, faculty

Department of Computer and
Information Science
Linköping University

ISBN:

ISSN: ISSN-0281-4250

Rapportnr:
Report no: LiTH-IDA-R-88-39

Upplagans storlek:
Number of copies:

Datum:
Date: September 1988

Projekt:
Project:

Titel:
Title: CAMAD: A Unified Data Path/Control Synthesis Environment

Författare:
Author Zebo Peng, Krzysztof Kuchcinski and Bryan Lyles

Uppdragsgivare:
Commissioned by:

Dnr.:
Call no:

Rapporttyp:
Kind of report:

- ☐ Examensarbete/Final project
- ☐ Delrapport/Progress report
- ☐ Reserapport/Travel report
- ☐ Slutrapport/Final report
- ☒ Övrig rapport/Other kind of report

Rapportspråk:
Language:

- ☐ Svenska/Swedish
- ☒ Engelska/English
- ☐ _____

Sammanfattning (högst 150 ord):
Abstract (150 words):

Abstract: This paper describes a set of design tools built at Linköping University to support automated synthesis of integrated circuits. They consists of algorithms for 1) automatic generation of unified design representations from high level behavioral descriptions, 2) synthesis of optimal data path and control, and 3) partitioning of VLSI systems into asynchronous modules. The set of design tools is integrated to form a unified design environment based on a design representation model derived from timed Petri net notation. Several ongoing projects are being carried out to enrich the design environment and will also be discussed in this paper.

Nyckelord (högst 8):
Keywords (8):

Bibliotekets anteckningar

A Selection of Previous Research Reports.

LiTH-IDA-R-88-34 **Arne Jönsson, Nils Dahlbäck:** Talking to a computer is not like talking to your best friend. Also in *Proc. of the First Scandinavian Conference on Artificial Intelligence*, March 9-10, 1988, Tromsö, Norway

LiTH-IDA-R-88-32 **Erik Sandewall:** An Approach to Non-Monotonic Entailment. To appear in *Methodologies for Intelligent Systems*, vol II, *Proc. of ISMIS 1988 conference*, Torino, Italy, 1988, Z.W. Ras and L. Saitta (eds), *North Holland*, 1988, pp. 391-397.

LiTH-IDA-R-88-31 **Erik Sandewall:** The Semantics of Non-Monotonic Entailment Defined Using Partial Interpretations.

LiTH-IDA-R-88-30 **Erik Sandewall:** Future Developments in Artificial Intelligence. Also in *Proc. of European Conference on Artificial Intelligence (ECAI)*, Munich, August, 1988.

LiTH-IDA-R-88-29 **Erik Sandewall:** Non-Monotonic Entailment for Reasoning about Time and Action. Part III: Decision Procedure.

LiTH-IDA-R-88-28 **Erik Sandewall:** Non-Monotonic Entailment for Reasoning about Time and Action. Part II: Concurrent Actions.

LiTH-IDA-R-88-27 **Erik Sandewall:** Non-Monotonic Entailment for Reasoning about Time and Action. Part I: Sequential Actions.

LiTH-IDA-R-88-26 **Lin Padgham:** NODE: A Database for Use by Intelligen Systems. Also in *Proc. of ISMIS '88,* Torino Italy, October 12-15, 1988.

LiTH-IDA-R-88-23 **Johan Fagerström, Yngve Larsson:** Two Contributions on Debugging Distributed Systems. Also in *Proc. of ACM SIGPLAN/SIGOPS, Workshop on Parallel and Distributed Debugging*, Madison, Wisconsin, May 5-6, 1988.

LiTH-IDA-R-88-22 **Johan Fagerström:** Design and Test of Distributed Applications. Also in *Proc. of the International Conference on Software Engineering, ICSE*, Singapore, April 1988.

LiTH-IDA-R-88-21 **Johan Fagerström, Lars Strömberg:** A Paradigm for Distributed System Design and Test. Also in *Proc. of COMPCON'SPRING 88*, San Fransicso, Febr-March 1988.

LiTH-IDA-R-88-19 **Lin Padgham, Ralph Rönnquist:** From a Technical to a Humane Environment: A Software System Supporting Cooperative Work. Also in *Proc. of the GDI Internationa Conference on USER INTERFACES,* Rüschlikon, Switzerland, October 20-21, 1986.

LiTH-IDA-R-88-18 **Nils Dahlbäck:** Mental Models and Text Understanding - a Commented Review.

LiTH-IDA-R-88-17 **Bengt Lennartsson:** Multi-Level Software Architectures - Industrial Experiences and Research Directions. This report is a revised version of a paper in *Advance Working Papers of The Second International Workshop on Computer-Aided Software Engineering.* Cambridge, Massachusetts. July 12-15, 1988.

LiTH-IDA-R-88-16 **Lin Padgham:** A Model and Representation for Type Information and Its Use in Reasoning with Defaults. Also in *Proc. of AAAI'88, American Association for Artificial Intelligence*, 1988.

LiTH-IDA-R-88-15 **Lin Padgham, Ralph Rönnquist:** LINCKS: An Imperative Object Oriented System. Also in *Proc. of the 20th Annual Hawaii Int. Conf. on System Sciences*, Hawaii, 1987.

LiTH-IDA-R-88-13 **Erik Tengvald:** Ett kartorienterat planeringssystem för autonoma farkoster, en design diskussion.

LiTH-IDA-R-88-12 **Erik Sandewall, Johan Hultman, Erik Tengvald:** Software Architecture and Programming Paradigms for Robotics Applications.

LiTH-IDA-R-88-11 **Mats Wirén:** An Incremental Chart Parser for PATR.

LiTH-IDA-R-88-10 **Mats Wirén:** A Control-Strategy-Independent Parser for PATR. Revised version of paper appearing in *Proc. of First Scandinavian Conf. on Artificial Intelligence*, Tromsö, Norway, March 9 - 11, 1988.

LiTH-IDA-R-88-09 **Peter Fritzson:** Incremental Symbol Processing.

LiTH-IDA-R-88-08 **Erik Sandewall:** Formal Semantics for Reasoning about Change with Ramified Causal Minimization.

The Department of Computer and Information Science
Linköping University

organizes undergraduate and graduate studies in *Computer Science*, *Telecommunication and Computer Systems*, and *Administrative Data Processing*. Research activities have an emphasis on advanced software technology and computer systems design and are organized in a number of research laboratories:

- *ACTLAB - Laboratory for Complexity of Algorithms*, which is concerned with the design and analysis of efficient sequential and parallel algorithms, and complexity theory, especially in the areas of computational geometry, data structures on bounded domains and graph algorithms.
- *ASLAB - Application Systems Laboratory,* which studies design of advanced support systems for interactive use of computers, including tools for automated construction of applications software.
- *CADLAB - Laboratory for Computer-Aided Design of Electronics*, which concentrates its research activities around tools for integrated development of hardware and software, graphics-based modelling and simulation techniques.
- *LIBLAB - Laboratory for Library and Information Science,* which studies methods for access to documents and the information contained in the documents, concentrating on catalogs and bibliographic representations, and on the human factors of library use.
- *LOGPRO - Laboratory for Logic Programming*, which concentrates its research activities around foundations of logic programming, relations to other programming paradigms and methodology.
- *NLPLAB - Natural Language Processing Laboratory*, which conducts research related to the development and use of natural language interfaces to computer software.
- *PELAB - Programming Environments Laboratory*, which works with design of tools for software development, specific functions in such tools and theoretical aspects of programs under construction.
- *RKLLAB - Laboratory for Representation of Knowledge in Logic*, which covers issues and techniques such as non-monotonic logic, probabilistic logic, modal logic and truth maintenance algorithms and systems.

Research Reports 1988

LiTH-IDA-R-88-39 **Zebo Peng, Krzysztof Kuchcinski, Bryan Lyles:** CAMAD: A Unified Data Path/Control Synthesis Environment. Also in *Proc. of the IFIP TC-10 Conference on Design Methodologies for VLSI and Computer Architecture*, Pisa, Italy, September 19-23, 1988.

LiTH-IDA-R-88-38 **Zebo Peng:** Let's Design Asynchronous VLSI Systems. Also published in *Microprocessing and Microprogramming, the Euromicro Journal, Vol. 24, 1988.*

LiTH-IDA-R-88-37 **Tony Larsson, Zebo Peng:** A Relational Approach to VLSI Design Databases. Also in *Proc. of the 36th ISMM International Conference on Mini and Microcomputers and Their Applications*, Sant Feliu de Guixols, Spain, June 27-30, 1988.

LiTH-IDA-R-88-36 **Zebo Peng:** Semantics of a Parallel Computation Model and its Applications in Digital Hardware Design. Also in *Proc. of the 1988 International Conference on Parallel Processing,* Pennsylvania State University, August 15-19, 1988.

LiTH-IDA-R-88-35 ...orithm for Data ... *1988 IEEE International* ... i Univ. of Technology,

LINKÖPINGS UNIVERSITET
30580 007213056