ICOT Technical Report: TR-375

TR-375

co-LODEX : A Cooperative Expert System for Logic Design

by
F. Maruyama, T. Kakuda,
Y. Matsunaga, Y. Minoda
and N. Kawato(Fujitsu)

May, 1988

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato ku Tokyo 108 Japan

(03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

co-LODEX: A Cooperative Expert System for Logic Design

Fumihiro Maruyama, Taeko Kakuda, Yusuke Matsunaga,
Yoriko Minoda, and Nobuaki Kawato

FUJITSU LIMITED
1015 Kamikodanaka, Nakahara-ku
Kawasaki 211, Japan

Distributed artificial intelligence (DAI) is concerned with cooperative solution of problems by distributed agents. Motivated by the conventional distinction between datapath design and control design, we study a DAI system for VLSI logic design. Assumption-based reasoning is also useful for automating the logic design process, which inevitably involves tentative decisions.

In this paper, we present a cooperative expert system for logic design, Co-LODEX, which features DAI and assumption-based reasoning. Co-LODEX consists of three distributed agents, two of which correspond to datapath design and control design, respectively. Each of the two agents iterates the refine-evaluate-redesign cycle, under constraints on conflicting criteria, area and time. If it itself cannot clear some constraint, it asks the other for changes. This is where cooperation takes place in an attempt for the agents together to satisfy all the given constraints.

We treat design decisions as assumptions since they are sometimes tentative and can be retracted later. We think of redesign as contradiction resolution by viewing constraint violation as contradiction. When a constraint violation is detected during evaluation, the redesign mechanism based on assumption-based reasoning is invoked. Assumption-based reasoning is of more importance when cooperation comes in, because some decisions are forced to be retracted by the other agents. Justifications for constraint violations called nogood justifications (NJ's) play a central role in the redesign mechanism. Co-LODEX carries out redesign by expanding and generating NJ's on the hierarchy that represents the circuit under design.

1. Introduction

The rapidly-progressing VLSI technology requires CAD systems that can produce designs of good quality within a short period. Although rule-based expert systems have great potential, at present their outputs are not satisfactory compared to those by experienced designers. One of the most serious problems seems to be the lack of mechanisms for complementing various kinds of knowledge and supporting the iterative design cycle: refine, evaluate, and redesign.

Distributed artificial intelligence (DAI) is concerned with cooperative solution of problems by distributed agents [Smith 85]. It is particularly effective for problems without any single global goal. Since logic design involves mutually conflicting criteria, for example, area and time, DAI systems for logic design are worthy to study. Previous approaches along this direction include ULYSSES [Bushnell 86] and the work of Brewer and Gajski [Brewer 86].

ULYSSES is a design environment consisting of existing CAD tools as knowledge sources (KS's). Knowledge sources communicate with each other through files within a global database called the blackboard. ULYSSES is a realistic approach to tool integration as it makes good use of existing tools. However, automating the iterative design cycle is beyond its scope.

Brewer and Gajski view that design at one level becomes a specification for the lower levels and propose a design paradigm of a set of communicating expert systems, each of which corresponds to each of the levels of abstraction. Although there are two directions, down for constraint propagation and up for failure reporting, design is limited to a single stream and is not flexible enough.

We recognize two streams of design, datapath design and control design, in the logic designers' community. Datapath design starts off with a block diagram and designs each functional component in a hierarchical manner, that is, designs a component with subcomponents and subcomponents with sub-subcomponents, and so on. Control design begins with a behavioral specification, establishes finite-state machines conforming to the specification, realizes them with flip-flops, and designs circuits that generate control signals. Design progresses in these two streams with interaction between them. We propose a cooperative expert system for logic design, co-LODEX, which contains two agents that correspond to datapath design and control design, respectively. Each of the two agents iterates the refine-evaluate-redesign cycle, under constraints on conflicting criteria, area and time. If it itself cannot clear some constraint, it asks the other for changes. If asked for changes, an agent tries to make some changes in an attempt for the agents together to satisfy all the given constraints.

In general, the consequence of a design decision is not always clear when it is made. On later evaluation against constraints, it sometimes happens that the decision was wrong in the particular environment and must be retracted. Since no advance is made in design as long as no decision is made, usually the most plausible alternative at that time is selected. With the results of the decision added to the body of design data, the next decision is made. When the design gets to the point where evaluation is possible, it is evaluated against constraints. Design goes like this.

Assumption-based reasoning is a type of reasoning that uses not only facts but also assumptions, which have been assumed to

hold but can be retracted at a later time [de Kleer 86]. Justification, originally introduced for truth maintenance [Doyle 79], is a key to manipulating information containing assumptions. It is noteworthy that justification is a logical concept. While the foundations of assumption-based reasoning are given in logic [Reiter 87], a direct application of logic to design is proposed [Finger 85].

We incorporate assumption-based reasoning into co-LODEX for the following three reasons:

(1) Design decisions are sometimes tentative and can be retracted later.

(2) Some decisions are forced to be retracted by the other agents in cooperation.

(3) Constraints are also subject to change.

Design decisions and constraints are treated as assumptions in co-LODEX. We think of redesign as contradiction resolution by viewing constraint violation as contradiction. When a constraint violation is detected during evaluation, the redesign mechanism based on assumption-based reasoning is invoked. Justifications for constraint violations called nogood justifications (NJ's) play a central role in the redesign mechanism. They are represented as conjunctions of assumptions and conditions about area or time. Co-LODEX carries out redesign by expanding and generating NJ's on the hierarchy that represents the circuit under design.

The rest of this paper is organized as follows. In the next section, we give the overview of co-LODEX focusing on its CAD aspects. Its DAI aspects and its distributed agents are described in section 3. In section 4, we discuss the redesign mechanism based on assumption-based reasoning. Our status and conclusions are given in section 5.

2. Co-LODEX Overview

Co-LODEX takes as input both a behavioral specification and a block diagram, and produces a CMOS standard cell implementation. It also accepts global constraints on area and time, against which design is evaluated. Design is carried out in two streams, datapath design from the block diagram and control design from the behavioral specification, with interaction between them. After putting their results together, co-LODEX does some optimization and outputs a CMOS standard cell circuit description in terms of standard cells and the connections between them.

The specification language for behavior is an extension of DDL [Duley 68], based on temporal logic [Moszkowski 86]. Figure 1 shows the specification of GCD (Greatest Common Divisor) [Camposano 87].

```
FUNCTION: main: clk;
  idle::
    STOP(rst=0), x<-xi, y<-yi, GOTO loop;
  loop::
    IF(x=y) THEN(ou:=x, GOTO idle)
            ELSE(IF(x<y) THEN(y<-y-x)
                         ELSE(x<-x-y),
                 GOTO loop);
FEND;
```

Figure 1 Example Behavioral Specification

There are two intervals, idle and loop, which are counterparts of states in DDL but are not limited to one clock cycle in length. 'STOP(rst=0)' means that interval idle is finished when rst is equal to 0. '<-' means register transfer and ':=' means terminal connection. The rest will be self-explaining.

The user is supposed to enter a block diagram through the graphics editor of co-LODEX by selecting and placing functional components and connecting them. A block diagram corresponding to

the behavioral specification in Figure 1 is shown in Figure 2. COMPARATOR for a comparator, SUB for a subtracter, MUX for a multiplexer, registers X and Y, input pins XI and YI, and output pin OU are functional components.

Constraints on area are expressed in inequalities about basic cell count. 'The total basic cell count must not exceed 1300' is an example. Constraints on time are expressed in inequalities about delay or clock cycle. 'The clock cycle must not be longer than 120ns' is an example. We think that constraints themselves are subject to change, because design is an exploratory process and is sensitive to the circumstances. The designer might want to explore another possibility by strengthen or relax some constraints, instead of sticking to the first result he or she achieved. Co-LODEX allows the user to add, retract, or restore constraints by storing all the constraints that have been given so far. With the constraints changed, it comes up with another solution by keeping intact the portion that has nothing to do with the changed constraints in order to make turnaround short.

3. Cooperative Distributed Agents

In logic design, design is evaluated based on mutually conflicting criteria, for example, area and time: there is no single global goal. In other words, it is impossible to serialize the designs conforming to the specification. Instead, the designer comes up with one in the permissible region in the virtual multi-dimension space with the axes corresponding to the criteria.

Our approach to this problem is as follows. Each distributed agent executes the design on it own to produce a partial solution, referring to the relevant global constraints. These partial solutions are combined to form a complete design. Some adjustments based on the results of evaluation may be necessary, which would be possible through cooperation between the agents.

3.1 Configuration and Communication

Figure 3 shows the configuration of co-LODEX. Co-LODEX has three agents, two of which we mentioned earlier. The third agent, User Interface Agent, is responsible for transferring information to and from the user.

Datapath Design Agent produces as partial solution the whole datapath with each functional component implemented by CMOS standard cells, while Control Design Agent designs circuits that control the flow of execution and each functional component of the datapath. These two partial solutions are linked at the control terminals of the functional components as illustrated in Figure 4.

Here are a few examples of how these two agents influence each other. First, if a delay constraint is so strict that the components on a path do not satisfy the constraint, Control Design Agent asks Datapath Design Agent to redesign some of them.

Secondly, if Datapath Design Agent cannot design components on a path fast enough to satisfy a clock cycle constraint, it asks Control Design Agent either to give up the path and take another or to break the operation along the path into sub-operations, each of which is executed in one cycle.

Finally, if Datapath Design Agent has to remove one of the duplicate components because of the area constraint, it asks Control Design Agent to change control so as to make sure that the reduced set of components is enough without any conflict, or double use.

In order to cooperate as seen in the above examples, the two agents exchange information, which includes the four types shown in Figure 5:

1. Request for change is issued with the failed constraint if Datapath Design Agent is unable to satisfy a constraint. What to change is left to Control Design Agent.
2. An alternative datapath is informed if Datapath Design Agent has replaced the current datapath with it.
3. Since Control Design Agent determines the accurate timing of each operation in the behavioral specification by establishing finite-state machines conforming to the specification, it generates internal constraints on time, according to those specified by the user. Although it may be logical that Control Design Agent itself evaluates the design in a case like the first example, it is expensive as it involves transfer of a large amount of design data between agents. Actually, instead, Control Design Agent sends the internal constraints on time to Datapath Design Agent asking for evaluation as soon as it generates them.
4. A new datapath is sent to Datapath Design Agent, that has been made possible by the change of control.

One of the distinctive pictures of cooperation in co-LODEX seems to emerge from the above. One agent comes up with one partial solution and makes the others check it. In the meantime, the agent proceeds to the next task. If the check is successful, the partial solution will continue to be valid at least until any contradiction occurs. Otherwise, the agent comes back to the previous task and tries to find another solution. For example, Control Design Agent can tell Datapath Design Agent to evaluate, first of all, "critical paths" in order to know as early as possible whether the control it has designed is feasible or not, while it begins to implement the control. A negative result is informed as an urgent message, which is handled right away and causes to design control in an another way.

3.2 Datapath Design Agent

The purpose of Datapath Design Agent is to design all the functional components forming the datapath. It starts off with the block diagram and designs each functional component in a hierarchical manner all the way down to CMOS standard cells. The design is done by applying rules in the KS's, under constraints on basic cell count and those on delay or clock cycle. Component Design KS contains rules for designing a component with subcomponents. Figure 6 shows an example for designing an n-bit subtracter with an n-bit adder and an n-bit one's complement. Technology Mapping KS takes a component and implements it with the CMOS standard cells in the library. Figure 7 shows an example of implementing 4n-bit adder with n 4-bit carry-lookahead-adder (CLA) cells.

3.3 Control Design Agent

Control Design Agent establishes finite-state machines con-

forming to the specification. At this point, every detail of timing is determined as far as synchronous portion is concerned. Control Design Agent generates internal constraints on time by examining which path must be covered withing one clock cycle. It sends these constraints out to Datapath Design Agent. It then realizes the finite-state machines with flip-flops and designs circuits that generate control signals to the components. Among its KS's are State Machine Design KS and State Assignment KS.

3.4 User Interface Agent

User Interface Agent is responsible for transferring information to and from the user. By information here we mean the following:

1. The behavioral specification is input in a text form.
2. The user is supposed to enter a block diagram through the graphics editor of User Interface Agent by selecting and placing components and connecting them.
3. User Interface Agent provides a menu-based dedicated window for specifying constraints in an inequality form. It also enables the user to retract and restore them by storing all the constraints that have been given so far.
4. The user can design any component manually through the graphics editor of User Interface Agent and forces co-LODEX to use it.

There seems to be an aspect of cooperation between the user and co-LODEX through User Interface Agent, focusing on the third point, the user exploring possibilities by strengthening or relaxing constraints, and the fourth point, the user volunteering to design some of the components.

4. Redesign Based on Assumption-Based Reasoning

As mentioned earlier, we regard design decisions as assumptions. Assumption-based Truth Maintenance System (ATMS) [de Kleer 86] enumerates all the assumptions in advance and examines all the combinations of them. In design, however, we are not interested in all the combinations of all the design decisions, because whether a decision has its meaning or not depends on the decisions made earlier. In the example of Figure 2, for instance, how to construct an adder has no meaning if the subtracter was designed without adders.

Besides, we want to talk about quantity, for example, the total basic cell count of the adder and the one's complement is not less than 432, not just combinations of assumptions. We notice that the two criteria we are interested in, area and time, are additive in the sense that the quantity of the whole is the total sum of that of its constituents. The delay along a path on the circuit, for example, can be attributed to that of the components on the path. Additivity allows us to break a condition concerning the whole into those concerning its parts. In order to do so hierarchical structure is of great help.

We propose a redesign mechanism based on nogood justifications (NJ's). They are represented as conjunctions of assumptions and conditions about area or time. Co-LODEX carries out redesign by expanding and generating NJ's on the hierarchy that represents the circuit under design.

4.1 Hierarchical Design Description

Design objects concerning datapath are represented in a hierarchy. Figure 8 shows a part of the hierarchy corresponding to Figure 2. There are two types of nodes, component nodes (ovals) and alternative nodes (rectangles). A component node

represents each component in the datapath. There is a special component node called the datapath node that corresponds to the whole datapath. An alternative node represents each alternative, contains information about the connection between the subcomponents, and has the subcomponent nodes as children. Figure 8 shows as follows. The current datapath, DATAPATH1, is one shown in Figure 2. The subtracter is composed of an adder and a one's complement like in Figure 6. The 32-bit adder consists of eight 4-bit CLA cells connected in series like in Figure 7. An alternative is called either "in" or "out", according to whether it is adopted or discarded, respectively. Each component node has at most one alternative node as "in" alternative. Other alternative nodes, which are invalid at the moment, are stored in the "out" alternative list for later restoration due to the change of environment.

4.2 Nogood Justification

A nogood justification (NJ) is a logical expression that must not hold during the design. Satisfying any of the NJ's means constraint violation and invokes the redesign mechanism. An NJ is in a conjunctive form. Each conjunct is one of the following:

1. a design decision, or an alternative,
2. a constraint,
3. a condition about basic cell count
4. a condition about delay.

Each NJ is put at one of the alternative nodes.

Among NJ's are what we call default NJ's. Suppose a constraint saying 'the total basic cell count of the datapath must not exceed 1300' is valid. The following default NJ is written at DATAPATH1 in Figure 8:

(# of basic cells < 1300) & DATAPATH1

& (SUB + COMPARATOR + ... > 1300) (1)

where the first conjunct is the constraint, the second an alternative, the third a condition about the total sum of the basic cells of the components. It is equivalent to the original constraint in the sense that any design violating the constraint with DATAPATH1 as datapath satisfies it. Default NJ's allow us to reduce evaluation against constraints to checking on NJ's. Now we are ready to define expansion and generation of NJ's.

4.3 NJ Expansion

NJ expansion is used to specialize an NJ in an attempt to narrow a scope down for contradiction resolution, or redesign. Expansion of an NJ with respect to a component node is defined as follows:

1. removing the component's contribution from every condition with the component in it,

2. making an NJ as the logical product of the conjunction obtained above, the component's "in" alternative, and the condition about the component's subcomponents, and

3. putting the resulting NJ at the alternative node of the component.

For example, expanding (1) with respect to SUB (in Figure 8) may give us the following NJ at SUB1:

(# of basic cells < 1300) & DATAPATH1

& (COMPARATOR + ... > 868) & SUB1

& (ADD + 1'S-COMPLEMENT > 432) (2)

This happens when the design turns out to exceed 1300 basic cells and the subtracter is selected as a candidate to be changed. Further, expanding (2) with respect to ADD will give us the following NJ at ADD1 as the 4-bit CLA cell consists of 50 basic

cells:

(# of basic cells < 1300) & DATAPATH1

& (COMPARATOR + ... > 868) & SUB1

& (1'S-COMPLEMENT > 32) & ADD1 (3)

If the one's complement circuit has no more alternative, the fifth conjunct always holds and can be deleted:

(# of basic cells < 1300) & DATAPATH1

& (COMPARATOR + ... > 868) & SUB1 & ADD1 (4)

Notice that (4) does not allow to use 4-bit CLA cells for a 32-bit adder under the condition:

(# of basic cells < 1300) & DATAPATH1

& (COMPARATOR + ... > 868) & SUB1

4.4 NJ Generation

Suppose every alternative for a component causes violation against constraints (not necessarily the same constraint). NJ generation enables us to get an NJ with no reference to the component, from a set of NJ's, each of which contains each alternative as a conjunct. If, for example, there are only three alternatives, a, b, and c, for component X, and we have NJ's, A & a, B & b, and C & c, then NJ A & B & C can be generated at the alternative node of X's parent node. This procedure is justified by resolution [Robinson 65]. The generated NJ suggests that we select a component as a candidate to be changed among those it refers to.

Suppose, in addition to (4), we have NJ's for an alternative with 2-bit CLA cells (ADD2: 256 basic cells) and for an alternative with 1-bit adder cells (ADD3: 256 basic cells):

(# of basic cells < 1300) & DATAPATH1

& (COMPARATOR + ... > 1012) & SUB1 & ADD2 (5)

(# of basic cells < 1300) & DATAPATH1

& (COMPARATOR + ... > 1012) & SUB1 & ADD3 (6)

If no alternative is available other than these three, from (4), (5), and (6), NJ generation gives us a new NJ:

(# of basic cells < 1300) & DATAPATH1

& (COMPARATOR - ... > 1012) & SUB1 (7)

This shows that the design illustrated in Figure 6 is impossible under the environment specified by the first three conjuncts. Co-LODEX would have to change either the subtracter or the environment.

4.5 Redesign Algorithm

The redesign algorithm using expansion and and generation of NJ's is outlined. The redesign mechanism is invoked when some default NJ turns out to be true for the evaluated design. The redesign algorithm begins with the NJ, from the alternative node where the NJ resides.

Step 1. If this alternative should be kept untouched, then select one of the subcomponents, expand the NJ with respect to it, go down the hierarchy by one level with the NJ expanded, and executes Step 1 again. Otherwise, go to Step 2.

Step 2. If there is any other alternative available, which can be an "out" alternative, that makes no NJ's at the ancestor nodes (including itself) true, then replace the current alternative with that and exit. Otherwise, go to Step 3.

Step 3. Generate an NJ and go up the hierarchy by one level with the NJ generated. If the NJ generated contains constraints only, fail! Otherwise, go to Step 1.

The decision on whether to replace an alternative or which subcomponent to select in Step 1 is made based on heuristics, which can be separated from the above algorithm. One of such heuristics

would be to select the largest or the slowest subcomponent.

We have proposed a redesign algorithm that is defined on the hierarchy representing the design. So it is already plugged in to the whole design process. From a pratical point of view, it is important for the user to be able to volunteer to design without disabling the system's justification mechanism. Our approach is to add "no-more-alternative" assumptions. Suppose, for example, we add a "no-more-alternative" as the fifth conjunct to NG (7) when generating it. If the user comes up with a smaller adder and enters it to the system, the system adopts it as "in" alternative and makes the assumption false. It means (7) is no more satisfied and the subtracter can be restored under the same environment as before.

5. Status and Conclusions

We are implementing co-LODEX on PSI's (Personal Sequential Inference Machines) in ESP [Chikayama 84] using the object-oriented feature and the remote object facility. Experiments and evaluation are scheduled with it completed.

Our work focuses on the cooperation between distributed agents, Datapath Design Agent and Control Design Agent, and the redesign mechanism based on assumption-based reasoning. One extension would be to add an agent for design for testability. We believe that assumption-based reasoning proves to be more effective when used in the context of DAI.

Acknowledgement

This work is based on the results of the R & D activities of the Fifth Generation Computer Systems Project of Japan. We would like to thank Mr. Fujii of ICOT for his encouragement and support.

References

[Smith 85] Smith, R. G., "Report on The 1984 Distributed Artificial Intelligence" AI Magazine, Fall 1985 (1985).

[Bushnell 86] Bushnell, M. L., Director, S. W., "VLSI CAD Tool Integration Using the ULYSSES Environment" Proc. of 23rd Design Automation Conf., pp.51-61 (1986).

[Brewer 86] Brewer, F. D., Gajski, D. D., "An Expert-System Paradigm for Design" Proc. of 23rd Design Automation Conf. pp.62-68 (1986).

[de Kleer 86] de Kleer, J., "An Assumption-Based Truth Maintenance System" Artificial Intelligence 28 (1986).

[Doyle 79] Doyle, J., "A Truth Maintenance System" Artificial Intelligence 24 (1979).

[Reiter 87] Reiter, R., de Kleer, J., "Foundations of Assumption-Based Truth Maintenance Systems: Preliminary Report" Proc. of AAAI-87, pp.183-188 (1987).

[Finger 85] Finger, J. J., Genesereth, M. R., "RESIDUE: A Deductive Approach to Design Synthesis" Tech. Rept. HPP-85-1, Stanford University (1985).

[Duley 68] Duley, J. R., Dietmeyer, D. L., "A Digital System Design Language (DDL)" IEEE Trans. Computers, Vol.C-17, No.9, pp.850-861 (1968).

[Moszkowski 86] Moszkowski, B., "Executing Temporal Logic Programs" Cambridge University Press (1986).

[Camposano 87] Camposano, P., "Structural Synthesis in The Yorktown Silicon Compiler" Proc. of VLSI'87, pp.29-40 (1987).

[Robinson 65] Robinson, J. A., "A Machine-Oriented Logic Based on the Resolution Principle" Journal of the ACM, Vol.12, No.1, pp.23-41 (1965).

[Chikayama 84] Chikayama, T., "Unique Features of ESP" Proc. of FGCS'84, pp.292-298 (1984).